3-14-2014

A Heuristic Approach to the Theater Distribution Problem

Emily K. Power

A HEURISTIC APPROACH TO THE THEATER DISTRIBUTION PROBLEM

THESIS

Emily K. Power, Second Lieutenant, USAF

AFIT-ENS-14-M-25

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT-ENS-14-M-25

A HEURISTIC APPROACH TO THE THEATER DISTRIBUTION PROBLEM

THESIS

Presented to the Faculty

Department of Operations Research

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

Emily K. Power, BS

Second Lieutenant, USAF

March 2014

AFIT-ENS-14-M-25

A HEURISTIC APPROACH TO THE THEATER DISTRIBUTION PROBLEM

Emily K. Power, BS
Second Lieutenant, USAF

Approved:

//signed//	13 Mar 2014
Jeffery D. Weir, PhD (Advisor)	Date
//signed//	13 Mar 2014
Raymond R. Hill, PhD (Reader)	Date

Abstract

Analysts at United States Transportation Command (USTRANSCOM) are tasked with providing vehicle mixtures that will support the distribution of requirements as provided in the form of Time Phased Force Deployment Data (TPFDD). An integer programming model exists to search for optimal solutions to these problems, but it is fairly time consuming, and produces only one of potentially several good quality solutions.

This research constructs a number of heuristic approaches to solving the Theater Distribution Problem (TDP). Two distinct shipping methods are examined and applied through both constructive and probabilistic vehicle assignment processes. Multistart metaheuristic approaches are designed and used in conjnction with the constructive and probabilistic approaches. Random TPFDDs of size 20, 100 and 1000 are tested, and solutions are compared to those obtained by the integer programming approach.

The heuristic models implemented in this research develop feasible solutions to the notional TPFDDs in less time than the integer program. They can very quickly identify a number of good quality solutions to the same problem.

To my mother, who instilled confidence in me when I needed it most.

Acknowledgments

I would like to thank Dr. Jeffery Weir for his outstanding guidance on this thesis research as well as the introduction to joint mobility modeling in OPER 674 which sparked my interest in this area of research. I would also like to thank Dr. Raymond Hill for serving as a reader and for providing the introduction to Heuristic Optimization in OPER 623, which provided a method for solving the problem. Next, I would like to thank Mr. Scott Percival for translating my code from Matlab into VBA, building the TPFDD generator, and providing invaluable insight. Lastly, I would like to thank my family, especially my sisters, for giving me the gifts of love and laughter every single day.

Emily K. Power

Table of Contents

Page

Abstract . . . iv

Dedication . . . v

Acknowledgments . . . vi

Table of Contents . . . vii

List of Figures . . . ix

List of Tables . . . x

List of Algorithms . . . xi

List of Acronyms . . . xii

I. Introduction . . . 1

- Background . . . 1
- Research Purpose and Objectives . . . 6
- Organization . . . 6

II. Literature Review . . . 8

- Pickup and Delivery Problem with Time Windows . . . 8
- Machine Scheduling . . . 9
- Heuristics . . . 10
  - Random Generation . . . 11
  - Problem Decomposition . . . 12
  - Constructive Approaches . . . 13
  - Local Improvement . . . 14
- Metaheuristics . . . 15
  - Multistart Constructive Approaches . . . 15
  - Tabu Search . . . 16
  - Simulated Annealing . . . 19
  - Genetic Algorithms . . . 20
- Conclusion . . . 21

Page

III. Methodology 23

Shipping Functions 25
Constructive Functions 27
Probabilistic Functions 30
Multistart Heuristics 33
Conclusion 35

IV. Testing 36

Data 36
Testing Considerations 39
Parameters 41
Results 43

V. Conclusions and Future Research 51

Summary 51
Conclusions 51
Future Study 53

Bibliography 55

Appendix: Research Poster 58

List of Figures

Figure		Page
1	End-to-End Distribution [17]	2
2	Tabu Search[13]	17
3	Tabu Status Check [13]	17
4	Example of One-Point Crossover in a Genetic Algorithm [27]	21
5	Objective Versus Best Found So Far	35
6	Frequency of Gap Length for Real World Data	38
7	Multistart Random Approach Solution Value and Best-so-Far	40

List of Tables

Table		Page
1	Notional TPFDD	24
2	Dataset A TPFDD Statistics	37
3	Comparison of Dataset A to Real World TPFDD Data	38
4	Dataset B TPFDD Statistics	39
5	Vehicle Payloads and Costs	41
6	Function Abbreviations	42
7	Function Attributes	43
8	20 Requirement Solution Values in 500 Iterations	44
9	Comparison Metrics for 20 Requirement TPFDDs	44
10	100 Requirement Solution Values in 150 Iterations	46
11	Comparison Metrics for 100 Requirement TPFDDs	47
12	Comparison Metrics for 100 Requirement TPFDDs in 500 iterations	47
13	1000 Requirement Solution Values in 60 Iterations	49
14	Comparison Metrics for 1000 Requirement Problems	49
15	20 Minute runs on 1000 Requirement Problem	50

List of Algorithms

Algorithm		Page
1	Local Search [8]	14
2	Simulated Annealing	20
3	Extra Room	25
4	Ship Early	26
5	Ship Over	27
6	Construct (On Time)	28
7	Construct (Late)	30
8	Probabilistic (On Time)	31
9	Probabilistic (Late)	32

List of Acronyms

Acronym	Definition
AMP	Analysis of Mobility Platform
DARP	Dial-A-Ride problem
EAD	earliest available date
GRASP	greedy randomized adaptive search procedure
MAL	maximum allowable lateness
MVDARP	Multiple Vehicle Dial-A-Ride problem
PDPTW	Pickup and Delivery Problem with Time Windows
POD	Port of Debarkation
POE	Port of Embarkation
RDD	required delivery date
R-TABU	reactive tabu
SMM	Strategic Mobility Modeling
TDD	time definite delivery
TDM	Theater Distribution Model
TDP	Theater Distribution Problem
TPFDD	Time Phased Force Deployment Data
USTRANSCOM	United States Transportation Command

A HEURISTIC APPROACH TO THE THEATER DISTRIBUTION PROBLEM

I. Introduction

Background

Recent budget constraints and economic crises have introduced a great deal of motivation into finding efficiencies in logistics, distribution, and transportation in the military. In an attempt to combat these budget constraints, a great deal of attention is given to distribution planning. The goal of distribution planning is to "defuse strategic problems before they become crises and resolve crises before they reach a critical stage [18]." In support of this goal, indepth analyses are conducted in reference to potential engagements for the military. A large portion of the analysis takes place at reoccurring force flow conferences held by United States Transportation Command (USTRANSCOM), where transportation feasibility and movement schedules are considered.

Distribution in the military, defined in Joint Publication 4-0, Joint Logistics, as "the operational process of synchronizing all elements of the logistic system to deliver the 'right things' to the 'right place' at the 'right time' to support the geographic combatant commander (GCC)," is carefully planned and conducted within joint operations. Distribution has been previously modeled in an end-to-end fashion that includes three major phases, depicted in Figure 1. These phases include intracontinental movement, intertheater movement, and intratheater movement. Intracontinental movement takes cargo and troops and moves them from a Point of Origin within the Continental United States (CONUS) to a Port of Embarkation (POE), or its planned exit point from CONUS. Intertheater distribution starts at the POE and brings cargo to a Port of Debarkation (POD) outside CONUS (OCONUS). Intratheater movement, the final step, is concerned with

getting the cargo from its POD in theater to its final destination. This thesis focuses primarily on the final phase of distribution: intratheater movement [17].

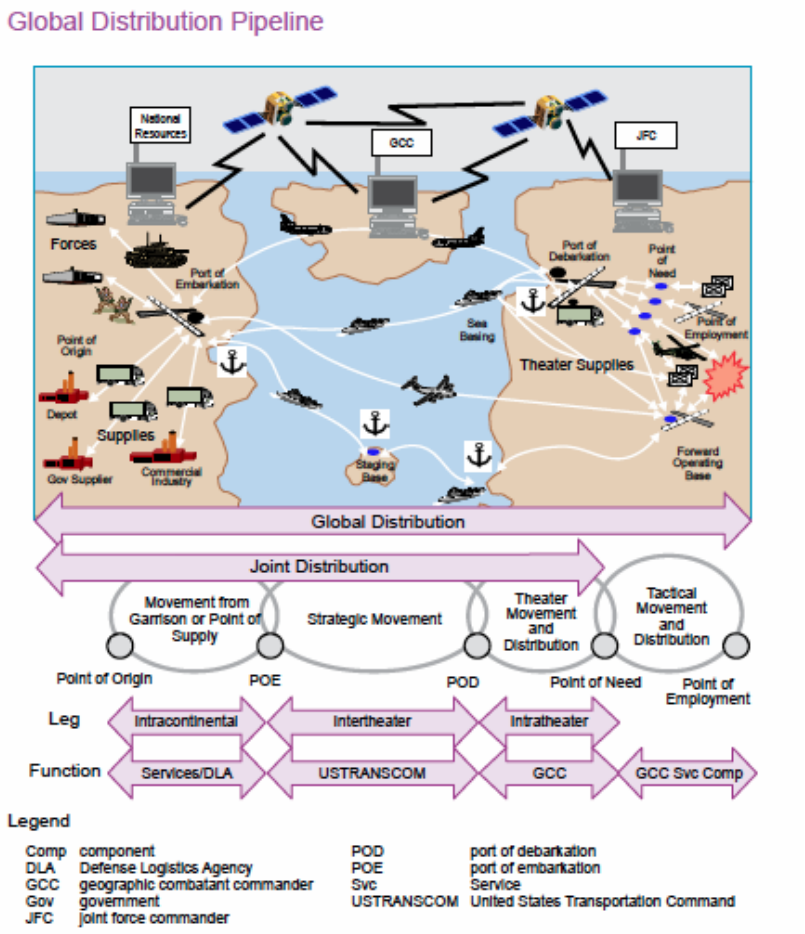


Figure 1: End-to-End Distribution [17]

Strategic Mobility Modeling (SMM) is used in the military to represent the flow of cargo and passengers in the end-to-end planning horizon. Movement of cargo is modeled from its initial origin through its POE, POD, and to its end destination or employment location. Varying levels of planning detail among these steps are considered through different models. The most basic details are considered in resource planning, which is used primarily for determining appropriate transportation assets for proposed scenarios in

the context of long-term planning. Deliberate planning introduces more detail by addressing plan feasibility in regards to specific deployment scenarios. The highest level of detail in mobility modeling is referred to as crisis action planning. Taking only days or weeks at the end of the planning horizon, crisis action planning reassesses previously drafted plans and updates them due to recent changes or additional knowledge [21]. The main focus at USTRANSCOM force flow conferences is in the first two levels of planning – resource and deliberate.

The models used in SMM require inputs to provide output results that yield insight. These inputs usually include a list of requirements, transportation resources, and scenario information. Requirements are provided in the form of Time Phased Force Deployment Data (TPFDD), which is essentially a list of cargo and passengers that need to be transported to theater. For each requirement, weight, earliest arrival date, latest arrival date, required delivery date, point of debarkation, point of embarkation, and final destination are included.

The most detailed and relevant strategic mobility model currently employed in the military is the Analysis of Mobility Platform (AMP). This particular SMM was developed to model the entire process of end-to-end distribution. The program takes in a TPFDD as input and simulates the movements. This process is extremely time consuming, and it returns results based solely on the input given. A great deal of detail is considered, but the data must be preprocessed and developed before determining the effects by judging the output from the model. AMP requires information about which cargo to put on what types of vehicles. Currently, the algorithms for assigning vehicles to requirements in a minimum cost fashion are limited. For this reason, trial and error is used regularly. Often times, it is so difficult to come up with reasonable data that solutions are accepted based solely on feasibility instead of solution quality. Finding better solutions through running several scenarios in the simulation is time consuming and computationally demanding.

In order to improve modeling capabilities in the military, a method of assigning requirements to vehicles and building a schedule for timely delivery is desired. Determining the number and type of vehicles required may dictate how overseas bases are set up for future engagements. Having the necessary beddown of vehicles in place at the start of a conflict can help ensure cost effective and timely delivery of necessary equipment to the men and women engaged in combat, possibly resulting in more efficient mission completion and success.

The problem of assigning requirements to vehicles in theater can be described in terms of a distribution network. Information from the TPFDD is used to build the network and determine additional constraints. In each theater there is a set of PODs, where cargo will initialize as well as a set of final destinations for these cargo. Due to the uncertain nature of deployments, it does not make sense to plan out specific routes between these locations. By assuming that a route exists between locations, deliveries can be scheduled based solely on origin to destination. The requirements in each Theater Distribution Problem (TDP) come directly from the TPFDD. Each item on the TPFDD has an associated weight in short tons, an availability date, and a required delivery date, along with many other parameters. Transportation modes considered between each origin-destination pair are air, rail, and road. Different costs and vehicle types are associated with using each of these modes. Additionally, each vehicle type has an associated maximum payload, speed, and loading and unloading parameters.

The TDP initializes with all requirements distributed at their respective PODs. The objective is to assign all requirements to vehicle types and schedule them on particular days in a way that minimizes the total cost of delivering all requirements in the TPFDD. Due to the nature of the military, there are a few special shipping cases that need to be addressed. First, there may be requirements in the list that are too large to fit on a single vehicle. In this case, it should be possible to split the delivery into several shipments.

Similarly, there may be several requirements at a particular location whose available and required delivery dates overlap. For this reason, vehicles should be able to process more than one requirement at a time as long as capacity constraints are not violated.

Longhorn and Kovich, analysts working at USTRANSCOM, addressed this problem by formulating an integer programming approach referred to as the Theater Distribution Model [20]. Their model attempts to find exact optimal solutions to the problem, but because the nature of integer programming and network formulation requires a large number of variables, it did not successfully accomplish this goal. For this reason, the initial TDM was further examined by Hafich, who developed a new formulation for the problem, called the Improved Theater Distribution Model. His model greatly reduced the number of unnecessary variables and was able to produce solutions, but still created a problem that took a long time to generate and solve [14].

A valid tool for providing solutions to difficult problems is a heuristic, defined by Silver in 2004 as "a method which, on the basis of experience or judgment, seems likely to yield a reasonable solution to a problem, but which cannot be guaranteed to produce the mathematically optimal solution." Because they do not require strict model assumptions, heuristic search methods can allow for better depiction of real world parameters. Heuristic methods are often easier to explain and understand, are more robust to variations in problem characteristics, and perform faster than strict optimization routines [29].

Basic heuristic methods include random generation, constructive methods, and local improvement or neighborhood search methods. These methods are useful, but can be expanded by adding rules to guide them in an iterative improvement fashion. This strategy is commonly referred to as a metaheuristic approach. Some of the more popular metaheuristic searches are simulated annealing, genetic algorithms, tabu search, and ant colony algorithms. Several of these metaheuristic approaches have been developed and

used for military applications such as weapons assignment problems, UAV routing problems, aircraft loading problems, and aircrew scheduling problems [16].

Research Purpose and Objectives

The purpose of this research is to improve modeling capabilities in the domain of military distribution. Currently, there are no models that effectively minimize the number of vehicles necessary in theater to sucessfully distribute the requirements of a TPFDD. The present method of trial and error is extremely time consuming and ineffective. The current techniques for finding optimal solutions generate very large problems usually that take a long time to solve. The solutions obtained using Hafich's model aimed to minimize total cost, but did not consider in detail the number of vehicles in place at each location. Due to the size of the problem, it is likely that alternative solutions exist with costs in the neighborhood of the optimal that are favorable due to a better distribution of vehicles by location or daily use.

The objective of this research is to examine metaheuristic approaches to solving the Theater Distribution Problem to provide a number of good solutions as opposed to a single optimal solution.

By providing USTRANSCOM with multiple feasible solutions of good quality, the analysts at force flow conferences may focus on picking one of several options instead of struggling through timeless simulation runs in an attempt to find a single option that works.

Organization

The remainder of this thesis is organized into four chapters. Chapter II discusses problems similar to the Theater Distribution Problem that have been solved as well as different heuristic approaches that may be applicable. Chapter III describes the heuristic approaches applied to the problem in this research. Chapter IV discusses test problem

generation and the computational results of the heuristics as compared to the optimal integer program. Chapter V provides concluding remarks and discusses possible future research in this area.

II. Literature Review

This chapter reviews the literature related to the theater distribution problem. It is organized as follows: First, some attention is given to problems in the literature that are related to the TDP, particularly the Pickup and Delivery Problem with Time Windows (PDPTW) and machine scheduling problems. Next, a discussion of heuristics and their relevance to this particular research is given. A number of basic heuristic approaches are discussed, and specific attention is given to several metaheuristics including multistart heuristics, Tabu search, simulated annealing, and genetic algorithms.

Pickup and Delivery Problem with Time Windows

A generalization of the vehicle routing problem with time windows, the PDPTW is concerned with constructing optimal routes and schedules to satisfy transportation requests at multiple locations. It is comparable to the Theater Distribution Model (TDM) primarily due to the time window and capacity constraints involved. An exact formulation for the problem was considered by Dumas et al, which used a column generation method to solve problems involving multiple depots and heterogeneous vehicle fleets to optimality. Their solution method uses a linear relaxation of the exact formulation followed by a branch-and-bound enumeration. Problems with 19 to 55 customers were solved using this approach [9].

Additional approaches to the PDPTW are found in the context of the Dial-A-Ride problem (DARP), a problem arising primarily in door-to-door transportation services for the elderly or disabled. The DARP is a generalization of the PDPTW. A detailed description can be found in [6].

An early survey of time window constrained routing problems, by [30] points towards approaches to the Multiple Vehicle Dial-A-Ride problem (MVDARP) suggested

by Roy et al and Jaw et al. Both approaches use parallel insertion, which construct routes for all vehicles simultaneously using distance as the main consideration. Jaw's approach was successfully implemented on a problem with 2600 customers and 20 homogeneous vehicles. More recently, metaheuristic approaches such as Tabu search [6], and large neighborhood searches [24] have been applied.

Due to the size of the problem considered in this research, exact approaches are not favorable. The number of requirements in a TPFDD is far greater than any PDPTW or DARP previously solved to optimality by any approach. For the TDP, it is important to model a large scale multiple vehicle heterogeneous fleet. The heuristic approaches developed for multiple vehicles described here have been applied to relatively small single-vehicle problems. Additionally, the PDPTW and DARP are both concerned with constructing routes for vehicles that start and end at central depots. The TDP requires solutions based solely on an origin to destination approach, so this amount of detail is unnecessary. The use of heuristics on instances of the PDPTW and DARP has been successful. Even though the specific algorithms applied previously are not directly applicable, it is likely that a heuristic approach to the TDP can produce favorable results

Machine Scheduling

The TDP is related to the machine scheduling problem; the goal is to assign requirements to vehicles in the same way jobs are assigned to machines. Objectives considered in the field of job scheduling include makespan, completion time, lateness, and tardiness, but little information is available on machine minimization. In their 2009 survey, Potts and Strusevich [26] discuss the computational complexity of numerous scheduling problems. Due to the level of computational complexity involved and the recent increase in heuristic research, much of the research in recent decades has focused on approximation schemes and heuristic procedures including local search, simulated annealing, Tabu search, and genetic algorithms [26].

Several publications exist in the field of scheduling that reference studies of different neighborhood definitions within metaheuristic searches. For the flow shop problem, the insert neighborhood, where a job is removed from its current position and inserted in a different position, has been shown to give better solutions than the swap neighborhood, which simply exchanges two jobs in a sequence. Both Osman and Potts [23] and Ogbu and Smith [22] show this in the context of a simulated annealing algorithm. An insert neighborhood is applicable to the TDP by taking requirements and shipping them on a different day or vehicle type.

Because most scheduling problems assume a number of machines prior to implementing approximation algorithms, it is difficult to apply the methods directly to the TDP. The increase in heuristic methods in recent years strengthens the belief that heuristics will be appropriate for this problem. However, an insert neighborhood can be applied to the TDP, and will be useful within a metaheuristic search. More detail on metaheuristics and neighborhood searches will be given in later sections of this chapter.

Heuristics

This section discusses basic heuristic search methods applied to general optimization problems. In general, an optimization problem is structured as some function of decision variables, possibly subject to a set of constraints. For illustrative purposes, the following formulation is presented, from [27]:

$$\begin{aligned} \text{Minimize} \quad & f(x) \\ \text{subject to} \quad & g_i(x) \geq b_i, \quad i = 1, ..., m; \\ & h_j(x) = c_j; \quad j = 1, ..., n. \end{aligned}$$

where x is a vector of decision variables, and $f(\cdot)$, $g_i(\cdot)$, and $h_j(\cdot)$ are general functions. Heuristic solution techniques are often applied to such optimization problems and are known for their simplicity, speed, and common sense notions.

There are several problem circumstances that make the use of heuristics both appropriate and advantageous. Those relevant to this particular problem include the use of inexact data to estimate model parameters and solving an inexact representation of the problem at hand [32]. Because the TDP is solved during resource and deliberate planning, it is possible that parameters and problem characteristics will change. If the model being used to represent the problem is simplified, it is already inaccurate, so the optimal solution may be irrelevant. In this case, it makes more sense to find a good solution quickly than to spend a great deal of time solving the model to optimality. The version of the TDP solved in this research is simplified on a number of different levels, so taking time to solve it optimally may be unimportant. Additionally, although exact methods do exist for solving the TDP, they are computationally unattractive. The integer programming approach to the TDP will find an optimal solution, but it is possible that it will take a very long time to do so. Lastly, heuristic approaches are generally simple and easy to understand. Understanding an approach will often instill confidence among leadership considering results from the model, resulting in a higher likelihood of the solutions being implemented. Additional reasons for using heuristic approaches as opposed to exact ones can be found in [32] and [29].

There are a large number of basic heuristic methods that are used to solve large problems quickly which also form the basis for more complex metaheuristic approaches. These include random solution generation, problem decomposition, constructive methods, and local improvement or neighborhood searches.

Random Generation.

Random solution generation is a very simple form of a heuristic. It operates by generating some number of feasible solutions to a problem, evaluating them, and choosing the best. The main benefit to random generation is that solutions can be generated very quickly. If large numbers of solutions can be generated, it is likely that one of them will be

good. Without considering specific problem characteristics, though, good performance is not expected [29].

Baum and Carlson examine probabilistic aspects of the use of random generation in decision problems through their 'better than most' approach. Their research allows for the calculation of confidence limits on obtaining highly ranked solutions for a given number of randomly generated feasible solutions. Although random generation is fast, in cases where the probability of a random solution being feasible is very low, Baum and Carlson argue that the 'better than most' approach can be computationally demanding [2].

Solutions to the TDP are in the form of very large and sparse matrices and checking feasibility requires looping over several variables. For this reason, the use of purely random solutions is not desirable, as most of the processing time is spent repairing and checking feasibility of the solutions.

Problem Decomposition.

In problem decomposition, a large complex problem decomposes into a number of smaller, simpler, subproblems. Complex optimization problems often consist of a number of decisions that need to be made. If these decisions can be decomposed, whether by their sequential nature, their chronological points in time, or their bearing on different resources, the problem as a whole is often much easier to solve. The method of decomposition depends a great deal upon specific problem characteristics. After the subproblems are defined, they can be solved independently, sequentially, or iteratively [29].

Clapp applied this technique to a multimodal PDPTW by splitting the problem into two components. The first component builds a simplified network using Dijkstra's algorithm, which solves for distances to be used as input for calculating costs in the scheduling portion of the heuristic. The second component solves the scheduling problem

by first assigning all requirements to a specific mode of travel and then assigning vehicles in a least cost fashion [4].

Constructive Approaches.

Constructive heuristics use specific problem characteristics to build solutions iteratively. Each iteration of a constructive heuristic adds one element to a partial solution until a full solution is found, typically concluding the procedure. Some constructive heuristics are referred to as greedy because at each step in the procedure they pick the solution element that most improves the immediate solution. This approach is quick to implement and easy to understand, but can lead to poor solutions due to initial choices voiding out possibly better choices later on [29].

The Clarke and Wright savings heuristic and the sweep algorithm, examined by Cordeau et al. [5], are good examples of greedy approaches to the vehicle routing problem. [28] and [31] are very early greedy algorithms for the popular multidimensional knapsack problem that use an effective gradient as a measure of value for each project. The first approach begins with an infeasible solution and systematically drops items until it becomes feasible, while the latter begins with an empty solution and adds items one at a time. Generally, greedy heuristics are designed with problem specific knowledge. For this reason, these particular approaches cannot be directly applied to the TDP, but the concepts behind them may assist in the development of an initial constructive solution.

Another constructive approach that is an alternative to greedy algorithms is the semi-greedy heuristic, which builds a solution constructively, but considers a number of solution elements at each iteration as opposed to only the best. Given a percentage p or cardinality c, the semi-greedy approach described by [15] considers at each iteration either the top c or the number of solutions within $p\%$ of the best, and chooses one at random to add to the solution.

Because constructive approaches are easy to implement and find solutions quickly, they serve as a good starting point for heuristic approaches to the TDP. An initial solution found by a greedy approach may not be the best quality, but it can be used as input to more advanced techniques.

Local Improvement.

Local improvement methods, unlike the previously mentioned techniques, require a feasible solution to the problem as a starting point. From this starting point, a neighborhood of solutions is evaluated, and the current solution is replaced by one of the solutions in the neighborhood with a better objective value. This process continues until no improvement can be found over the current solution. More formally, given a solution space S, cost function f, and neighborhood structure N, an algorithm for local improvement, from [8] can be found in Algorithm 1.

Select a starting solution $s_0 \in S$;
repeat
  Select s such that $f(s) < F(s_0)$ by a suitable method;
  Replace s_0 by s;
until $f(s) > f(s_0)$ *for all* $s \in N(s_0)$;
s is the approximation to the optimal solution.

Algorithm 1: Local Search [8]

Neighborhood definition is very important for this method. In short, the neighborhood of a solution is defined as the set of all solutions that can be obtained by performing a simple transformation on the current solution. Several neighborhoods can be obtained from the same solution. Examples of simple transformations include swapping 0-1 bits in binary problems, exchanging the order of jobs in a sequencing problem, or interchanging two or more edges in a routing problem. To save time on large problems, some local searches move to the first improving solution as opposed to evaluating all solutions and choosing the best. Local improvement methods can guarantee local optimal

points, but they do not guarantee globally optimal solutions. As a result of this, local searches perform best when combined with diversifying measures such as randomly generated restarts or as part of a metaheuristic optimization routine [29].

Metaheuristics

Basic heuristics are useful in finding good quality initial solutions. Certain heuristics, like random generation, are very good at finding diverse solutions from a large area of the search space. Other methods, like local improvement, are better at intensifying the search in a particular area. In order to build a strong, capable heuristic, both of these qualities should be present. One way of ensuring a search that is diverse but also has the ability to intensify is to add an outer process to guide it. This technique is commonly referred to as the use of a metaheuristic. Metaheuristics are very popular and have been studied a great deal in reference to problems with various applications. This section will discuss multistart constructive approaches, Tabu search, simulated annealing, and genetic algorithms.

Multistart Constructive Approaches.

The solutions obtained through the use of constructive heuristics, as described in the previous section, are highly dependent on the starting point of the procedure. This is especially true for greedy approaches, where typically only one solution is obtained from each starting point. An effective method for generating a large number of solutions very quickly is through the use of a multistart constructive approach. This approach runs a constructive heuristic multiple times, each with a different starting point. After a set number of solutions are generated, the best solution can be chosen and returned, or local searches can be performed on the most favorable solutions [29].

An adaptation to the simple multistart constructive approach is the greedy randomized adaptive search procedure (GRASP), originally developed for the partition problem, which deals with separating clusters of objects in order to minimize interaction.

Similar to the semi-greedy approach described in the previous section, GRASP works constructively by considering one solution element at a time. At each iteration, a candidate list of solution elements is constructed based on solution quality, and one element from the list is chosen at random to be the next piece of the solution. After the solution is entirely constructed, a local improvement phase is triggered. This process is performed for a set number of iterations, and the best solution is reported. Readers interested in GRASP can reference [19].

Because multistart constructive approaches run quickly and require very little additional structure than a simple constructive approach, they are very applicable to the TDP. A multistart constructive approach can be easily implemented on the TDP and can identify several good solutions. Because of this, the multistart constructive approach is the first type of metaheuristic applied in this research.

Tabu Search.

Since it's development in the mid 80s, Tabu Search has been applied to a large number of problems, including vehicle routing, network design, scheduling problems, and several military applications [13] [16]. Originally conceived by Fred Glover as a part of a graduate school project, Tabu search introduced memory structures in order to expand the search space and prevent local optimality traps. Provided that a starting solution is given, a properly defined neighborhood exists, and there exists some function that evaluates the quality of the solutions in the neighborhood, Tabu search can be used as an effective guiding tool.

Unlike some of the previously discussed methods, Tabu search is not constructive, and must begin with a complete solution as input. This solution can be obtained by a simple constructive, greedy, or random approach. The procedure then develops additional complete solutions by examining the neighborhoods of the current solution. At each step in the search, a move to a solution in the neighborhood is performed, causing the current

solution to change. In order to escape from local optima, Tabu search permits moves to inferior points. Additionally, to avoid the possibility of cycling back to recent solutions, a memory of past solutions or solution characteristics is kept on a tabu-list. These solutions are forbidden for the 'length' of the list, or a set number of iterations. Some instances of Tabu search permit moves to tabu solutions with desirable qualities through the use of an aspiration criteria, such as an objective value better than the best found so far. The procedure is terminated after either a set number of iterations is completed, or if no improvement is achieved after some number of consecutive iterations. Two flow charts from [13] describing the procedure are available in Figures 2 and 3. Readers interested in Tabu search should reference [13] and [11] for basic understanding and practical applications of the method, and [12] for additional refinements, adaptations, and applications.

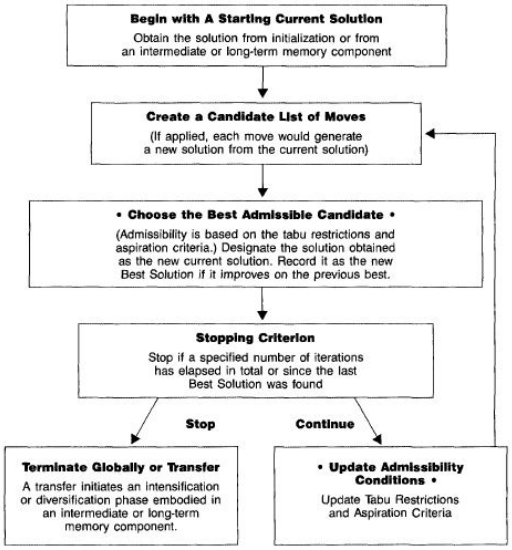


Figure 2: Tabu Search[13]

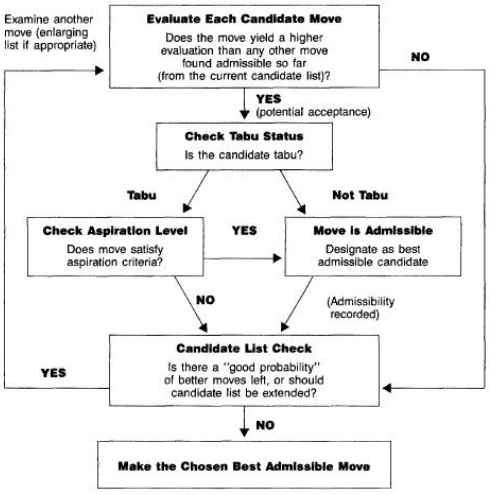


Figure 3: Tabu Status Check [13]

Batitti and Tecchiolli developed the reactive tabu (R-TABU) in 1994, which introduced long term memory and an adjustable length tabu list. In addition to the basic

search memory of solution value, this approach also keeps track of each visited solution's location, the iteration it was last visited at, and the number of times it has come up in the search. The method also introduces parameters which allow for increasing and decreasing the length of the tabu list and an escape mechanism that assists in increasing diversity in the search. For a detailed explanation of R-TABU, see [1].

In "A Guide to Vehicle Routing Heuristics," Cordeau et al give several examples of Tabu search strategies applied to the vehicle routing problem. Their results give comparative statistics on 5 different Tabu search based metaheuristics, including an adaptive memory approach, which achieved a high level of accuracy. For more information on the different search strategies used and see [5].

In 2004, Crino, Barnes, and Nanry developed a group theoretic Tabu search to solve the theater distribution vehicle routing and scheduling problem. The particular problem determined vehicle routes and schedules in order to achieve time definite delivery (TDD) for a list of demands. Their detailed methodology resulted in near optimal routing and scheduling of vehicles, and was applied to a number of problems of varying size, each with up to 90 vehicles and 30 requirements [7]. This research was followed by another application in 2010 by Burks et al. which introduced pickup and delivery and location requirements. Their implementation of Tabu search solved problems with 80 heterogeneous vehicles and up to 200 demands [3].

Although the research done by Crino et al.[7] and Burks et al. [3] considered more detail than necessary for the research conducted here, their use of Tabu search shows favorable results to a similar problem. The instances of Tabu search implemented by Cordeau et al also show favorable results for a vehicle routing problem [5].

With the proper framework in place, it is likely that an implementation of Tabu search, tailored to the TDP, will identify high quality solutions. This framework would include a heuristic that builds initial solutions, a neighborhood search to examine

additional solutions, and a Tabu outer function that tracks previous solution attributes and values. This research does not explicitly examine the aplication of Tabu search due to the excessive framework required.

Simulated Annealing.

The optimization routine referred to as simulated annealing was introduced in the early 1980s. The name of the procedure is derived from the physical process of annealing, where materials are heated to a liquid state and systematically cooled back down to a solid state, resulting in a stronger material. The algorithm mimics this process through the use of a temperature parameter, which is controlled throughout the search. Designed primarily to assist in escaping from local optima, simulated annealing, like Tabu search, begins with a complete solution and generates additional solutions in an iterative fashion. At the start of the search, the temperature parameter is high, introducing a random element to the search, and can allow for non-improving moves. As the search continues, the temperature parameter is iteratively decreased, which increases intensification and forces convergence to a local optimum. An outline of the process, from [8] can be found in Algorithm 2.

Readers interested in detailed methodology of simulated annealing should reference [10] which discusses some tips for choosing parameters in a simulated annealing search, some performance improving modifications to the algorithm, and points towards computational results in the literature.

Simulated annealing is a quick running heuristic that requires little memory and is good at finding diverse solutions through controlled randomization. In general, only the best solution found so far is maintained in memory, but the method can be adapted to include several of the best found solutions. By increasing the memory in the procedure, it is likely that an implementation of simulated annealing will produce favorable results to the TDP in a short amount of time. Similar to Tabu searches, though, Simulated

Select a starting solution s_0;
Select an initial temperature $t_0 > 0$;
Select a temperature reduction function α;
Repeat
  Repeat
    Randomly select $s \in N(s_0)$;
    $\delta = f(s) - f(s_0)$;
    if $\delta < 0$ **then**
      $s_0 = s$
    else
      generate random x uniformly in the range $(0, 1)$;
      if $x < e^{\frac{-\delta}{t}}$ **then**
        $s_0 = s$
      end
    end
  Until $iterationcount = nrep$;
  Set $t = \alpha(t)$;
Until stopping condition = true;
s is the approximation to the optimal solution.;

Algorithm 2: Simulated Annealing

Annealing requires an initial framework of an initial solution generator and a neighborhood search before it can be considered.

Genetic Algorithms.

Genetic Algorithms were developed by relating the concepts of natural selection to optimization and the search for good solutions. The idea of strong characteristics surviving through several generations and combining to build strong organisms was translated to optimization by Holland in the 1960s and 70s. The most basic genetic algorithm begins with a population of solutions in the form of encoded representations of the decision variables. These solutions are evaluated to determine their fitness levels, and a certain number of the most fit are given more opportunity take part in recombination. Similar to the way reproduction takes place in nature, two "parent" solutions are combined to create offspring in each iteration of a genetic algorithm. The simplest method for combining parents is through a one-point crossover operator, which chooses a point in

one parent solution and swaps everything following that point with the elements of the other parent solution. An example of a one-point crossover, with crossover point X, parent solutions P1 and P2, and offspring O1 and O2, is given in figure 4. In addition to crossover, most genetic algorithms utilize a mutation operator, which chooses a random element of a solution and changes it. This introduces a degree of randomization, which helps keep the population from converging prematurely to a local optima [27].

Figure 4: Example of One-Point Crossover in a Genetic Algorithm [27]

The use of crossover and mutations operators can cause many of the offspring solutions to be infeasible. As a result of this, feasibility checks and repair operators are required. Because the TDP has very limiting constraints and checking feasibility is difficult, this method is not particularly useful in the context of this research.

Conclusion

Although a great deal of research has been done in the area of routing and scheduling vehicles with time constraints, most of the approaches involve a level of detail that is unnecessary for this particular problem. In reference to the PDPTW, most of the attention in the research is given to the routing aspect of the vehicles, which is not considered in the TDP. Additionally, the approaches assume a central depot and often a set fleet size. The goal of the TDP is to find the least cost set of vehicles that satisfies the origin to destination requirements of the TPFDD, so a fleet size is not required and a central depot is not considered.

Although little research has been done on minimizing the number of machines in scheduling problems, the types of approaches for other problems in the field will likely be useful in this case. The research done on the effectiveness of neighborhood searches gives a starting point for a metaheuristic approach to the TDP.

The literature available for heuristic research is vast. The specific algorithms covered in this chapter are only a few of the more popular approaches. In terms of this particular application, a multistart constructive approach is used to identify good quality solutions, and simulated annealing and Tabu search are likely to identify high quality solutions after a reasonable neighborhood search is constructed. Genetic algorithms are not expected to perform well due to the large size of a TPFDD and the likelihood of generating infeasible solutions through crossover and mutation operations.

This research focuses primarily on finding good quality constructive algorithms that can identify feasible solutions to the TDP. It lays a framework for further applications of metaheuristic approaches such as Simulated Annealing or Tabu search, but does not explicitly develop or implement them.

III. Methodology

This research examines of a number of heuristics designed to build solutions to the TDP. Two shipping methods are considered. The first shipping method takes requirements and attempts to ship them as early as possible within their time windows. The second uses the entire available window, attempting to ship with fewer vehicles over a longer period of time. Both of these shipping methods are used in conjunction with two different kinds of constructive heuristics that consider vehicle types in different ways. The first considers vehicles strictly in order of minimum cost, while the second considers an input of desired ratios and probabilistically chooses vehicles for each shipment. In total, four basic constructive algorithms are built. Finally, metaheuristic approaches, in the form of multistarts, were applied to all algorithms in an attempt to locate a number of good solutions, and in the case of the probabilistic multistart, to converge on a favorable ratio of vehicles. Six multistart heuristics are considered in this research.

This chapter is organized by first examining both shipping functions. An explanation of the constructive heuristics that consider modes in order of minimum cost will follow. Probabilistic approaches are explained next. Finally, the chapter concludes with a discussion of the multistart heuristics that work in conjunction with all four algorithms.

All functions described here require a TPFDD document and supporting data to build constraints. Table 1 contains some notional TPFDD data in the form that was used by the programs discussed in this research. Column 1 contains the requirement number or ID, which is used for referencing the data in the associated rows. In some cases, these rows are shuffled, but the requirement number still aligns with the remainder of its row. The second and third columns refer to the POD and the final destination of the requirement. The column labeled ‘tons’ lists the weight in shorttons of the corresponding requirement. The earliest available date (EAD) refers to the day that the requirement

arrives at it's POD. It is not available for shipping until the following day. RDD refers to the requirement's due date. Finally, maximum allowable lateness (MAL) refers to the number of days a requirement is permitted to be delivered past its RDD.

Req	POD	Dest	Tons	EAD	RDD	MAL
1	1	1	500	2	7	1
2	1	1	250	3	8	1
3	1	1	750	4	9	1
4	1	1	200	5	10	1
5	1	1	100	6	11	1
6	1	2	600	2	8	1
7	1	2	400	3	9	1
8	1	2	200	4	10	1
9	1	2	300	5	11	1
10	1	2	500	6	12	1
11	2	1	500	4	8	1
12	2	1	400	5	9	1
13	2	1	300	6	10	1
14	2	2	1000	3	8	1
15	2	2	200	5	10	1
16	2	2	500	7	12	1

Table 1: Notional TPFDD

A number of assumptions were made in the context of these algorithms. First, all tons are assumed to be 'liquid'. In other words, it is assumed that the requirements can be broken up into as many shipments as desired. In the real world, there are shipments, such as tanks or other large equipment, that cannot be broken down. Next, if a vehicle is chosen for a particular day, it is assumed to complete all possible cycles. This saves time and calculation efforts, but may result in a calculation of shipping more than the required tonnage. To alleviate this, the functions attempt to fill this space with other available requirements. Lastly, all outloading and unloading calculations are made for the same day. This is true for vehicles whose cycles are at least 0.5 per day, but it does not apply to vehicles with longer cycle times.

Shipping Functions

The shipping algorithms discussed here are not built to work as standalone functions. Instead, they work within the constructive and probabilistic routines discussed in later sections of this chapter.

The first method of shipping works with all functions in this research. The method is called ExtraRoom and it checks if there are vehicles under capacity already traveling the necessary route for the specified requirement and time window. Pseudo-code for the function is included in Algorithm 3 for reference.

Data: Given *start* and *last*
for *day* = *start:last* **do**
  if extraroom(i, j, day) > 0 and reqleft(n) > 0 **then**
    fill extraroom with req(n);
  end
end

Algorithm 3: Extra Room

The next method of shipping, Ship Early, considers the concept of shipping goods as early as they are available. Prior to calling the function, the number of vehicles needed to ship a requirement with the current vehicle type is calculated. If it is possible to load and unload this number of vehicles at the corresponding POD and destination on the first available day, the algorithm assigns the shipment and concludes. If it is not possible to ship the entire requirement on the first available day, the maximum feasible number of vehicles is assigned instead. The remainder of the requirement is then used to recalculate the number of vehicles required. Next, an attempt is made to ship the requirement entirely on the next day in the available window. This same process continues until there is nothing left to be shipped for the current requirement, or every day in the time window has been considered. Pseudo-code is included in Algorithm 4 for reference.

```
Data: Given current vehicle m, start, and finish
day = start;
while reqleft > 0 and day < finish do
    if outloading + numveh < maxoutloading and
    unloading + numveh < maxunloading then
        ship remaining requirement;
        update unloading, offloading, cost, and reqleft;
    else
        ship as much as possible on current day;
        update unloading, offloading, cost, and reqleft;
        recalculate numveh required to ship current requirement;
    end
    day = day + 1;
end
if reqleft <0 then
    extraroom = - reqleft;
end
```

Algorithm 4: Ship Early

The third shipping function, Ship Over, considers the entire available time window for a particular requirement. Similar to Ship Early, Ship Over runs within a constructive or probabilistic routine, where the current vehicle and requirement combination is determined. Given this particular vehicle type, the ship function checks to see if there is room for one more vehicle on any of the days in the available window. If this is the case, the function iterates through each day, adding one of the current vehicle type to each day where it is feasible. This process continues until the entire requirement has been shipped, or there is no longer room for more vehicles of the current type within the available time window. Pseudo-code for the function is available for reference in Algorithm 5.

```
Data: Given current vehicle m, start, and finish
while reqleft > 0 and m can ship between (start, finish) do
    for day = start to last do
        if outloading + 1 < maxoutloading and
        unloading + 1 < maxunloading then
            add one vehicle to current day;
            update unloading, offloading, cost, and reqleft;
        end
    end
    if reqleft < 0 then
        extraroom = - reqleft;
    end
end
```

Algorithm 5: Ship Over

Constructive Functions

The constructive functions designed in this research, referred to as Construct 1 and Construct 2, are greedy and thus do not necessarily provide optimal solutions. They are designed primarily to provide a solution in a short amount of time. Both greedy constructive approaches require inputs including the TPFDD, average payload, possible cycles between origin and destination combinations, outloading capacities for each origin, and unloading capacities for each destination. Given these inputs, the functions construct solutions to the TDP by utilizing one of the previously discussed shipping methods. Each time the function is called, the solution matrix is empty. After each iteration of the function, pieces of the solution are added until all requirements are exhausted, leaving the shipping schedule in matrix form. This matrix can be used to display a final solution. The cost, or objective value, of the solution is updated throughout the program. Each time a shipment is assigned, the cost is updated accordingly. At the conclusion of the procedure, the total cost is returned.

The nature of the constructive algorithm ensures that solutions are feasible in terms of outloading and unloading, but not by lateness. Some solutions require deliveries past

their maximum allowable lateness. When portions of requirements are delivered past their required delivery date (RDD), a penalty function, which considers amount of tonnage late and time past due, is applied. Poor solutions are often a reflection of an inability to deliver all requirements on time.

Both Construct 1 and Construct 2 follow the procedure described by Algorithm 6, which describes the shipments made on time, and Algorithm 7, which describes how late shipments are scheduled. The only difference between the two functions is that Construct 1 ships with Ship Early, and Construct 2 ships with Ship Over.

The algorithm orders all vehicles by minimum cost after reading in all associated data including the TPFDD, available vehicle types, and constraint values. It also calculates the number of vehicles required to deliver each requirement by every vehicle type. Next, starting with the first requirement listed in the TPFDD and continuing in order of appearance, the program looks up the POD, *i*, and destination, *j*, for the current requirement, *n*. Once this information is obtained, the program checks to see if there is any extra room available on vehicles already traveling from *i* to *j* on any day between the current requirement's EAD and RDD. If extra room is available, it is utilized, and the remaining requirement is updated to reflect the amount shipped with previous requirements.

```
VehOrder = Order vehicles from least to most expensive;
for n = 1 to numreq do
    ExtraRoom(EAD(n),RDD(n));
    for v = 1 to numveh do
        m = VehOrder(v);
        if reqleft >0 and shipment by m is possible then
            Ship(EAD(n), RDD(n));
        end
    end
end
```

Algorithm 6: Construct (On Time)

The next step in the algorithm is to assign a vehicle to the current requirement. By first ordering all vehicles in the order of minimum to maximum cost, the program attempts to ship entirely by the most inexpensive vehicle. A more expensive type is used only if it is no longer feasible to ship by the cheapest vehicle. Examples of infeasibility include a location/destination pair not being able to unload and outload a particular vehicle mode, or unloading and outloading constraints for a particular mode being full. If a requirement cannot feasibly travel exclusively by one vehicle type, as much as possible is shipped by the least expensive vehicle available, and the remainder of the shipment is considered for the next vehicle type.

After all requirements are considered for delivery within their EAD and RDD, the function loops through all requirements once more to check if any shipments were not fulfilled. In the event that some requirement remains, the function checks again for any extra room that may have opened up due to vehicles being scheduled after the requirement was first considered for shipment. If this does not clear up the requirement in full, the requirement is scheduled on or after it's MAL using the same method as in the on time shipments.

Like most greedy algorithms, the solutions obtained depend upon their starting point. In this particular instance, the 'starting point' is the order of requirements considered for shipment. It is possible to reproduce a particular solution simply by using the same ordering of requirements.

Because most requirements in a TPFDD require many vehicles, the first constructive function will usually build solutions that fill up unloading and outloading constraints right away. The greedy nature of this algorithm can result in later requirements being unable to ship within their time windows, resulting in a number of late deliveries. The second constructive function attempts to alleviate the greediness of the algorithm by shipping over the entire available window.

```
for n = 1 to numreq do
    if reqleft >0 then
        ExtraRoom(EAD(n),RDD(n));
    end
    if reqleft > 0 then
        start = RDD;
        finish = RDD+MAL;
        while reqleft >0 do
            ExtraRoom(start,finish);
            for v = 1 to numveh do
                m = VehOrder(v) if  shipment by m is possible then
                    Ship(start, finish);
                end
            end
            start = finish;
            finish = finish +1;
        end
    end
end
```

Algorithm 7: Construct (Late)

Greedy constructive algorithms are helpful in that they provide solutions very quickly. However, both constructive functions provide solutions based on starting position. When given the same TPFDD, a constructive function of this type will always return the same solution. To find varied solutions with each run of an algorithm, randomization is introduced.

Probabilistic Functions

The probabilistic functions implemented in this research rely on randomizing the choice of vehicles in order to ensure that many different solutions are generated for the same TPFDD.

The probabilistic methods implemented were modeled primarily after the constructive functions. A probabilistic element was introduced to ensure different solutions resulted from each run of the program. To do this, the functions require

additional inputs of ratios which list the desired probability for choosing each vehicle type. In order for the programs to function properly, these ratios must add up to one. Because not all vehicle types are always available at every location, the first step in each function is to reevaluate these given ratios based on locations. If a vehicle type is not available at a particular location, the probability assigned to that vehicle is split evenly between all other available vehicle types. This results in a number of location specific ratios, indexed by i and j.

```
for req = 1 to numreq do
    ExtraRoom(EAD(n),RDD(n));
    while reqleft > 0 and some vehicle type is feasible do
        k = pick a vehicle type at random from feasible vehicles;
        if k can travel from i to j between EAD and RDD then
            Ship(EAD, RDD);
        else
            k is infeasible;
        end
    end
end
```

Algorithm 8: Probabilistic (On Time)

The probabilistic functions work by making random draws to choose vehicle types for shipment while keeping track of which vehicles are feasible for the current requirement being considered. Each time a new requirement is considered for shipment, the current ratio is set equal to the previously calculated ratios by location corresponding to the current requirement's POD and destination. By placing these values in a separate variable, it is possible to make temporary changes to the ratios. The function randomly chooses a vehicle type for the current requirement and attempts to ship by this vehicle. If it is no longer possible to add any vehicles of the current type within the shipping window, the ratio for the current vehicle is set to zero and the remaining ratios are adjusted

accordingly. This process continues until there is nothing left of the current requirement or until all vehicles are infeasible.

Pseudo-code for the on time delivery portion of the probabilistic methods is available within Algorithm 8. The only difference between the first and second probabilistic functions is that Probabilistic 1 ships by Ship Early, and Probabilistic 2 ships by Ship Over.

Late deliveries for the probabilistic functions are considered after attempting to schedule on time deliveries for all requirements in a similar manner as in Algorithm 7. In the event a requirement cannot be delivered on time, it is considered for travel by extra room, followed by delivery within it's MAL, and finally, delivery past MAL. Pseudo-code for the late portion of the probabilistic algorithm is found in Algorithm 9.

```
for n = 1 to numreq do
    if reqleft >0 then
        ExtraRoom(EAD(n),RDD(n));
    end
    if reqleft > 0 then
        start = RDD;
        finish = RDD+MAL;
        while reqleft > 0 and some vehicle type is feasible do
            ExtraRoom(start,finish);
            k = pick a vehicle type at random from feasible vehicles;
            if k can travel from i to j between start and finish then
                Ship(start, finish);
            else
                k is infeasible;
            end
            start = finish;
            finish = finish +1;
        end
    end
end
```

Algorithm 9: Probabilistic (Late)

Multistart Heuristics

Both the constructive and probabilistic algorithms find solutions to a given TPFDD. Unfortunately, there is no guarantee that these solutions are favorable or even feasible. In order to find a number of good solutions to a given TPFDD, it is necessary to run these constructive algorithms a number of times from different starting points through the use of a metaheuristic. This research considers three particular metaheuristics, in the form of multistarts, that work in conjunction with the constructive and probabilistic functions.

The first multistart heuristic works with the constructive functions. It requires as input a specified number of iterations to complete as well as a specification of which function, Construct 1 or Construct 2, to build the solutions. Because these approaches depend on the order of the requirements in the TPFDD, the multistart heuristic works by randomizing the order of the requirements before attempting to build a solution. Each iteration of the function uses a different permutation of requirements. It returns an objective value and the permutation of requirements that created it. The cost and permutations for each iteration, as well as the minimum cost found so far, are maintained in memory throughout the heuristic. By saving permutations instead of full solutions, it is possible to recreate solutions identified as favorable without using up too much memory. Due to the nature of the constructive algorithms, running the program on the same permutation of requirements will build an identical solution.

Unfortunately, because the heuristic is purely random, there is no guarantee that good solutions will be found. Increasing the number of iterations helps ensure a favorable probability of finding a good solution. Because the multistart heuristic runs so quickly, it is reasonable to increase the iteration count to find better solutions.

This approach is useful for finding multiple solutions, but it does not have the ability to converge on local optima. It is purely random. The probabilistic multistart, however, does take convergence into consideration, and will search the solution space while

converging on the best found ratio in the procedure. The function takes in a TPFDD, sorted by minimum delivery window, a set of initial ratios, a number of iterations, and a specification of which function, Probabilistic 1 or Probabilistic 2, to build solutions with. Because this approach does not involve shuffling the TPFDD before each iteration, ordering the TPFDD in order of minimum delivery window allows for difficult or bottleneck shipments to be scheduled first. This reduces the occurance of shipments being forced late due to unecessary early shipment of other easier to schedule requirements. Given all necessary inputs, MultistartProb will initialize storage vectors to maintain memory, by iteration, of ratio, objective value, best found objective value, and best found ratio. The ratio for each iteration, or current ratio (CR), is calculated with the following formula,

$$CR = IR + \frac{\eta}{iter} * (BR - IR)$$

where IR is the initial ratio given by the user, η is the current iteration number, $iter$ is the total number of user specified iterations, and BR is the best ratio found so far. This calculation ensures that the ratios used to construct solutions converge to the best found ratio in the procedure. If a new best is found within the program, the ratios will continue converging toward this new best. Figure 5 shows a graph depicting the best found solution and the current solution value for one run of the program.

The randomness of the algorithm is apparent in the large fluctuations in the current iteration's objective value. However, as the iteration count gets closer to the specified limit, in this case 500, the spread of solutions becomes smaller. This is due to the program converging on a best ratio identified in the search.

The last metaheuristic applied is a combination of the first two. It works the same way as the probabilistic multistart, but it shuffles the TPFDD before each iteration. This increases the amount of randomness in the method and allows for a broader search.

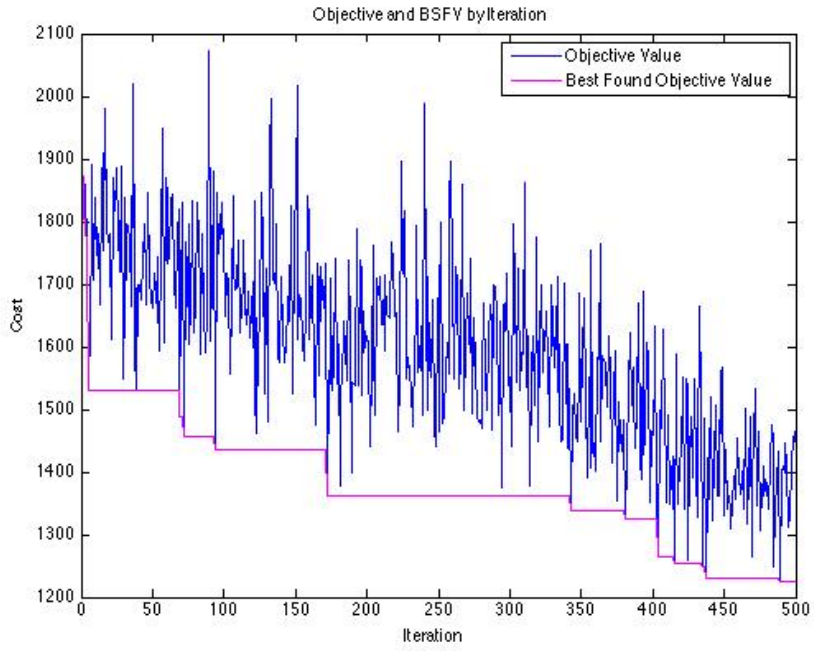


Figure 5: Objective Versus Best Found So Far

Conclusion

This chapter discusses the inner workings of the algorithms, both constructive and probabilistic, developed in this research. It also introduced three metaheuristic approaches. Chapter IV will discuss testing and analysis of these functions.

IV. Testing

The main focus of this research is to efficiently provide reasonable solutions to the TDP. To test the algorithms constructed in this research, the multistart heuristics discussed in Chapter 3 are run on various test problems. The best found objective values and the time at which they were found is recorded for all problems. All six variations described in Chapter III are tested, and the results are compared to the optimal solutions acheived by Hafich's integer program.

This chapter begins by discussing the development of the notional TPFDD data used for testing. A description of the test plan, some specific considerations, and problem parameters are also discussed. Finally, results from the tests are given and conclusions are drawn.

Data

The TPFDDs used in this analysis were constructed by a Visual Basic program designed by Percival [25]. Using parameters that attempt to mimic real world data, the program randomly generates notional TPFDDs of various sizes. Each TPFDD is generated as a .csv file, and includes 20, 100, 1000, 5000, or 10,000 requirements. Due to lack of time and the amount of memory required for calculating solutions, this research only analyzes problems up to size 1000. The random TPFDD data also includes each requirement's corresponding POD and destination in alpha-numeric format, weight in short tons, EAD, RDD, and MAL. To work directly with the TPFDDs, the POD and destination data points were converted to numeric indices. This allowed importing the TPFDDs into Matlab, where the bulk of the testing and analysis took place. Because real world TPFDDs are classified, notional data are created for the purpose of testing. The

development of random TPFDD data is a new and ongoing project; different parameters for their generation are being examined.

There are several features in the data that can effect the performance of the algorithms. Some problem characteristics from the first set of test problems, built using the original TPFDD generator, are available in Table 2. First, the average weight of requirements in the TPFDD was noted for each problem. The span of days that the problem considered, calculated by taking the maximum RDD and subtracting the minimum EAD for each data set, was also considered. Finally, the average delivery window, or gap, was calculated by taking the difference between each requirement's RDD and EAD and averaging over all requirements.

Table 2: Dataset A TPFDD Statistics

Metric	**20**	**100**	**1000**
Weight	257.95	285.21	268.19
Range	14.74	30	76.4
Gap	4.032	4.066	4.047

Table 2 shows that for the most part, the average requirement weight and delivery windows stay more or less constant as problem size increases. The range of the problem is the only attribute that clearly increases with problem size.

The second set of test problems was designed in an attempt to alleviate the tightness of the delivery windows and create something that was more realistic in terms of actual TPFDD data. To accomplish this, data from 5000 requirement problems were compared to data from a declassified condensed version of a real world TPFDD containing 4,426 requirements. The comparison is available in Table 3.

Table 3 shows several differences between the problems being randomly generated and the real world TPFDD. Particularly, the difference in the Gaps for the problems is

Table 3: Comparison of Dataset A to Real World TPFDD Data

Metric	5000	Real TPFDD
Weight	300.46	197.17
Range	187.75	295
Gap	4.070	16.93

very large. To better mimic real world data, Dataset B was built with an increased delivery window. The histogram in Figure 6 shows the distribution of the gaps in the declassified TPFDD.

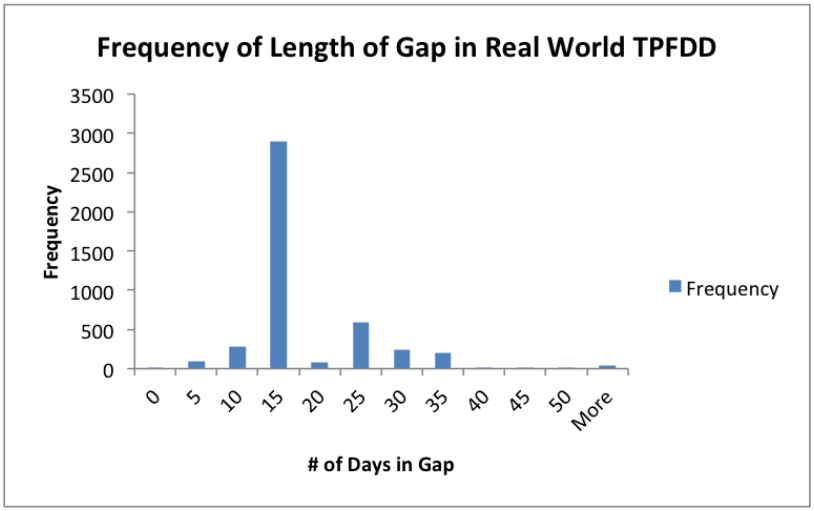


Figure 6: Frequency of Gap Length for Real World Data

In Dataset A, a random number drawn from a uniform distribution was used to generate due dates. Clearly, the data from the real world TPFDD do not follow a uniform distribution. For this reason, the gaps in Dataset B are created with normally distributed random variables. This ensures that the majority of requirements have a delivery window close to the average of 15 days, but still allows some requirements to have shorter or longer windows. Although the data from the declassified TPFDD do not necessarily follow a normal distribution, it is a better approximation than a uniform distribution.

Problem characteristics from Dataset B, the testing set for this research, are recorded in Table 4.

Table 4: Dataset B TPFDD Statistics

Metric	**20**	**100**	**1000**
Weight	243.41	357.68	371.25
Range	30.95	49.75	94.15
Gap	15.97	15.27	15.45

Testing Considerations

Because the integer programming model is designed to run for at least 20 seconds, it is desirable to run the heuristics for an equivalent amount of time. However, the probabilistic functions use the total iteration number in calculations at each iteration, so it was not possible to simply run the functions for a set period of time. Instead, the first step in setting up the tests was to determine a comparable number of iterations to run on the heuristics.

Running the heuristics for the same amount of time as the integer program allows for comparison of performance between each heuristic and the optimal solution. However, because the performance of heuristics is a function of the number of iterations completed, it is difficult to determine which of the heuristics is the most efficient. For this reason, all hueristics had to perform the same number of iterations. Additionally, in order to ensure that various solutions were due to differences in the heuristics and not to random variation, a random number stream was maintained for all problems.

To find a reasonable iteration count for each problem size, the number of iterations completed by each heuristic in 20 seconds of computer time was recorded. This data was used to determine a number of iterations that enabled the majority of the heuristics to run for 20 seconds. These numbers are provided with the problem results.

All solutions identified as the best are based on cycle cost considerations and do not explicitly account for the beddown required by the solution. The heuristics attempt to reduce the number of vehicles in the solution by filling empty space and reusing vehicle types for each requirement; the cost of obtaining vehicles is not considered in the objective function. Additionally, the best found solutions are more than likely one of several solutions found with similar costs throughout the course of the search. Figure 7 shows an example of this graphically.

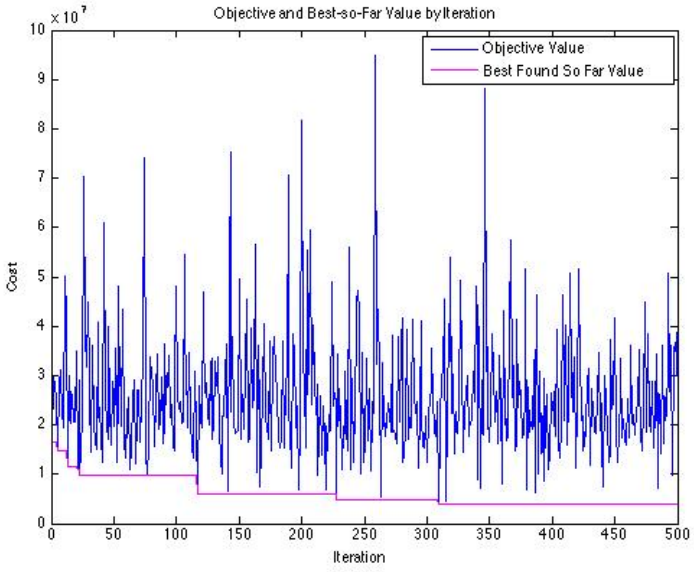


Figure 7: Multistart Random Approach Solution Value and Best-so-Far

The graph in Figure 7 depicts a case where the best found solution from a constructive approach is possibly one of several good quality solutions. This figure, generated by a Multistart of Construct1 on a 20 requirement problem, depicts both the value of the best found objective so far as well as the solution value at each iteration. The presence of several points that give the same or only slightly higher costs than the best solution found is obvious. These near best solutions may be worth examining, as they may

provide more favorable qualities, such as a better distribution of vehicles or a more favorable timeline. As the number of requirements increase, the possibility of finding several solutions that give similar costs is also expected to increase. TPFDD documents are usually made up of a large number of requirements, so alternate optimal solutions are common.

Parameters

A number of parameters must be set to build a solution for the TDP. These include MAL for each requirement, lateness penalty, and available vehicle modes and types. Additionally, each vehicle's maximum payload and cycle cost is required, as well as the number of cycles possible for each POD, destination, and mode combination. To ensure consistency between the optimal integer program and the heuristics, identical parameters are used in both methods. The cycles between each POD and destination were held constant for all vehicles and origin/destination combinations at a level of 1 cycle per day. MAL was also maintained at one day for each requirement. The penalty for each ton late was set at $10,000 per day late. Vehicle modes, types, and their corresponding costs and payloads used in the analysis are available in Table 5.

Table 5: Vehicle Payloads and Costs

Mode	Type	Payload	Cost
Air	C130	12	3
	C17	35	9
	C5	60	16
Road	HEMTT	7	1
	M1083	5	1
	M35	8	1
Rail	DODX	200	60
	FTX	150	42
	ITTX	180	52

Due to variation between TPFDDs of equivalent sizes, it was difficult to choose reasonable unloading and outloading constraints that worked for all TPFDDs. A level for unloading and outloading must be low enough to ensure problem feasibility, but not so high that single vehicle solutions are obtained. To acheive this, levels were varied for each test problem. The levels of unloading and outloading were based on the average weight of the requirements in the TPFDD divided by the payloads for the first vehicle type in each mode. For the 20 and 100 requirement problems, an additional divisor of 2 was used to increase difficulty and reduce the probability of getting single vehicle solutions. For the more difficult 1000 requirement problems, a divisor of 1.1 was used to ensure feasibility. Outloading and unloading constraints were identical to and from all locations.

All probabilistic functions require an additional input of the desired ratio of vehicle types. For simplicity, all problems were started with equivalent vehicle likelihood. This was accomplished by setting each vehicle ratio to 1/9.

The remainder of this chapter will use the function abbreviations in Table 6. Associated attributes are shown in Table 7

Table 6: Function Abbreviations

Function	**Abbreviation**
Construct 1	C1
Construct 2	C2
Probabilistic 1	P1
Probabilistic 2	P2
Probabilistic 3	P3
Probabilistic 4	P4

Table 7: Function Attributes

Function	C1	C2	P1	P2	P3	P4
Shuffled	x	x			x	x
Probabilistic			x	x	x	x
Ship Early	x		x		x	
Ship Over		x		x		x

Results

The first step in the analysis is to build solutions to the TPFDDs with 20 requirements using the heuristics described in Chapter III. Because these problems are relatively small, each iteration of the heuristics runs in a very short amount of time. This allowed the programs to complete 500 iterations in the 20 second time frame. Table 6 lists the function name abbreviations used in the remainder of this section, and Table 8 shows the results from the 20 requirement problems.

All heuristics found feasible solutions without lateness to all of the 20 requirement problems in under 20 seconds. In several cases, both Probabilistic 2 and Probabilistic 4 found the optimal solution. In addition to the objective values, the number of iterations completed and the time it took to reach the best solution were recorded for each problem. The best heuristic solution found for each problem is in bold text in the table. Based on Table 8, Probabilistic 2 and 4 appear to outperform the other approaches. The summary statistics in Table 9 better compare the results. Table 9 provides the average time until the best solution is found, the average optimality gap, and the percentage of time the approach found the best heuristic solution.

The values in Table 9 indicate that the solutions found by the heuristics were within 17% of the optimal for the constructive functions, within 5 to 7% for Probabilisic 1 and 3, and within 2% for Probabilistic 2 and 4. Again, P2 and P4 appear the best.

Table 8: 20 Requirement Solution Values in 500 Iterations

Problem	Optimal	C1	C2	P1	P2	P3	P4
1	610	732	736	694	**632**	656	**632**
2	738	875	858	844	**738**	744	**738**
3	632	722	722	641	**632**	660	**632**
4	604	731	700	654	**604**	663	**604**
5	475	543	543	492	**476**	487	**476**
6	772	903	902	827	812	805	**774**
7	927	1098	1121	1047	1000	1007	**978**
8	565	660	660	595	**567**	614	**567**
9	597	681	681	623	**597**	610	**597**
10	607	735	736	640	616	619	**607**
11	535	629	615	573	**535**	544	**535**
12	512	624	623	585	**537**	620	**537**
13	686	782	784	772	**707**	725	755
14	640	777	777	708	**657**	709	**657**
15	568	658	653	574	**569**	587	**569**
16	604	691	689	624	**604**	621	**604**
17	602	690	689	653	611	617	**603**
18	599	702	704	605	640	668	**613**
19	585	678	668	592	**586**	590	**586**

Table 9: Comparison Metrics for 20 Requirement TPFDDs

Metric	C1	C2	P1	P2	P3	P4
Time to Best	3.92	8.36	3.65	12.00	3.37	11.25
% Best Found	0	0	0	73.7	0	94.7
% From Optimal	17.3	16.8	7.2	2.0	5.8	1.6

Table 9 indicates a distinct difference between the two shipping functions, Ship Early, and Ship Over. In the methods where requirements were shipped over their entire available window, the best solutions were not found until close to 10 seconds on average. When requirements were shipped as early as they were available, good solutions were

found in an average of under 5 seconds. Additionally, the solutions found using Ship Over were better than those found using Ship Early.

Probabilistic 2 and 4 are clearly performing better than all other heuristics implemented. The greediness of the constructive functions result in quick feasible solutions, but they do not outperform the probabilistic functions in any cases. Although Probabilistic 2 and 4 take longer to find solutions than Probabilitic 1 and 3, they found the optimal solution in several cases, and still perform faster than the integer program.

The next step in the analysis examines the solutions obtained by the heuristics for 100 requirement TPFDDs. A similar approach was taken to compare the heuristics to the optimal solutions as was used for the 20 requirement TPFDDs. Due to the size of the problems, each iteration of the functions took a longer amount of time. While the 20 requirement TPFDDs allowed 500 iterations in 20 seconds, the 100 requirement TPFDDs allowed only 150. Results from the 100 requirement tests are found in Table 10

The heuristics were able to generate on-time solutions to the majority of the 100 requirement problems. In the case of problem 9, each the heuristics actually found better solutions than the integer program. This is possible because the integer program is designed to return a solution when it comes within 50% of the lower bound, which in the case of problem 9 was found to be 4127. Therefore, the solution returned by the integer program is not always the actual optimal solution.

Interestingly, the trend observed from the 20 requirement problems did not necessarily continue on to the 100 requirement problems. In these cases, as evidenced by the bolded lowest found solutions in the table, Constructive 2 provided the best solutions the majority of the time, instead of Probabilistic 2 or 4. However, for problems 15 and 16, both constructive functions were unable to find on-time solutions. As long as an on time solution can be found by the constructive solutions, it is generally good. Inability to find

Table 10: 100 Requirement Solution Values in 150 Iterations

Prob	Opt	C1	C2	P1	P2	P3	P4
1	5608	5879	**5683**	6089	5839	6157	6118
2	4335	5199	**5054**	5458	5574	5519	5575
3	4819	5776	**5686**	6461	6342	6039	6631
4	4535	5484	**5411**	5825	5831	5672	5923
5	5267	5990	5867	6079	6022	6481	**5822**
6	4895	5622	**5512**	6016	6403	6271	5775
7	5755	6496	**6449**	6716	6994	6821	6867
8	4478	5495	**5282**	5856	5540	5805	5452
9	5615	5033	**4932**	5419	5571	5274	5551
10	5079	5828	**5726**	6320	6163	6248	6093
11	4984	5996	**5794**	6237	6175	6449	6052
12	4956	5568	**5513**	6040	6194	6052	6136
13	4901	5553	**5432**	5901	5831	5876	5966
14	5481	6148	**5990**	6662	6276	6797	6320
15	4902	895937	2695830	6293	6628	6479	**6053**
16	4594	555716	1125651	**5888**	6148	6202	6104
17	5228	6498	**6270**	7034	6427	6960	6774
18	4535	5506	**5447**	5775	5792	5650	5691
19	4911	6111	**5968**	6630	6286	6605	6285
20	4948	6034	**5864**	5926	5882	6035	6031

on-time solutions due to the greediness of the algorithms will only increase as problem size increases.

Similar metrics were taken from these results as in the 20 requirement problems, however, the presence of lateness in some solutions makes it difficult to compare some of the metrics. To properly analyze the differences in the performance of the heuristics on the 100 requirement problems, additional metrics were used. First, a percentage was calculated to reflect the proportion of time a heuristic found an on time solution. Next, the average of the percent difference from optimal using these on time solutions was taken. This allows for consideration of the likelihood of finding an on time solution and the quality of solution we can expect, given an on time solution, for each of the heuristics

studied. The average number of tons late, given a late solution, was also recorded for each heuristic. These metrics are compiled in Table 11.

Table 11: Comparison Metrics for 100 Requirement TPFDDs

Metric	**C1**	**C2**	**P1**	**P2**	**P3**	**P4**
Time to Best	7.11	18.15	9.10	28.73	10.29	29.09
% From Optimal	15.8	13.2	23.3	22.6	24.0	21.9
% On Time	90	90	100	100	100	100
% Best Found	0	85	5	0	0	10
Tons Late	72.58	191.07	N/A	N/A	N/A	N/A

We note that there is degraded performance by all algorithms from the 20 requirement problems to the 100 requirement problems. This is most likely due to the inability to perform as many iterations. Instead of coming within 5 or 10% of the optimal solution, the heuristics are within 15 to 25% of the optimal. The time until the best solution is found also increased from under 5 or 15 seconds to closer to 10 or 30. The difference between the two shipping methods, however, remains consistent. Although the methods shipping over the entire available window take longer, they tend to find better solutions.

To further examine the performance of the algorithms on the 100 requirement problems, the functions were allowed to perform 500 iterations. The results from these runs are summarized in Table 12

Table 12: Comparison Metrics for 100 Requirement TPFDDs in 500 iterations

Metric	**C1**	**C2**	**P1**	**P2**	**P3**	**P4**
Time to Best	19.38	51.00	28.07	92.16	27.80	91.34
% From Optimal	5.1	4.3	13.1	20.4	12.4	18.5
% Best Found	0	100	0	0	0	0

Table 12 shows that increasing the number of iterations reduces the optimality gap. Particularly, the gaps for the Constructive and Probabilistic 1 and 3 functions reduce more than the gaps for the Probabilistic 2 and 4 functions. The previous late solutions from the constructive functions are resolved with the extended run time, resulting in Constructive 2 finding all the lowest heuristic solutions.

The next and final step in the analysis is to test the heuristics on 1000 requirement problems. Due to time constraints, only the first 1000 requirement problem was tested in the integer programming model. Using Hafich's Decision Support System, just over four minutes was spent generating the integer model. The integer program then took an additional 12 minutes to find a lower bound of 62,537 within the model solver. In twenty minutes, a feasible objective was found at 64,664.

A quick analysis on the functions processing the 1000 requirement problems showed 20 iterations or fewer being completed in 20 seconds. However, since the optimal integer program can take several minutes to generate and even longer to actually solve, the 1000 requirement problems were allowed to run for longer than 20 seconds.

The heuristics performed an average of 1 iteration per second when using the Ship Early function, and 1 iteration every 2 seconds when using the Ship Over function. Each heuristic was run for 60 iterations, or roughly one to two minutes per problem. Unfortunately, as predicted, the constructive heuristics could not generate on-time solutions to the majority of the 1000 requirement problems. The results are found in Table 13, with the best found solution value bolded for each problem.

Although not all problems were run through the integer programming model, we can note that all probabilistic functions found on-time solutions to the first problem in under 2 minutes. The solutions found in this case were within 40% of the optimal. Interestingly, Probabilistic 1 is responsible for finding the lowest solution in 65% of the

Table 13: 1000 Requirement Solution Values in 60 Iterations

Problem	C1	C2	P1	P2	P3	P4
1	1.35E+10	4.93E+09	90083	**82770**	89843	83592
2	1.08E+06	1.08E+06	**79446**	82589	80653	82273
3	2.71E+09	3.38E+08	81062	82227	**80682**	81728
4	1.34E+09	5.23E+07	80351	82480	**80062**	82054
5	1.71E+09	2.53E+06	**79268**	81248	79773	79704
6	1.27E+09	3.42E+06	**79876**	81723	80784	81774
7	7.52E+09	2.23E+09	**81486**	83314	82047	83196
8	5.39E+08	3.52E+06	**77185**	79127	78091	79322
9	1.90E+09	6.65E+07	**80930**	82915	81826	82463
10	1.90E+08	**70680**	79128	81217	79757	80452
11	3.67E+09	2.42E+08	**79225**	79400	79752	80451
12	8.23E+09	7.28E+09	**81539**	83199	83062	82967
13	4.33E+08	3.59E+05	**77880**	79149	78427	79604
14	70685	**69350**	78309	80682	78547	80822
15	5.78E+07	7.50E+05	**78142**	80064	78854	81009
16	1.37E+09	1.35E+08	**80668**	82713	81285	83050
17	2.59E+09	1.23E+08	77993	78703	**77371**	79731
18	3.67E+09	2.35E+08	**80121**	82792	81228	81656
19	4.14E+08	**70900**	80164	81804	80230	81708
20	3.89E+09	2.19E+09	**81617**	83223	82137	83755

problems. Also, every time Constructive 2 finds an on-time solution, it is the best solution found. The summary statistics are available in Table 14.

Table 14: Comparison Metrics for 1000 Requirement Problems

Metric	C1	C2	P1	P2	P3	P4
Time to Best	30.865	81.789	90.087	129.577	67.690	116.880
% Best Found	0	15	65	5	15	0
% On Time	5	15	100	100	100	100

To better compare the results of the heuristics to the results of the integer programming model, the heuristics were allowed to run for twenty minutes each on the first problem. The results from these extended tests are available in table 15. The

constructive approaches were unable to find on-time solutions in the extended runs, and therefore are not included in the table.

Table 15: 20 Minute runs on 1000 Requirement Problem

	P1	**P2**	**P3**	**P4**
Solution	83679	82699	84395	82801
Time to Best	497.66	1047.30	435.51	775.73
% From Optimal	29.4	27.9	30.5	28.0

As evidenced by the figures in Table 15, 20 minute runs on the 1000 requirement problems decreased the gap from the IP solution. The gap decreased for Probabilistic 1 and 3 more than it did for Probabilistic 2 and 4, showing that increased iterations on Ship Early approaches is more beneficial than on Ship Over approaches.

The results presented in this chapter show applicability of a heuristic solution to the TDP. Further conclusions from the research will be discussed in Chapter V.

V. Conclusions and Future Research

This chapter provides a summary of the research conducted in this thesis. A discussion of the conclusions follow and future topics of study are addressed.

Summary

This research constructed two distinct shipping methods used in conjunction with four heuristics and two distinct metaheuristic approaches for solving instances of the Theater Distribution Problem. The first shipping method examined an approach that shipped as much of each requirement as possible on the first day it was available, while the seccond method examined an approach that spread the shipment of each requirement over its entire available shipping window. A purely constructive approach was designed around each of the shipping methods. These constructive approaches were then adapted to include a probabilistic vehicle choice element and ulitimately became the probabilistic 1 and 2 approaches. Finally, multistart metaheuristics were designed so that a number of solutions could be found for each problem.

Following the development of the heuristic functions, test data was developed through the use of a random TPFDD generator. Changes were made to the generator to include longer and normally distributed delivery windows so that the data better mimicked real world scenarios. The heuristics were tested on problems with 20, 100, and 1000 requirements.

Conclusions

All of the heuristics developed in this research quickly provide good solutions to the 20 and 100 requirement problems. In several cases, the heuristics found equivalent or better solutions than those found by an integer programming model.

Unfortunately, on-time solutions to the majority of the 1000 requirement problems were not found by the constructive approaches. All probabilistic approaches were successful in finding solutions to these problems in just 1 to 2 minutes. These results are very good compared to the 20 to 30 minutes that it takes for the integer model to identify solutions to problems of this size.

None of the heuristics dominates the others. On smaller, 20 requirement problems, the Probabilistic 2 and 4 approaches outperformed the Constructive 1 and 2 and Probabilistic 1 and 3 approaches. On 100 requirement problems, though, Constructive 2 provided the majority of minimum heuristic solutions. Finally, Probabilistic 1 was responsible for the majority of the minimum solutions found on the 1000 requirement problems, again different from the best performers in smaller cases. It is likely that the increased size of the problems and inherent characteristics within them make problems of different sizes better candidates for certain types of searches. Additional research can focus on determining what methods work better on different problem types.

Although it is not particularly clear which of the heuristic models is the best approach to the TDP, the constructive approaches can be ruled out. The inability to find on-time solutions to the majority of the 1000 requirement problems does not fare well for their application to even larger, full scale TPFDDs. The probabilistic approaches had no issues generating solutions to these problems.

As problem size grows, the amount of time spent in the Probabilistic 2 and 4 functions increases more than the time spent in Probabilistic 1 and 3. This is due to the iterative shipping method used in Ship Over, which adds only one vehicle at a time to a requirement, as opposed to Ship Early, which adds as many vehicles as necessary or possible all at once. For larger problems, faster solutions can be found using the Ship Early approaches.

This research identifies the applicability of heuristic models to the TDP. In general, for small scale problems on the order of 20 to 100 requirements, the integer programming model is appropriate. However, real world TPFDDs are on the order of thousands of requirements. The time it takes to generate and develop solutions to a TPFDD of this size within the integer program is too long, and optimal solutions may not even be necessary.

The methods developed in this research are a reasonable first answer to the TDP. Additional methods should be examined and applied in order to ensure feasibility, quality of solution, and control over problem size. This work shows the applicability of a heuristic approach to the TDP, and provides a basis for further research.

Future Study

There are several possibilities for future study in regards to heuristic approaches to the TDP. First, simple additions to the heuristics studied in this research are possible, such as the use of a neighborhood search or variable reduction techniques. Next, a more advanced metaheuristic search approach such as a tabu search or simulated annealing could be applied. Finally, a study of TPFDD data parameters and their effect on solution quality would be of interest.

The heuristics applied in this thesis are good at developing solutions quickly, but there is no guarantee that the solutions obtained will be on-time. The addition of a feasibility check and repair operator could greatly benefit the solution quality obtained, and increase the amount of feasible and on-time solutions that are found. This could be accomplished through the use of a simple neighborhood search such as one that swaps requirements with different days or vehicle types. This could be applied not only for repairing solutions but also for improving good solutions.

Along the same lines of the neighborhood search, a more advanced metaheuristic approach that guides the neighborhood search could be applied. Tabu search is just one example of an elegant search procedure that could provide favorable results.

In addition to developing a neighborhood search, there is a definite need for some variable reduction techniques within the code. Similar to the number of variables generated in the original TDM, the coding applied in this thesis involved a lot of very large matrices that included nonsense combinations. This resulted in problems running much slower as the number of requirements increased. If more efficient data storage techniques are applied within the heuristics, the amount of memory used in the procedure would be greatly reduced. Efficient data storage eliminates the need to store values of zero or nonsense combinations, and allows more iterations to be completed in shorter time frames. With this, solutions for problems with more than 1000 requirements could be obtained with relative ease.

Although Mr. Percival created a program to build random TPFDDs, the effect of TPFDD characteristics on problem performance is largely unknown. It is hard to develop TPFDDs that vary on more than simply the number of requirements. Knowledge on the effect of varying parameters such as vehicle cost, cycles, and unloading or outloading constraints on the cost of shipping a particular TPFDD could be very helpful. By determining where the biggest bottlenecks are, it is easier for decision makers to determine where their efforts should be focused. Understanding the effect that certain parameters have could allow for a more streamlined heuristic solution approach.

Bibliography

[1] Batitti, Roberto and Giampietro Tecchiolli. "The Reactive Tabu Search". *ORSA Journal on Computing*, 6(2):128–140, 1994.

[2] Baum, Sanford and Robert Carlson. "On Solutions that are Better Than Most". *The International Journal of Management Science*, 7(3):249–255, 1979.

[3] Burks, Robert E., James T. Moore, J. W. Barnes, and John E. Bell. "Solving the Theater Distribution Problem with Tabu Search". *Military Operations Research*, 15(4):5–26, 2010.

[4] Clapp, Benjamin. *Vehicle Minimization for the Multimodal Pickup and Delivery Problem with Time Windows*. Master's thesis, Department of Operations Research, Air Force Institute of Technology, Wright Patterson AFB, OH, 2013 (AFIT-ENS-13-M-03).

[5] Cordeau, J-F, M. Gendreau, G. Laporte, J-Y Potvin, and F. Semet. "A Guide to Vehicle Routing Heuristics". *Journal of the Operational Research Society*, 53(5):512–522, 2002.

[6] Cordeau, Jean-Fracois and Gilbert Laporte. "A Tabu Search Heuristic for the Static Multi-Vehicle Dial-A-Ride Problem". *Transportation Research Part B*, 37(6):579–594, 2003.

[7] Crino, J. R., J. T. Moore, J. W. Barnes, and W. P. Nanry. "Solving the Theater Distribution Vehicle Routing and Scheduling Problem using Group Theoretic Tabu Search". *Mathematical and Computer Modeling*, 39(6):599–616, 2004.

[8] Dowsland, Kathryn. *Chapter 2: Simulated Annealing*. Modern Heuristic Techniques for Combinatorial Problems. Ed. Colin Reeves. McGraw-Hill International (UK) Limited, 1995.

[9] Dumas, Yvan, Jacqes Desrosiers, and Fracois Soumis. "The Pickup and Delivery Problem with Time Windows". *European Journal of Operational Research*, 54(1):7–22, 1991.

[10] Eglese, R. W. "Simulated Annealing: A tool for Operational Research". *European Journal of Operational Research*, 46:271–281, 1990.

[11] Glover, Fred. "Tabu Search–Part I". *ORSA Journal on Computing*, 1(3):190–206, 1989.

[12] Glover, Fred. "Tabu Search–Part II". *ORSA Jounral on Computing*, 2(1):4–32, 1990.

[13] Glover, Fred. "Tabu Search: A Tutorial". *Interfaces*, 20(4):74–94, 1990.

[14] Hafich, Micah. *A Mixed Integer Programming Model for Improving Theater Distribution Force Flow Analysis*. Master's thesis, Department of Operations Research, Air Force Institute of Technology (AU), Wright Patterson AFB, OH, March 2013 (AFIT-ENS-13-M-05).

[15] Hart, Pirie J. and Andrew W. Shogan. "Semi-Greedy Heuristics: An Empirical Study". *Operations Research Letters*, 6(3):107–114, 1987.

[16] Hill, Raymond R. and Edward A. Pohl. *An Overview of Meta-heuristics and their use in Military Modeling*, 9.1–9.25. Handbook of Military Industrial Engineering. Taylor & Francis Group, LLC, Boca Raton, FL, 2009.

[17] Joint Chiefs of Staff. *Joint Logistics. Joint Publication 4-0.* JCS, Washington, 2008.

[18] Joint Chiefs of Staff. *Distribution Operations. Joint Publication 4-09*. JCS, Washington, 2010.

[19] Laguna, Manuel, Thomas A. Feo, and Hal C. Elrod. "A Greedy Randomized Adaptive Search Procedure for the Two-Partition Problem". *Operations Research*, 42(4):677–687, 1994.

[20] Longhorn, Dave C. and Joshua M. Kovich. *Solving the Theater Distribution Problem Using Planning Factor and Integer Programming Approaches*. Scott AFB. 2012.

[21] McKinzie, K. and J. W. Barnes. "A Review of Strategic Mobility Models Supporting the Defense Transportation System". *Mathematical and Computer Modeling*, 39(6):839–868, 2004.

[22] Ogbu, FA and DK Smith. "The Application of the Simulated Annealing Algorithm to the Solution of the $n|m|C_{max}$ Flowshop Problem: A Computational Study". *Computational Operations Research*, 17:243–253, 1990.

[23] Osman, IH and CN Potts. "Simulated Annealing for Permutation Flow-Shop Scheduling". *Omega*, 17:551–557, 1989.

[24] Parragh, Sophie N. and Verena Schmid. "Hybrid Column Generation and Large Neighborhood Search for the Dial-A-Ride Problem". *Computers & Operations Research*, 40(1):490–497, 2013.

[25] Percival, Scott. "TPFDD Generator", 2014. Excel VBA code.

[26] Potts, CN and VA Strusevich. "Fifty Years of Scheduling: A Survey of Milestones". *Journal of the Operational Research Society*, 60(S1):S41–S68, 2009.

[27] Reeves, Colin and John Beasley. *Chapter 1: Introduction*, 1–19. Modern Heuristic Techniques for Combinatorial Problems. Ed. Colin Reeves. McGraw-Hill International (UK) Limited, 1995.

[28] Senju, Shizuo and Yoshiaki Toyoda. “An Approach to Linear Programming With 0-1 Variables”. *Management Science*, 15(4):B196–B207, 1968.

[29] Silver, E. A. “An Overview of Heuristic Solution Methods”. *Journal of the Operational Research Society*, 55(9):936–956, 2004.

[30] Solomon, Marius M. and Jacques Desrosiers. “Time Window Constrained Routing and Scheduling Problems”. *Transportation Science*, 22(1):1–13, 1988.

[31] Toyoda, Yoshiaki. “A Simplified Algorithm for Obtaining Approximate Solutions to Zero-One Programming Problems”. *Management Science*, 21(12):1417–1427, 1975.

[32] Zanakis, Stelios H. and James R. Evans. “Heuristic ”Optimization”: Why, When, and How to Use it”. *Interfaces*, 11(5):84–91, 1981.

A Heuristic Approach to the Theater Distribution Problem

2D Lt Emily K. Power Department of Operational Sciences (ENS), Air Force Institute of Technology

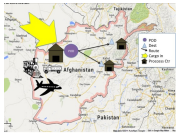
Purpose

To improve modeling capabilities in the military by providing analysts at USTRANSCOM with a method for providing multiple shipment solutions to a given problem in a reasonable amount of time.

Objectives

Develop metaheuristic approaches to the TDP and compare the heuristic solutions to those obtained by the ITDM

Background



There are several complicating constraints that need to be considered in the context of the Theater Distribution Problem.

- Time windows – EAD, RDD, and MAL
- Daily unloading and outloading limits that vary at locations throughout theater
- Multiple possible modes of travel
- Multiple vehicles for each mode
- Varying parameters for each vehicle type – cycles, payloads, and capacities

This problem was previously solved by trial and error. Recently, a functional integer program was able to build solutions to the problems, but it is still time consuming for large problems.

Contact Information

Advisor: Dr. Jeffery Weir

Reader: Dr. Raymond Hill

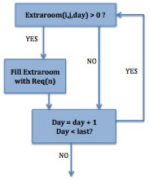
Sponsor: USTRANSCOM

Shipping Methods

- All shipping methods work inside either a constructive or probabilistic approach

Extra Room

Checks to see if previously scheduled vehicles have extra capacity. If so, the extra room will be filled with the current requirement.

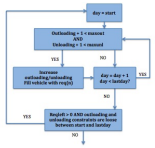


Ship Early

Ships a requirement as early in it's time window as possible by filling outloading and unloading constraints to their maximums.

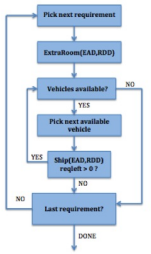
Ship Over

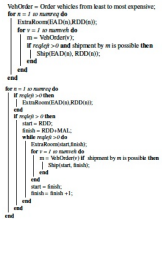
Designed in an attempt to reduce vehicle beddowns, this method ships over a requirement's entire available window with fewer vehicles per day.



Constructive Approach

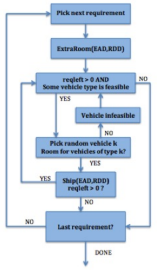
- Pure greedy approach
- Vehicles are ordered by minimum cost
- Provides a single solution

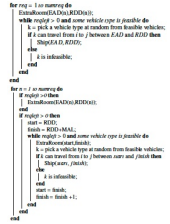




Probabilistic Approach

- Introduces a random vehicle choice element
- Reduces amount of greediness in algorithm
- Allows for multiple shipment solutions





Conclusions

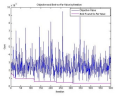
- "Best" approach is difficult to identify as different heuristics performed better on different problem sizes
- Demonstrated applicability of heuristics to the Theater Distribution Problem
- Provided a framework for future heuristic approaches to the problem

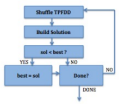
Future Research

- Simple additions to the heuristics used in this research such as neighborhood searches or feasibility checks and repair operators
- More advanced metaheuristc approaches such as Tabu or Simulated Annealing
- Smarter variable storage within code to allow for faster solutions
- Study of the effect of varying TPFDD parameters

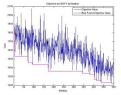
Metaheuristics

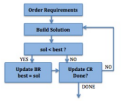
Random Multistart: Working with the Constructive function, this method shuffles requirements in a TPFDD to create multiple shipment solutions.





Probabilistic Multistart: Working with the Probabilistic function, this method alters the vehicle ratios at each iteration to provide multiple shipment solutions and converge on a favorable ratio.





REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704–0188

1. REPORT DATE *(DD–MM–YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From — To)*
27–03–2014	Master's Thesis	Oct 2012–Mar 2014

4. TITLE AND SUBTITLE

A Heuristic Approach to the Theater Distribution Problem

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Power, Emily K., Second Lieutenant, USAF

5d. PROJECT NUMBER

JON 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/EN)
2950 Hobson Way
WPAFB, OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT-ENS-14-M-25

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

United States Transportation Command
Joint Distribution Process Analysis Center
Attn: Pat Mcleod
508 Scott Drive DSN: 770-5238
Scott Air Force Base, IL 62225-5357 patrick.k.mcleod.civ@mail.mil

10. SPONSOR/MONITOR'S ACRONYM(S)

USTRANSCOM/TCAC

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

DISTRIBUTION STATEMENT A. APPROVED FOR PUBLIC RELEASE; DISTRIBUTION IS UNLIMITED

13. SUPPLEMENTARY NOTES

14. ABSTRACT

Analysts at USTRANSCOM are tasked with providing vehicle mixtures that will support the distribution of requirements as provided in the form of TPFDD. An integer programming model exists to search for optimal solutions to these problems, but it is fairly time consuming, and produces only one of potentially several good quality solutions.
This research constructs a number of heuristic approaches to solving the TDP. Two distinct shipping methods are examined and applied through both constructive and probabilistic vehicle assignment processes. Multistart metaheuristic approaches are designed and used in conjnction with the constructive and probabilistic approaches. Random TPFDDs of size 20, 100 and 1000 are tested, and solutions are compared to those obtained by the integer programming approach. The heuristic models implemented in this research develop feasible solutions to the notional TPFDDs in less time than the integer program. They can very quickly identify a number of good quality solutions to the same problem.

15. SUBJECT TERMS

greedy, heuristics, military distribution, theater distribution

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Dr. Jeffrey D. Weir (ENS)
U	U	U	UU	72	19b. TELEPHONE NUMBER *(include area code)* (937) 255-3636 x0000 Jeffrey.Weir@afit.edu