MULTIPLE VEHICLE ROUTING PROBLEM WITH FUEL CONSTRAINTS

A Thesis

by

DAVID R LEVY

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Chair of Committee,	Sivakumar Rathinam
Committee Members,	Sergiy Butenko
	Dvahg Swaroop
Head of Department,	Andreas Polycarpou

August 2013

Major Subject: Mechanical Engineering

ABSTRACT

In this paper, a Multiple Vehicle Routing Problem with Fuel Constraints (MVRPFC) is considered. This problem consists of a field of targets to be visited, and a collection of vehicles with fuel tanks that may visit the targets. Consideration of this problem is mainly in the improvement of feasible solutions, but the following steps are discussed: Cost Matrix Transformation, Field Partitioning, Tour Generation and Rerouting, and Tour Improvement.

Four neighborhoods were investigated (2-opt, 3-opt, Target Vehicle Exchange, Depot Exchange), using the Variable Neighborhood Descent and Variable Neighborhood Search schemes, with APD and Voronoi partition methods. These neighborhoods were compared to investigate their performance for various instances using the above schemes and partition methods. In general, 2-opt performed as well as 3-opt in less time than 3-opt; in fact, 3-opt was the slowest of the four neighborhoods. Additionally, the Variable Neighborhood Descent scheme was found to produce better results than the Variable Neighborhood Search.

NOMENCLATURE

APD	Approximate Primal-Dual algorithm
LKH	Lin-Kernighan Heuristic
MV	Multiple Vehicle
MVRPFC	Multiple Vehicle Routing Problem with Fuel Constraints
TSP	Traveling Salesman Problem
TVE	Target Vertex Exchange
VND	Variable Neighborhood Descent
VNS	Variable Neighborhood Search

TABLE OF CONTENTS

Page

LIST OF FIGURES

Page

LIST OF TABLES

Page

1 INTRODUCTION

The Traveling Salesman Problem (TSP) is a canonical problem in the field of optimization, with many practical applications in fields such as logistics, military surveillance, and disaster relief (Army, 2007), (Curry, Maslanik, Holland, & Pinto, 2004). This problem, in its most basic form, is to find the cheapest order to visit a collection of targets for a vehicle such that each target is visited at least once and the total distance traveled by the vehicle is a minimum. Finding an optimal solution for the TSP is challenging because the computation time required by the existing algorithms increase exponentially with the size of the problem. This difficulty is compounded when multiple vehicles are considered, and even more when fuel constraints are imposed on these vehicles. The TSP, with these additional considerations, is called the Multiple Vehicle Routing Problem with Fuel Constraints (MVRPFC). This problem can be stated as follows: given a set of targets, fuel stations, and vehicles, find a path for each vehicle such that every target is visited at least once, none of the vehicles violate their fuel constraints along their respective paths, and the total travel cost is a minimum.

The MVRPFC is a generalization of the standard TSP and is NP-hard. Therefore, the focus of this thesis is to develop heuristics that can find good solutions to the MVRPFC as quickly as possible. We do this through the framework of the Variable Neighborhood Search (VNS) and Variable Neighborhood Descent (VND). VNS and VND are meta-heuristics used to solve difficult combinatorial and global optimization problems. These are iterative algorithms where in each iteration, the algorithms search through multiple neighborhoods of the current feasible solution to find a feasible solution with lower cost. The use of multiple neighborhoods allows the solution in the VNS and VND heuristics to move away from local optima as quickly as possible. To generate an initial feasible solution to the problem, we rely on the approximation algorithm develop by Kaarthik et al. in (Sundar & Rathinam, 2013). An α-approximation algorithm is a polynomial time algorithm that produces a solution whose cost is at most α times the optimal cost for any instance of the problem.

The single vehicle version of the MVRPFC has been addressed by the authors in (Khuller, Malekian, & Mestre, 2011), (Sundar & Rathinam, 2013). Khuller et al. present an approximation algorithm for the symmetric version of the problem. Kaarthik et al. present an approximation algorithm for the asymmetric version of the problem. The MVRPFC is also closely related to routing problems with intermediate facilities [Ghiani, Angelelli, Crevier] as discussed in (Sundar & Rathinam, 2013). Variants of the MVRPFC have also been studied in the literature. Dell et al considered a multiple vehicle TSP from a practical perspective, incorporating time windows and equity constraints (Dell, Batta, & Karwan, 1996). Approximation algorithms and heuristics for a heterogeneous multiple vehicle TSP are studied by Rangarajan, where some targets must be visited by certain vehicles (Rangarajan, 2011). Oberlin discusses a transformation of a heterogeneous multiple vehicle, multiple depot TSP into an asymmetric TSP so that algorithms for the standard TSP can be put to good use (Oberlin, 2009). Rathinam and Sengupta determine lower bounds for a multiple depot, multiple vehicle TSP, along with a 2-approximation algorithm for solving this problem (Rathinam & Sengupta, 2006).

Hansen and Mladenovic review improvement schemes in (Hansen & Mladenovic, 2001), notably the Variable Neighborhood Descent (VND) and Variable Neighborhood Search (VNS), which are the focus of this paper.

In this paper, an α-approximation algorithm is used in combination with improvement heuristics to calculate solutions for an MVRPFC. The method discussed in this paper consists of the following steps: Cost Matrix Transformation, Partitioning, Tour Generation and Rerouting, and Tour Improvement. For the Partitioning step, an Approximate Primal-Dual algorithm is used and compared with a Voronoi implementation. For the Tour Improvement step, several heuristics are used: the 2-opt, 3-opt, Depot Exchange, and Target Vehicle Exchange heuristics. These heuristics are used as part of VNS and VND schemes, and are discussed in details, along with results from application of these schemes on several instances.

2 PROBLEM STATEMENT

Consider a multiple vehicle routing problem with K vehicles with fuel capacities $L_1, L_2, \ldots, L_k$, let T denote the set of targets to be visited, and let D denote the set of depots that are available. The problem is then formulated on a complete undirected graph $G = (T \cup D, E)$, where E is the set of edges between every pair of members of $T \cup D$, and is assumed to satisfy the triangle inequality: $cost(x, y) + cost(y, z) \geq cost(x, z)$. Additionally, some constraints are imposed on the problem data: for every target in T, it is required that there is a depot in D that is reachable by every vehicle:

$$\exists d \in D \; s.t. 2 * cost(t, d) \leq L_k \; \forall k, \forall t \in T.$$

Then the objective of this paper is to find K tours, 1 for each vehicle, such that every target in T is visited at least once, the total cost to traverse the tours is at a minimum, and none of the vehicles run out of fuel while traversing their tours.

3 ALGORITHMS

3.1 Overview

The algorithm discussed in this thesis consists of five basic steps: cost matrix transformation, partitioning, tour generation, tour rerouting, and tour improvement. The first step in the process is to adjust the given cost matrix to account for possible necessary refueling trips. The resulting cost matrix will then contain real fuel costs to travel between two nodes. The next step is the partitioning of the target field. In this step, groups of nodes are assigned to vehicles based on their proximity to the nodes. Once these partitions have been found, the next step is to find a tour for the partition that will only visit each target once. These tours are not necessarily feasible, so refueling trips are added, where necessary, to ensure that the vehicle can realistically navigate the tour. Once the tours have been made feasible, they are improved using a variety of improvement heuristics. The intent of these heuristics is to reduce the cost required for the vehicles to navigate the tours.

3.2 Cost Matrix Transformation

The first step in the algorithm is to adjust the cost matrix to account for refueling trips. This step is required to ensure that the partitioning algorithm has a monotonically non-increasing set of cost matrices to work with.. The first step is to determine the closest refueling station to every target that is to be surveilled. This information is used when adding required refueling trips. Once the closest stations have been found, the algorithm iterates through every pair of targets, determines if a refueling trip is necessary, and if so, calculates the cost of the refueling trip. To find the cheapest path for the refueling trip, a graph is formed that contains the nodes of interest and every refueling station on the field. Edges between nodes are then added to the graph if they are feasible. Once this graph has been formed, a simple shortest path algorithm is run. This shortest path is stored, and the cost to traverse it is placed in the adjusted cost matrix. A detailed technical description of the algorithm follows.

First, consider any two targets x and y, where the cost to travel from x to y is called D_{xy}. We denote the closest depot to x as d_x, where the cost to travel from x to d_x is called D_x. Similarly for y, we define d_y and D_y. Then our objective is to determine a path from $d_x \to x \to y \to d_y$ that is feasible; that is, the vehicle will not run out of fuel traversing this path. If L_k is the fuel capacity of the kth vehicle (the vehicle of interest), then to travel this path directly, we must have

$$D_x + D_{xy} + D_y \leq L_k \ \Rightarrow D_{xy} \leq L_k - D_x - D_y.$$

However, if this inequality is not satisfied, an indirect path must be found from x to y. To accomplish this, an intermediate graph G_L is formed. First, all the depots that are reachable from x after the vehicle has visited d_x are added to the graph; that is, all depots within a distance of $L_k - D_x$ from x are added to the graph, along with the edges from x to these depots. These depots are selected to ensure that they are reachable from x in the best case, when the vehicle has the most possible fuel it can have at x. Similarly for y, depots within a distance of $L_k - D_y$ from y are added to the graph. Then, the remaining depots on the field are added to the graph, and edges between the depots are added if the cost to travel those edges are less than L_k (see Figure 3-1).

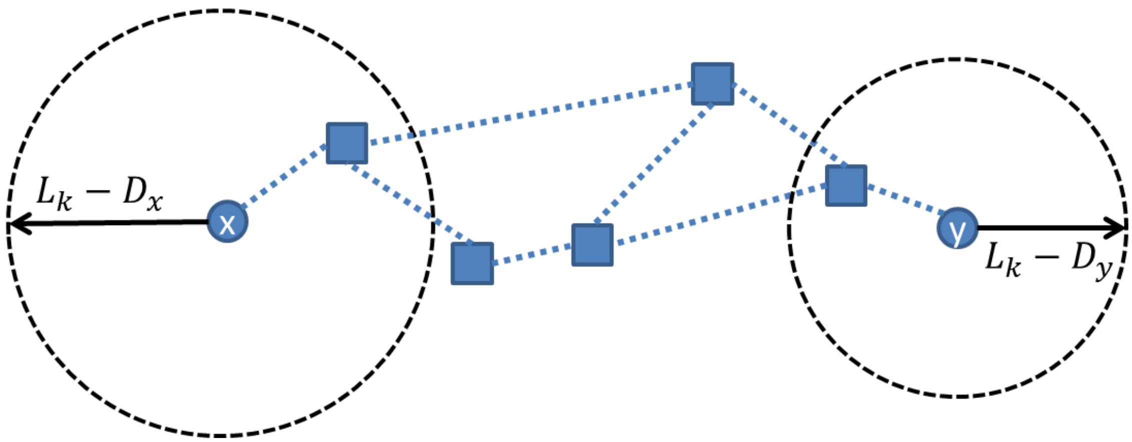


Figure 3-1 G_L Formation

Then, a shortest path is determined to travel from x to y along this graph, using Dijkstra's algorithm. This path, shown in Figure 3-2, is the indirect path from x to y, and the cost to travel this path is called L_{xy}. This L_{xy} value is calculated for every pair of targets and for every vehicle.

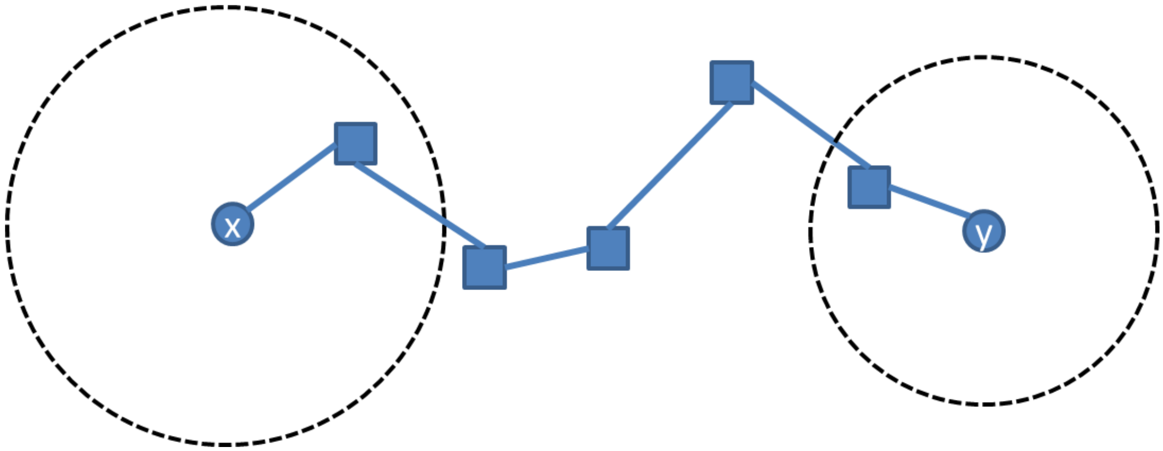
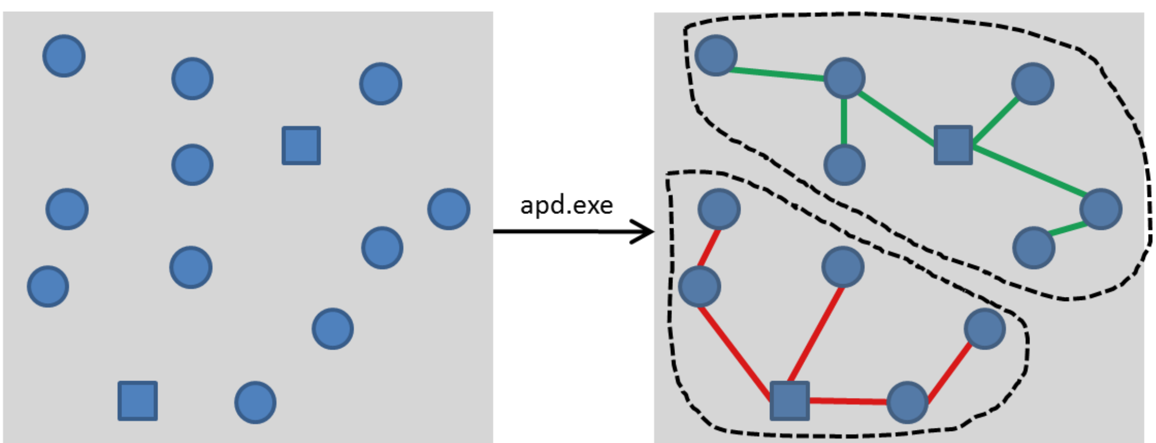


Figure 3-2 Shortest Path Determination

An important result of this algorithm, that is also a requirement for the Approximate Primal Dual algorithm, is that if the vehicles are ordered such that $L_1 \geq L_2 \geq \cdots \geq L_k$, then $L_{xy}^1 \leq L_{xy}^2 \leq \cdots \leq L_{xy}^k \; \forall x, y \in T$: the L_{xy} matrices are monotonically nondecreasing. This result is easy to understand when the graph formed between x and y is considered. For example, take the above algorithm for a pair of targets x and y and for the first vehicle. Then the graph between x and y contains only edges with a cost that is less than L_1, the fuel capacity of the vehicle. When the L_{xy} value is calculated for the second vehicle, this graph cannot contain more edges than that for the first vehicle, because $L_2 \leq L_1$. Therefore, L_{xy}^2, the cost of the shortest path from x to y in this graph, must be greater than or equal to L_{xy}^1.

3.3 Partitioning

The next step in the algorithm is to determine partitions for the vehicles. This consists of assigning groups of nodes to certain vehicles, based on some criteria. The partitioning algorithm used in this paper is an extension of the primal dual algorithm described by Jungyun Bae (Bae & Rathinam, 2011). The primal dual algorithm takes advantage of the relation between linear programs and their duals; it repeatedly tightens primal constraints via dual variables until there are no more constraints to tighten. At this point, a pruning step is performed to retrieve disjoint sets of nodes for each vehicle. The output of the algorithm is the field partitions for each vehicle, as shown in Figure 3-3.



The Approximate Primal Dual algorithm calculates spanning trees for each vehicle, which are directly translated into field partitions.

Figure 3-3 Primal Dual Algorithm Depiction

3.4 Tour Generation

The task of generating tours from partitions found in the previous step was delegated to K. Helsgaun's implementation of the Lin-Kernighan heuristic (Helsgaun, 2012). The lkh.exe executable takes the field partitions as input, and produces tours for each vehicle. The lkh executable finds low cost tours for the partitions that are input, without considering fuel capacity restrictions.

3.5 Tour Rerouting

The Lin-Kernighan heuristic implementation does not know about the fuel capacities of the vehicles, so the tours it returns are not guaranteed to be feasible. Therefore, refueling trips must be inserted into the tours where necessary. The first step in this process is to reintroduce the indirect refueling trips found in the cost matrix transformation step (Section 3.2). This step is called tour expansion, and is shown in Figure 3-4.

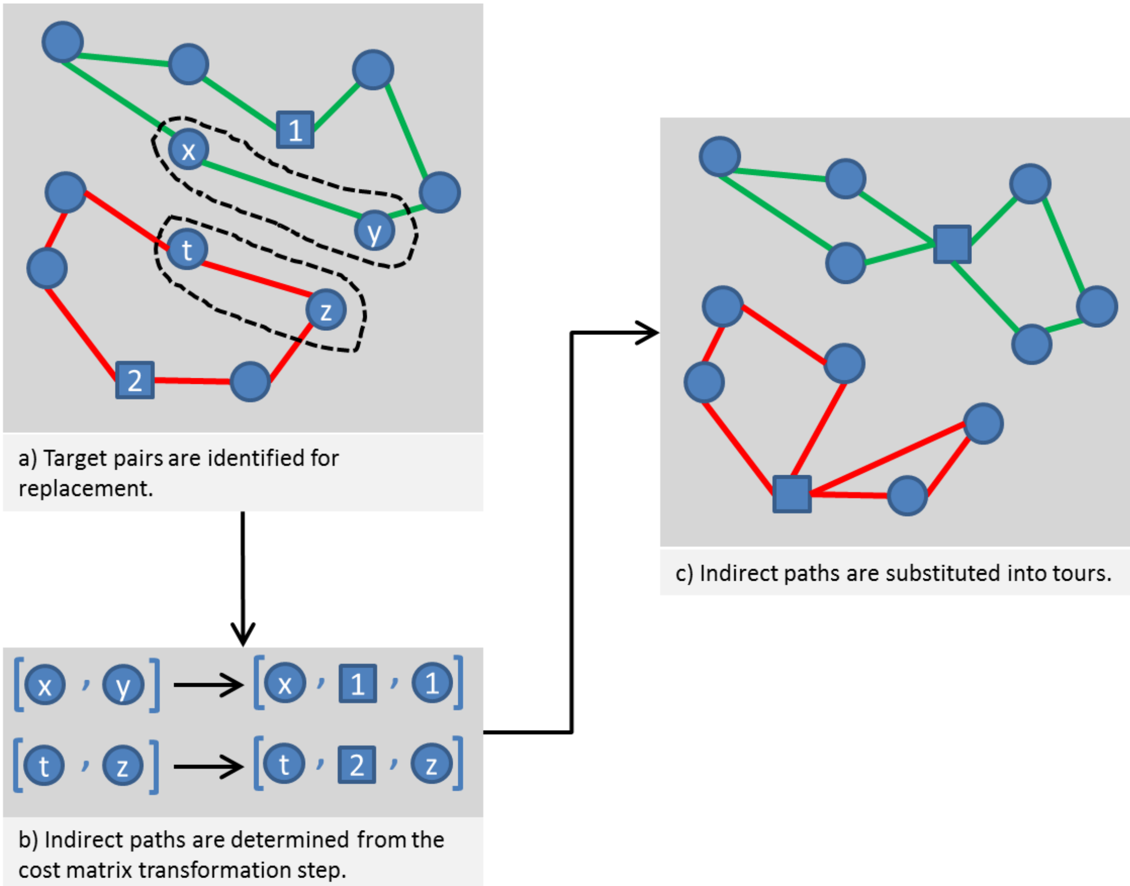


Figure 3-4 Tour Expansion

From this augmented tour, strands between refueling visits are identified and extracted. Each strand is checked for feasibility by calculating the cost required to travel the nodes in the strand. For each infeasible strand, an augmented greedy strand is generated where every node visit is succeeded by a refueling trip to the station nearest to the node. Refueling trips that are not required for strand feasibility are removed, and the strands are rejoined to form the feasible tour.

3.6 Tour Improvement

The tours generated at this point in the algorithm are far from optimal. In an effort to improve them, several schemes are used in conjunction with an implementation of a variable neighborhood search. The following neighborhoods are examined in this paper: 2-Opt, 3-Opt, Depot Exchange, and Target-Vehicle Exchange.

3.6.1 Variable Neighborhood Search

The variable neighborhood search is a method that is used to search for cheaper tours in multiple neighborhoods. A variable neighborhood search consists of 3 main steps: Shaking (covered in Section 3.6.1.1), Local Search (Section 3.6.1.2), and Move Or Not (Section 3.6.1.3) (Hansen & Mladenovic, 2001). To set up for a variable neighborhood search, a collection of k neighborhoods and an initial solution x are required. Once these have been determined, the first neighborhood N_1 is chosen as the "current" neighborhood. A concise description of the algorithm is shown in Figure 3-5. In this figure, step 3 is the Shaking step, step 4 is the Local Search step, and step 5 is the Move Or Not step.

1. Start with an initial set of tours $\mathcal{T}$.
2. Set $k = 1$.
3. Choose random neighbor $\mathcal{T}'$ from neighborhood N_k of $\mathcal{T}$.
4. Choose best neighbor $\mathcal{T}''$ from neighborhood N_k of $\mathcal{T}'$.
5. If $cost(\mathcal{T}'') < cost(\mathcal{T})$, set $\mathcal{T} = \mathcal{T}'', k = 1$, and go to Step 3. Otherwise, increment k.
6. If $k > length(N)$, output $\mathcal{T}$. Otherwise, go to Step 3.

Figure 3-5 Variable Neighborhood Search Algorithm

Figure 3-6 shows a depiction of the solution space, with the initial solution x denoted.

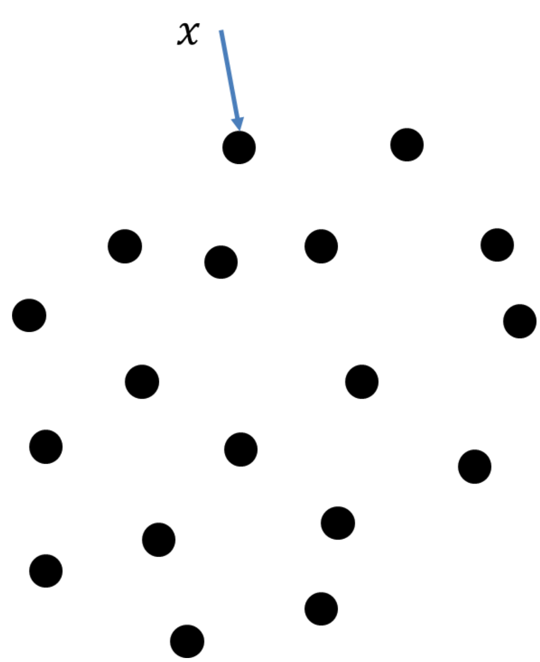


Figure 3-6 Variable Neighborhood Search Initialization

3.6.1.1 Shaking

The shaking step is the characteristic feature of the variable neighborhood search. In this step, a random member of the currently selected neighborhood of *x* is found, and denoted as *x'*. Selecting a random neighborhood member introduces fluctuations into the solution search path, and acts to generally avoid getting stuck in local optima. For certain neighborhoods, it may be possible that there are no feasible neighbors; i.e. the neighborhood of *x* is empty. In this special case, the shaking step is skipped, and *x'* is the same as *x*. Figure 3-7 shows the solution space, where the neighborhood N_1 is denoted by a red circle, and the product of the shaking step is denoted as x'.

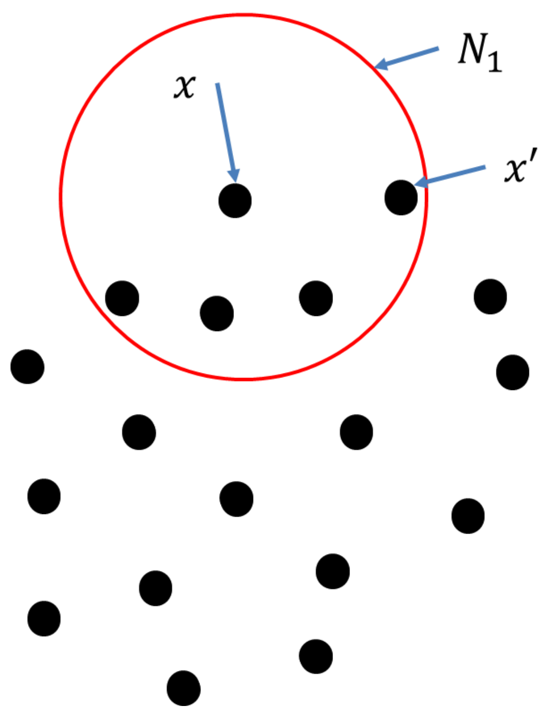


Figure 3-7 Variable Neighborhood Search Shaking Step

3.6.1.2 Local Search

Once *x'* has been found, a simple local search is used to find the cheapest solution in the current neighborhood of *x'*, which is denoted as *x''*. Again, it may be possible that the current neighborhood of *x'* is empty. If this is the case, *x''* is set to be the same as *x'*.

3.6.1.3 Move Or Not

In this step, the cost of *x''* is compared to the cost of *x*. The two possible outcomes of interest are when the cost of *x''* is less than the cost of *x*, and when the cost of *x''* is greater than or equal to the cost of *x*. In the first case, *x''* is cheaper than *x*. When this is true, *x* is set to be *x''* (see Figure 3-8). The first neighborhood is set as the "current" neighborhood, and computations continue with the Shaking step (Section 3.6.1.1). When the second case is true, *x''* is forgotten, and the next neighborhood is set as the "current" neighborhood. If there is no next neighborhood; i.e. the "current" neighborhood is the last neighborhood designated, the algorithm terminates.

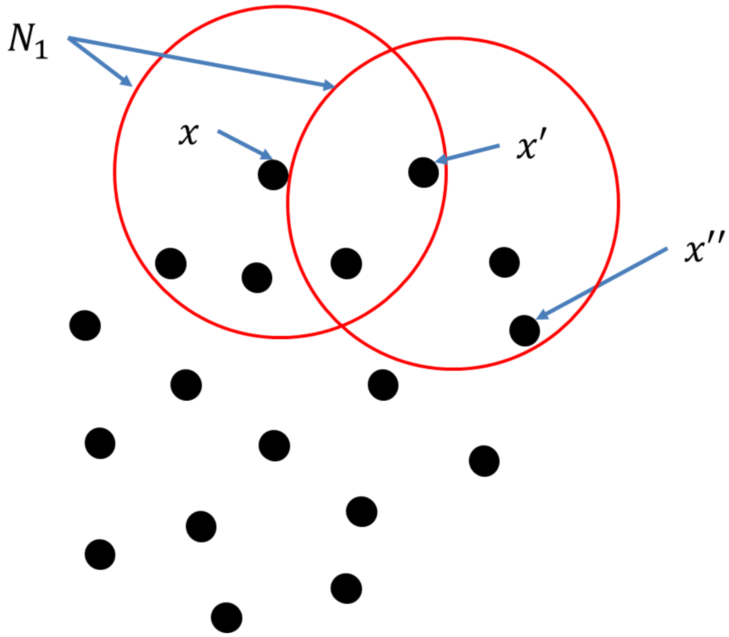


Figure 3-8 Variable Neighborhood Search Move-or-not Step

3.6.2 Variable Neighborhood Descent

The Variable Neighborhood Descent method, also described in (Hansen & Mladenovic, 2001), is very similar to the Variable Neighborhood Search method, save for the absence of the Shaking step.

3.6.3 2 Opt

The 2 opt neighborhood can be generated from an initial tour by removing 2 edges, and then reconnecting the tour in a different arrangement. In the 2 opt case, only one rearranged tour can be generated from an initial tour. Once the rearranged tours have been generated, infeasible tours are removed, and the tour with the lowest cost is chosen as the new initial tour, as shown in Figure 3-9.

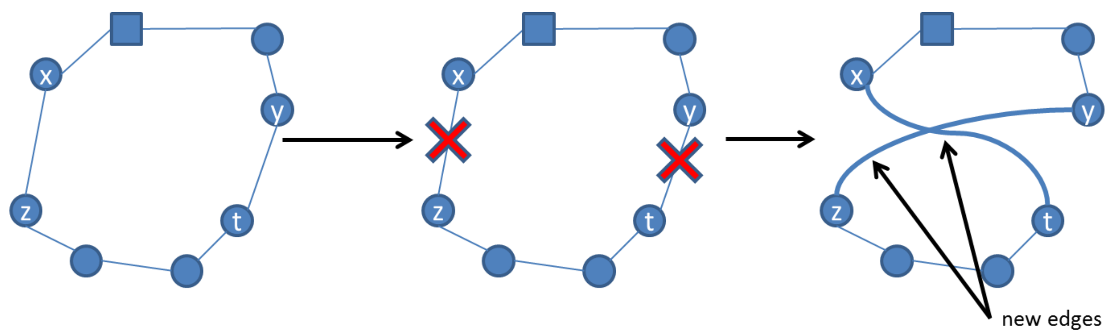


Figure 3-9 2-opt Move

3.6.4 3 Opt

The 3 opt neighborhood is generated in a similar fashion to the 2 opt neighborhood; 3 edges are removed from the initial tour, and the tour is then reconnected in 7 different arrangements (shown in Figure 3-10). The feasible configurations are then added to the neighborhood.

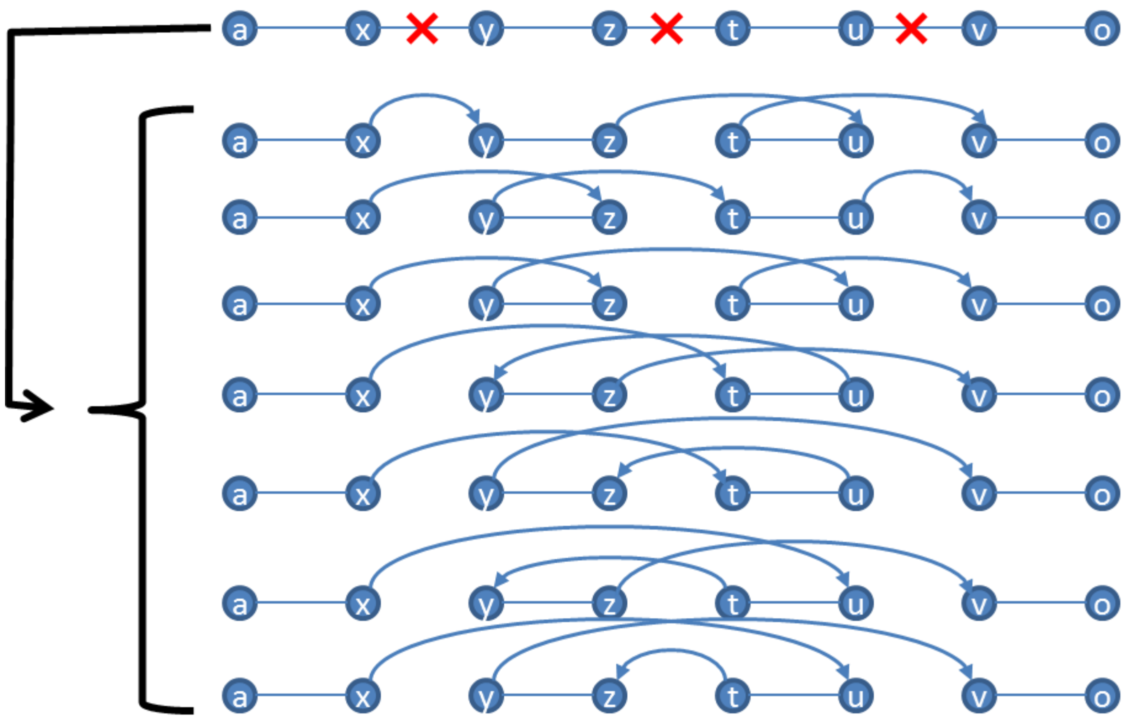


Figure 3-10 3-opt Move

3.6.5 Depot Exchange

The depot exchange neighborhood is the simplest of the neighborhoods. To form this neighborhood, visits to depots that are not the vehicle's starting depot are first identified in the initial tour. For each visit to depot *D*, all other depots in the graph are substituted. Tours with this substitution that are feasible are added to the neighborhood. Figure 3-11 shows an example depot exchange move.

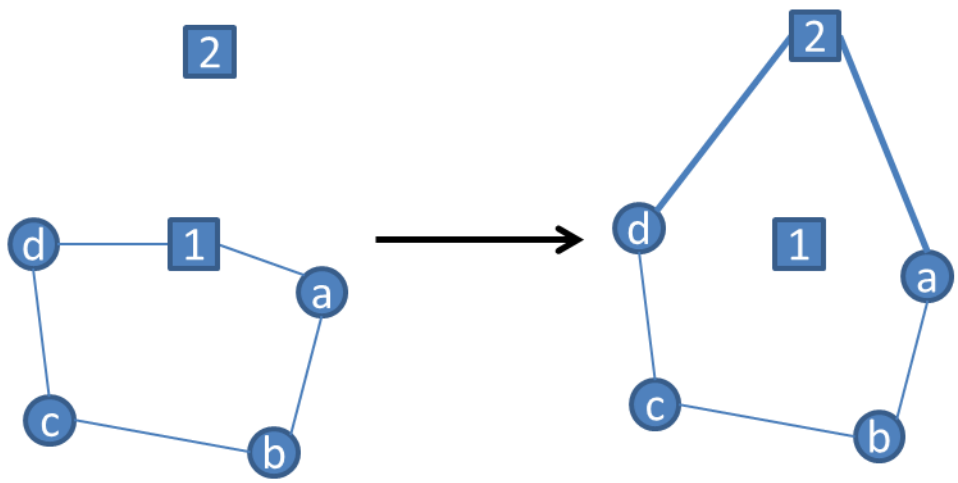


Figure 3-11 Depot Exchange Move

3.6.6 Target Vehicle Exchange

To build the target vehicle exchange neighborhood, all possible pairs of vehicles are determined. Then, for each pair, one vehicle is chosen as the donor vehicle and the other is designated as the beneficiary vehicle. Then, each target visited by the donor vehicle is inserted into the tour of the beneficiary vehicle in as many locations as possible. The configurations that retain the feasibility of both the donor vehicle's tour and the beneficiary vehicle's tour are added to the neighborhood. Figure 3-12 shows a possible Target Vehicle Exchange move, where a target visit is chosen in the tour of the first vehicle, and possible insertions of this target into the tour of the second vehicle are checked for feasibility, and improvement.

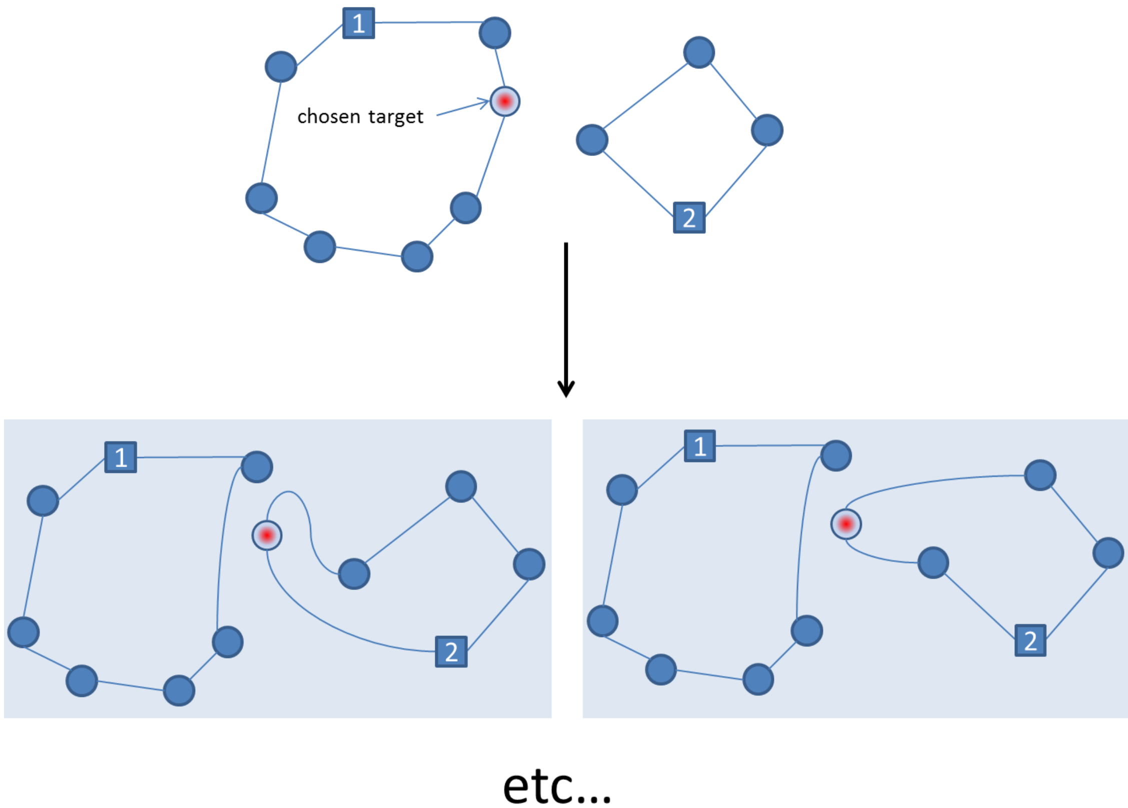


Figure 3-12 Target Vehicle Exchange Example

4 RESULTS

4.1 Implementation

Implementation of the algorithms discussed in Section 3 was achieved by way of three separate executables: apd_nographics, lkh, and mv. apd_nographics and mv were written by the student, and implement the approximate primal-dual algorithm (Bae & Rathinam, 2011) and the multiple vehicle algorithm, respectively. The lkh executable was the reference implementation of the Lin-Kernighan heuristic (Helsgaun, 2012). To facilitate communication and transfer of data between the processes, two file formats were developed: APD and MVGS.

4.2 Neighborhood Configuration Investigation

The above multiple vehicle algorithm was run for 23 problem instances, with names ranging from p01 to p21, and pr01 to pr21. Complete detailed results for these instances are contained in Appendix A. However, a summary of the interesting results are discussed here.

First, the effect of the order of neighborhoods was investigated when 4 neighborhoods were considered, and the best results are listed in Table 4-1.

Table 4-1 Effect of Neighborhood Order (4 Neighborhoods)

Scheme	Partition	N1	N2	N3	N4	Avg Improvement (percent)	Avg. Time (secs)
VND	APD	2-opt	TVE	3-opt	Depex	33.92%	78.04881
VND	APD	2-opt	TVE	Depex	3-opt	33.79%	77.33881
VND	APD	2-opt	3-opt	TVE	Depex	33.76%	197.7543
VNS	APD	TVE	2-opt	3-opt	Depex	27.52%	70.8876
VNS	APD	TVE	2-opt	Depex	3-opt	27.39%	75.12725
VNS	APD	TVE	Depex	2-opt	3-opt	26.99%	75.44715
VND	Voronoi	Depex	TVE	3-opt	2-opt	38.08%	310.7845
VND	Voronoi	TVE	3-opt	Depex	2-opt	37.94%	304.7469
VND	Voronoi	Depex	TVE	2-opt	3-opt	36.85%	203.9593
VNS	Voronoi	TVE	Depex	2-opt	3-opt	42.22%	221.0236
VNS	Voronoi	TVE	Depex	3-opt	2-opt	38.23%	741.5726
VNS	Voronoi	Depex	TVE	3-opt	2-opt	37.52%	799.8579

At first, it can be seen that the order does not significantly affect improvement percent, when observation is constrained to a specific scheme. However, the Variable Neighborhood Descent produces better final results, with higher average improvement percentages. Additionally, the 2-opt, Depex, TVE, 3-opt configuration produced the best improvement percentages, while requiring an order of magnitude less time than the other configurations, on average. Considering the neighborhood selection behavior of the VND and VNS methods, where the first neighborhood is selected as the active neighborhood whenever a better solution is found, a conclusion can be drawn that the 2-opt neighborhood produces comparable (if not better) results than the 3-opt neighborhood, while taking significantly less time to run.

Next, the effect of neighborhood order was investigated for three neighborhoods. The best results are listed in Table 4-2.

Table 4-2 Effect of Neighborhood Order (3 Neighborhoods)

Scheme	Partition	N1	N2	N3	Avg Improvement (pct)	Avg Time (secs)
VND	APD	2-opt	TVE	3-opt	33.99%	483.1837
VND	APD	2-opt	3-opt	TVE	33.86%	595.6045
VND	APD	TVE	3-opt	2-opt	33.49%	89.5769
VNS	APD	TVE	2-opt	3-opt	27.52%	71.3513
VNS	APD	TVE	3-opt	2-opt	26.77%	178.1168
VNS	APD	2-opt	TVE	3-opt	25.87%	79.254
VND	Voronoi	TVE	Depex	3-opt	42.59%	323.1872
VND	Voronoi	TVE	2-opt	3-opt	36.24%	225.4593
VND	Voronoi	Depex	TVE	3-opt	35.97%	891.1418
VNS	Voronoi	TVE	2-opt	3-opt	37.37%	241.9253
VNS	Voronoi	TVE	3-opt	Depex	37.18%	298.7079
VNS	Voronoi	Depex	TVE	3-opt	36.93%	795.2278

Again, the results show that the VND scheme produces better improvement percentages than the VNS scheme. Additionally, it can be seen that the 3-opt neighborhood occurs frequently in these runs, but is mainly the last neighborhood in the configuration. This reinforces the conclusion drawn previously that 3-opt is effective, but slow, so is not preferred by the improvement schemes.

Next, configurations with 2 neighborhoods were investigated, with the best results listed in Table 4-3.

Table 4-3 Effect of Neighborhood Order (2 Neighborhoods)

Scheme	Partition	N1	N2	Avg Improvement (pct)	Avg Time (secs)
VND	APD	TVE	2-opt	32.82%	417.9984
VND	APD	2-opt	TVE	32.15%	422.4113
VND	APD	TVE	3-opt	31.99%	495.1695
VNS	APD	TVE	3-opt	22.76%	170.846
VNS	APD	3-opt	TVE	21.24%	173.4163
VNS	APD	3-opt	2-opt	20.38%	121.7078
VND	Voronoi	TVE	3-opt	39.42%	753.4336
VND	Voronoi	3-opt	TVE	33.73%	1109.158
VND	Voronoi	TVE	2-opt	30.32%	11.05305
VNS	Voronoi	TVE	3-opt	33.93%	503.4305
VNS	Voronoi	3-opt	TVE	29.22%	479.8828
VNS	Voronoi	TVE	2-opt	27.92%	7.714526

These results exhibit the same behavior as the above tables with regards to the improvement scheme, but provide a clearer picture of the overall improvement capability of the different neighborhoods. From Table 4-3, it is immediately seen that the Target Vehicle Exchange (TVE) and 3-opt neighborhoods appear most frequently, followed by the 2-opt neighborhood. This indicates that TVE and 3-opt are the most effective neighborhoods, even though they take one or two orders of magnitude longer to run than other neighborhoods.

Finally, configurations with only one neighborhood were investigated. The best of these runs are tabulated in Table 4-4.

Table 4-4 Effect of Neighborhood Order (1 Neighborhood)

Scheme	Partition	N1	Avg Improvement (pct)	Avg Time (secs)
VND	APD	3-opt	22.96%	1462.62
VND	APD	2-opt	21.96%	420.1842
VND	APD	TVE	14.08%	1.017391
VNS	APD	3-opt	17.32%	109.5311
VNS	APD	2-opt	10.36%	0.75705
VNS	APD	TVE	8.13%	0.30115
VND	Voronoi	3-opt	21.29%	835.2096
VND	Voronoi	TVE	17.56%	1.6659
VND	Voronoi	2-opt	15.56%	4.2504
VNS	Voronoi	3-opt	19.92%	459.5533
VNS	Voronoi	2-opt	13.19%	3.118789
VNS	Voronoi	TVE	8.87%	0.399421

Results in this table are not surprising when the previous tables are considered. These results exhibit the same behavior regarding improvement scheme and neighborhood selection. However, this set of results makes it easier to see the relative execution times of the three best neighborhoods; 3-opt generally takes an order of magnitude longer than other neighborhoods, but provides the best improvement percent of the three top neighborhoods.

4.3 Voronoi Partitioning Investigation

To determine the effectiveness of the Approximate Primal-Dual algorithm in determining vehicle partitions, runs were performed using a simple Voronoi partitioning scheme, where targets are assigned to the vehicle whose starting depot is closest to the target. The costs of the solutions output by the construction heuristic using the Voronoi partitions, relative to the same costs using the APD algorithm are listed in Table 4-5.

Table 4-5 Voronoi Construction Heuristic Costs

Instance	APD Cost	Voronoi Cost	Voronoi Pct
p01	2483.04	2474.20	99.64%
p03	2170.27	3666.57	168.95%
p04	2569.91	4354.53	169.44%
p05	2588.21	2645.91	102.23%
p06	5089.49	4158.92	81.72%
p07	5759.59	5798.38	100.67%
p08	40044.10	59003.20	147.35%
p09	37783.80	51802.70	137.10%
p10	34990.60	50382.60	143.99%
p11	34610.90	71234.30	205.81%
p12	5917.04	12618.00	213.25%
p15	23619.60	9628.57	40.77%
p21	77354.60	19131.90	24.73%
pr01	6964.58	9940.97	142.74%
pr02	7625.70	7854.82	103.00%
pr03	21303.90	29336.80	137.71%
pr04	14659.20	19382.00	132.22%
pr05	9780.66	24196.30	247.39%
pr06	24778.50	9144.85	36.91%
pr07	4026.09	10006.30	248.54%

Table 4-5 Continued

Instance	APD Cost	Voronoi Cost	Voronoi Pct
pr08	9340.81	14959.80	160.16%
pr09	12207.20	16921.70	138.62%
pr10	9119.63	30491.90	334.35%

From Table 4-5, it can be seen that the cost of the Voronoi partitions were generally more than that of the APD partitions, save for a few outliers where the Voronoi partitions were slightly cheaper.

However, the costs of the solutions output by the construction heuristic do not tell the entire story. Therefore, the Voronoi partitioned instances were improved using the same neighborhoods and schemes as their APD partitioned counterparts, and selected results are listed in Table 4-6. In this table, the 7th column shows the final cost using the Voronoi partitions as a percentage of the final cost using the APD partitions. The 6th column is provided as a reference.

Table 4-6 Comparison Between Improved APD and Voronoi Runs

Scheme	N1	N2	N3	N4	APD Improved Cost (pct)	Voronoi Improved Cost (pct)
VNS	TVE	2-opt	Depex	3-opt	100.00%	90.10%
VND	Depex	2-opt	TVE	3-opt	100.00%	95.58%
VND	Depex	2-opt	TVE	3-opt	100.00%	95.58%
VNS	TVE	3-opt	Depex	3-opt	100.00%	118.64%
VND	3-opt	TVE	Depex		100.00%	98.20%
VNS	TVE	2-opt	3-opt		100.00%	91.27%
VND	TVE	2-opt	3-opt		100.00%	94.15%
VNS	TVE	2-opt			100.00%	101.52%
VND	TVE	2-opt			100.00%	93.53%
VNS	3-opt	2-opt			100.00%	135.45%
VND	TVE	3-opt			100.00%	97.90%
VNS	TVE				100.00%	151.36%
VND	2-opt				100.00%	103.04%
VND	TVE				100.00%	134.98%
					Average:	**107.24%**

From Table 4-6, it can be seen that the instances that used the Voronoi partitions did about the same, on average, than their APD counterparts. In fact, the average Voronoi cost percentage over all runs was 107.24%. This indicates that selection of partition method is not a very important factor in final solution quality.

4.4 Total Best Combinations

The averages of all the improvement percentages were used to determine the best configurations for the 1, 2, 3, and 4 neighborhood groups. These configurations were then run for every instance to ensure a complete result set.

4.4.1 Best 1 Combination

The best 1 neighborhood group was found to be 3-opt, without shaking and with the APD partition method. These runs averaged a 22.96% improvement percentage. Results from these runs are shown in Table 4-7.

Table 4-7 Best 1 Combination Results

Instance	Start Cost	Impr Cost	Impr Pct	Secs
p01	2483.04	1942.48	21.77%	7.368
p03	2170.27	1172.44	45.98%	4.822
p04	2569.91	1301.3	49.36%	15.816
p05	2588.21	1468.28	43.27%	30.267
p06	5089.49	2904.23	42.94%	180.685
p07	5759.59	4456.7	22.62%	195.286
p08	40044.1	40044.1	0.00%	192.995
p09	37783.8	37783.8	0.00%	142.009
p10	34990.6	34990.6	0.00%	374.885
p11	34610.9	34610.9	0.00%	30.816
p12	5917.04	2064.31	65.11%	25.047
p15	23619.6	3105.85	86.85%	5842.052
p21	5564.69	4889.38	12.14%	28.90068
p21	77354.6	76936.9	0.54%	9000
pr01	6964.58	6885.05	1.14%	11.20039
pr02	7625.7	7625.7	0.00%	9.709453
pr03	21303.9	21303.9	0.00%	29.87566
pr04	14659.2	13465.5	8.14%	943.9597
pr05	9780.66	8137.73	16.80%	286.9558
pr06	24778.5	14108.8	43.06%	32975.92
pr07	4026.09	2151.59	46.56%	2.17765
pr08	9340.81	7385.68	20.93%	178.9951
pr09	12207.2	9294.9	23.86%	480.1001
pr10	9119.63	9119.63	0.00%	50.35585

As can be seen, some of the instances (the larger ones) were unable to find any improvement for this configuration. This is a common trend in the result set; it seems that the larger instances started in local optima more frequently than the smaller instances.

4.4.2 Best 2 Combination

The best 2 combination was found to have TVE as the first neighborhood, and 3-opt for the second, using the VND scheme and the Voronoi partitions. For this configuration, the improvement percentages averaged 39.42%. Table 4-8 lists these results.

Table 4-8 Best 2 Combination Results

Instance	Start Cost	Impr Cost	Impr Pct	Secs
p01	2474.2	1096.8	55.67%	1.012
p03	3666.57	1253.25	65.82%	16.18
p04	4354.53	1762.64	59.52%	206.734
p05	2645.91	1465.84	44.60%	38.875
p06	4158.92	1829.36	56.01%	22.65
p07	5798.38	2224.81	61.63%	48.702
p08	59003.2	59003.2	0.00%	518.078
p09	51802.7	51802.7	0.00%	557.311
p10	50382.6	50382.6	0.00%	479.043
p11	71234.3	67496.8	5.25%	9570.828
p12	12618	1573.43	87.53%	8.731
p15	9628.57	2398.17	75.09%	23.964
p21	19131.9	5543.1	71.03%	58.40544
pr01	9940.97	9940.97	0.00%	2.577926
pr02	7854.82	2425.53	69.12%	16.12673
pr03	29336.8	22436	23.52%	4844.833
pr04	19382	18776.1	3.13%	508.4298

Table 4-8 Continued

Instance	Start Cost	Impr Cost	Impr Pct	Secs
pr05	24196.3	17398.5	28.09%	4531.525
pr06	9144.85	5088.55	44.36%	252.208
pr07	10006.3	5294.35	47.09%	2.493464
pr08	14959.8	8907.27	40.46%	142.704
pr09	16921.7	5272.41	68.84%	100.4253
pr10	30491.9	30491.9	0.00%	46.63218

This configuration shows typically longer run times than that of the best 1 combination, but with a higher average improvement percentage. This indicates that the TVE neighborhood is effective for solution improvement.

4.4.3 Best 3 Combination

The best 3 combination configuration used TVE, Depot Exchange, and 3-opt neighborhoods, without shaking, and using the Voronoi partitioning method. These runs averaged a 42.59% for improvement percent, and are shown in Table 4-9.

Table 4-9 Best 3 Combination Results

Instance	Start Cost	Impr Cost	Impr Pct	Secs
p01	2474.2	1096.8	55.67%	0.962
p03	3666.57	1253.25	65.82%	16.196
p04	4354.53	1634.81	62.46%	186.685
p05	2645.91	1465.84	44.60%	38.83

Table 4-9 Continued

Instance	Start Cost	Impr Cost	Impr Pct	Secs
p06	4158.92	1829.36	56.01%	22.633
p07	5798.38	2224.81	61.63%	48.9
p08	59003.2	59003.2	0.00%	515.802
p09	51802.7	51802.7	0.00%	554.835
p10	50382.6	50382.6	0.00%	476.858
p11	19327.5	4599.05	76.20%	212.3943
p12	12618	1573.43	87.53%	8.829
p15	9628.57	2398.17	75.09%	24.106
p21	19131.9	5391.13	71.82%	89.22671
pr01	9940.97	9940.97	0.00%	2.634846
pr02	7854.82	2435.17	69.00%	14.46503
pr03	29336.8	22475.2	23.39%	4365.047
pr04	19382	18776.1	3.13%	541.5995
pr05	24196.3	17403.7	28.07%	4190.062
pr06	9144.85	5165.15	43.52%	235.299
pr07	10006.3	5510.2	44.93%	2.12073
pr08	14959.8	8689.09	41.92%	179.7975
pr09	16921.7	5289.71	68.74%	94.15305
pr10	30491.9	30491.9	0.00%	46.17499

These runs took less time than the best 2 combination runs, but provided a slightly better average improvement percentage. This indicates that the Depot Exchange neighborhood provides some improvement at a negligible run time increase.

4.4.4 Best 4 Combination

The best configuration with 4 neighborhoods was TVE, Depot Exchange, 2-opt, and 3-opt, using a Variable Neighborhood Search and Voronoi partitions. These runs averaged a 42.22% improvement, and are shown in Table 4-10.

Table 4-10 Best 4 Combination

Instance	Start Cost	Impr Cost	Impr Pct	Secs
p01	2474.2	694.271	71.94%	0.232
p03	3666.57	1143.62	68.81%	4.915
p04	4354.53	1551.03	64.38%	58.451
p05	2645.91	1476.75	44.19%	19.571
p06	4158.92	1474.53	64.55%	12.117
p07	5798.38	1464.33	74.75%	26.84
p08	59003.2	59003.2	0.00%	519.381
p09	51802.7	51802.7	0.00%	559.957
p10	50382.6	50346.4	0.07%	470.61
p11	71234.3	65498.1	8.05%	1910.843
p12	12618	2385.06	81.10%	53.939
p15	9628.57	2409.39	74.98%	11.693
p21	19131.9	5492.56	71.29%	78.08324
pr01	9940.97	9940.97	0.00%	2.691766
pr02	7854.82	2107.41	73.17%	14.26489
pr03	29336.8	22494.6	23.32%	974.1806
pr04	19382	18603.2	4.02%	213.7126
pr05	15198.4	9801.09	35.51%	484.938
pr06	9144.85	5240.83	42.69%	98.146
pr07	10006.3	4827.39	51.76%	1.632319
pr08	14959.8	8357.23	44.14%	64.22047
pr09	16921.7	4676.42	72.36%	168.083
pr10	30491.9	30473.4	0.06%	47.34827

These runs took longer, thanks to the addition of the 2-opt neighborhood, but did not offer much improvement over the best 3 group. This is probably due to the overlap in the solution spaces of the 3-opt and 2-opt neighborhoods.

5 CONCLUSION

To examine the effects of different neighborhoods and different schemes on solution improvement for a multiple vehicle routing problem with fuel constraints, two algorithms were implemented in two executables: the apd.exe executable and the mv.exe executable. These implement the Approximate Primal-Dual algorithm and the Multiple Vehicle algorithm, respectively. To facilitate result analysis and program intercommunication, two file formats were created: the .apd file format and the .mv file format.

Once these implementations were completed, four neighborhoods were selected: 2-opt, 3-opt, Depot Exchange, and Target Vehicle Exchange. Then, all combinations of one, two, three, and four of these were run on 23 instances, utilizing the Variable Neighborhood Descent and Variable Neighborhood Search schemes, and the APD and Voronoi partition methods. Once as many of these runs were completed as possible (some instances required too much time to complete), certain combinations of neighborhoods, scheme, and partition methods were chosen to investigate their effects on solution improvement.

The first observation that was made was that the Variable Neighborhood Descent produced better solution improvement than the Variable Neighborhood Search. This is a counter-intuitive result because the shaking step of the VNS was intended to break out of local minima. The next observation was that the 2-opt neighborhood provides improvement comparable to the 3-opt neighborhood, in an order of magnitude less time. Finally, the effect of neighborhood order on improvement was investigated. This found that the best improvement percentages were achieved when 2-opt was the first neighborhood. Additionally, these runs completed in less time than other configurations, because the VND and VNS schemes utilize the first neighborhood more often than the others. Then the 2-opt neighborhood is used more, and its small solution space becomes an advantage, allowing a quick approach to a local minima. The effect of the partition method (either the Approximate Primal-Dual algorithm, or the Voronoi partitions) was investigated, and it was found that while the Voronoi partitions produced solutions with a higher starting cost, the final costs after improvement were close to that of the APD partitions. This suggests that the improvement step is somewhat resilient to the initial solution given to it.

The final investigation performed against the run data was to find the best configurations that include only one, two, three, and four neighborhoods. The configuration groups were sorted by improvement percent, and resulted in the following best configurations:

1. VND, APD: 3-opt
2. VND, Voronoi: TVE, 3-opt
3. VND, Voronoi: TVE, Depot Exchange, 3-opt
4. VNS, Voronoi: TVE, Depot Exchange, 2-opt, 3-opt

These results imply that the neighborhoods can be ordered in descending improvement percentage: 3-opt, TVE, Depot Exchange, and 2-opt.

REFERENCES

Army, U. S. (2007, October 18). *Soldiers Train With Raven UAVs*. Retrieved January 20, 2013, from United States Army: http://www.army.mil/article/5644/soldiers-train-with-raven-uavs/

Bae, J., & Rathinam, S. (2011). A Primal-Dual Algorithm for a Heterogeneous Traveling Salesman Problem. *arXiv*, http://arxiv.org/abs/1111.0567.

Curry, J. A., Maslanik, J., Holland, G., & Pinto, J. (2004). Applications of Aerosondes in the Arctic. *Bulletin of the American Meteorological Society*, 1855-1861.

Dell, R. F., Batta, R., & Karwan, M. H. (1996, May). The Multiple Vehicle TSP with Time Windows and Equity Constraints over a Multiple Day Horizon. *Transportation Science, 30*(2).

Hansen, P., & Mladenovic, N. (2001). Variable Neighborhood Search: Principles and Applications. *European Journal of Operational Research*, 449-467.

Helsgaun, K. (2012, August). *LKH.* Retrieved September 2012, from Research Page of Keld Helsgaun: http://www.akira.ruc.dk/~keld/research/LKH/

Khuller, S., Malekian, A., & Mestre, J. (2011, July). To Fill or Not To Fill: The Gas Station Problem. *ACM Transactions on Algorithms, 7*(3).

Oberlin, P. (2009, May). Path Planning Algorithms for Multiple Heterogeneous Vehicles. *Master's Thesis*. College Station, Texas: Texas A&M University.

Rangarajan, R. (2011, May). Approximation Algorithms and Heuristics for a Heterogeneous Traveling Salesman Problem. *Master's Thesis*. College Station, Texas: Texas A&M University.

Rathinam, S., & Sengupta, R. (2006, March). *Lower and Upper Bounds for a Symmetric Multiple Depot, Multiple Traveling Salesman Problem.* Berkely, CA: Institute of Transportation Studies.

Sundar, K., & Rathinam, S. (2013, April). Algorithms for Routing an Unmanned Aerial Vehicle in the Presence of Refueling Depots. *arXiv*, http://arxiv.org/abs/1304.0494.

APPENDIX A

FULL RESULTS

Table 5-1 summarizes the result set from the runs performed for this paper. For each combination of instance, scheme, and partition method, the improved cost of the best combinations with 1, 2, 3, and 4 neighborhoods are listed, respectively. Additionally, the starting cost is listed (the cost after the construction heuristic).

Table 5-1 - Result Summary

				Best 1 Combo		Best 2 Combo		Best 3 Combo		Best 4 Combo	
Instance	Scheme	Partition	Start Cost	Impr Cost	Secs Taken	Impr Cost	Secs Taken	Impr Cost	Secs Taken	Impr Cost	Secs Taken
p01	VNS	APD	2483.04	1942.48	8.290	1148.07	2.602	1029.72	1.716	813.37	1.125
p01	VNS	Voronoi	2474.20	1929.48	0.024	792.15	0.274	537.45	0.265	537.45	0.266
p01	VND	APD	2483.04	1163.14	0.149	1051.55	0.184	1051.55	0.179	1029.72	8.438
p01	VND	Voronoi	2474.20	1499.81	0.050	1047.64	3.993	890.03	1.599	890.03	1.615
p03	VNS	APD	2170.27	1617.30	0.313	1062.95	2.185	1036.81	2.525	963.07	1.062
p03	VNS	Voronoi	3666.57	1801.66	9.760	1411.49	10.434	1120.64	5.349	1099.25	6.356
p03	VND	APD	2170.27	1172.44	4.822	1158.89	5.021	1138.59	1.719	1134.36	1.792
p03	VND	Voronoi	3666.57	1801.66	9.783	1253.25	16.180	1177.55	14.394	1177.55	14.324
p04	VNS	APD	2569.91	1301.30	15.748	1298.91	5.206	1292.39	4.118	1281.77	5.195
p04	VNS	Voronoi	4354.53	2832.90	182.724	2770.80	349.353	1663.61	114.348	1551.03	58.451
p04	VND	APD	2569.91	1301.30	15.816	1301.30	15.758	1301.30	15.943	1301.30	15.890
p04	VND	Voronoi	4354.53	2809.46	194.482	1753.87	227.212	1543.85	224.807	1474.78	32.102
p05	VNS	APD	2588.21	1478.15	34.644	1477.15	18.722	1473.29	16.819	1459.48	15.729
p05	VNS	Voronoi	2645.91	1465.84	39.486	1465.84	39.472	1465.84	39.655	1457.50	19.640
p05	VND	APD	2588.21	1468.28	30.267	1468.28	30.110	1468.28	30.329	1468.28	30.308
p05	VND	Voronoi	2645.91	1465.84	38.858	1465.84	38.861	1465.84	39.109	1465.84	39.484
p06	VNS	APD	5089.49	3428.53	205.309	2894.21	134.432	2883.44	136.893	2876.65	83.280
p06	VNS	Voronoi	4158.92	3414.69	0.208	1052.84	18.011	1052.84	18.036	1052.84	18.073

Table 5-1 Continued

				Best 1 Combo		Best 2 Combo		Best 3 Combo		Best 4 Combo	
Instance	**Scheme**	**Partition**	**Start Cost**	**Impr Cost**	**Secs Taken**	**Impr Cost**	**Secs Taken**	**Impr Cost**	**Secs Taken**	**Impr Cost**	**Secs Taken**
p06	VND	APD	5089.49	2904.23	180.685	2904.23	181.884	2904.23	181.093	2904.23	181.794
p06	VND	Voronoi	4158.92	2624.25	0.959	1829.36	22.650	1776.17	277.707	1752.59	8.741
p07	VNS	APD	5759.59	4456.70	197.979	3294.58	862.749	2392.51	2.833	2392.40	3.502
p07	VNS	Voronoi	5798.38	4038.83	3.191	1928.29	53.184	1558.05	181.364	1464.33	26.840
p07	VND	APD	5759.59	2962.01	2.306	2395.09	274.106	2395.09	277.892	2395.09	277.484
p07	VND	Voronoi	5798.38	3829.60	123.154	2224.81	48.702	2136.65	48.252	2114.11	24.676
p08	VNS	APD	40044.10	40044.10	0.904	40044.10	194.059	39793.90	568.133	39730.70	579.968
p08	VNS	Voronoi	59003.20	59003.20	1.208	59003.20	521.369	58868.50	1029.656	58868.50	1034.539
p08	VND	APD	40044.10	40044.10	0.638	40044.10	193.823	40044.10	192.518	40044.10	192.859
p08	VND	Voronoi	59003.20	59003.20	0.909	59003.20	522.883	59003.20	522.492	59003.20	521.843
p09	VNS	APD	37783.80	37586.20	1.261	37586.20	2.112	37586.20	140.221	37586.20	142.557
p09	VNS	Voronoi	51802.70	51802.70	0.945	51796.40	0.739	51562.60	1633.865	51562.60	1656.211
p09	VND	APD	37783.80	37783.80	0.696	37783.80	147.777	37783.80	143.700	37783.80	143.602
p09	VND	Voronoi	51802.70	51802.70	0.720	51802.70	546.646	51802.70	559.428	51802.70	564.941
p10	VNS	APD	34990.60	34990.60	1.610	34990.60	372.422	34990.60	381.882	34990.60	369.433
p10	VNS	Voronoi	50382.60	50346.40	0.713	50346.40	1.555	50289.00	930.216	50289.00	956.784
p10	VND	APD	34990.60	34990.60	1.520	34990.60	381.347	34990.60	377.317	34990.60	375.210
p10	VND	Voronoi	50382.60	50382.60	0.677	50382.60	481.611	50382.60	482.115	50382.60	475.982
p11	VNS	APD	34610.90	34610.90	0.399	32694.60	397.689	31863.90	13.479	31731.40	70.013
p11	VNS	Voronoi	71234.30	67871.30	33.827	67509.30	7733.787	65936.30	1610.337	65498.10	1910.843
p11	VND	APD	34610.90	30679.70	9.765	30558.60	1520.315	30558.60	1474.569	30558.60	1486.398
p11	VND	Voronoi	71234.30	67496.80	9505.046	67496.80	9532.939	4599.05	115.675	67496.80	9549.344
p12	VNS	APD	5917.04	3069.05	32.756	2088.40	5.984	2054.97	7.104	1979.15	5.916
p12	VNS	Voronoi	12618.00	3552.20	641.124	2707.32	155.453	1559.21	84.632	1559.21	84.086
p12	VND	APD	5917.04	2064.31	25.047	2039.37	2.865	2039.37	2.835	2039.37	2.852
p12	VND	Voronoi	12618.00	1556.68	661.698	1545.49	25.355	1545.49	25.492	1545.49	25.657
p15	VND	APD	23619.60	3105.85	5842.052	3105.85	5935.886	3080.96	349.839	3105.85	5870.366
p21	VND	APD	5564.69	4889.38	15.740	24147.60	9000.000	49162.80	9000.000	-	-
pr01	VNS	APD	6964.58	6885.05	6.318	6885.05	9.594	6885.05	8.066	6885.05	7.956

Table 5-1 Continued

				Best 1 Combo		Best 2 Combo		Best 3 Combo		Best 4 Combo	
Instance	**Scheme**	**Partition**	**Start Cost**	**Impr Cost**	**Secs Taken**	**Impr Cost**	**Secs Taken**	**Impr Cost**	**Secs Taken**	**Impr Cost**	**Secs Taken**
pr01	VNS	Voronoi	9940.97	9940.97	0.000	9742.18	5.616	9711.07	0.062	9711.07	2.730
pr01	VND	APD	6964.58	6885.05	6.100	6885.05	5.881	6885.05	5.850	6885.05	6.038
pr01	VND	Voronoi	9940.97	9940.97	0.016	9940.97	1.451	9940.97	1.435	9940.97	1.419
pr02	VNS	APD	7625.70	7209.52	0.032	5671.24	78.328	5671.24	75.723	5596.61	121.401
pr02	VNS	Voronoi	7854.82	5187.07	0.733	2226.53	13.697	2047.71	6.630	2047.71	6.583
pr02	VND	APD	7625.70	3001.42	3.494	3001.42	3.526	3001.42	3.744	3001.42	3.884
pr02	VND	Voronoi	7854.82	4326.63	20.920	2425.53	8.783	2145.94	12.199	2145.94	12.246
pr03	VNS	APD	21303.90	21303.90	0.125	21303.90	17.160	21303.90	26.302	21114.30	32.651
pr03	VNS	Voronoi	29336.80	24552.40	1964.436	22713.00	2656.131	22423.60	1920.069	22423.60	2287.047
pr03	VND	APD	21303.90	21303.90	0.093	21303.90	16.318	21303.90	16.177	21303.90	16.395
pr03	VND	Voronoi	29336.80	24528.40	2153.634	22436.00	2638.612	22369.10	496.132	22369.10	508.206
pr04	VNS	APD	14659.20	13465.50	545.101	13463.20	694.660	13456.30	410.674	13429.00	348.773
pr04	VNS	Voronoi	19382.00	18776.10	286.185	18620.90	2.871	18544.80	51.527	18387.50	133.377
pr04	VND	APD	14659.20	13465.50	514.103	13448.50	318.275	13448.50	310.319	13448.50	313.392
pr04	VND	Voronoi	19382.00	18776.10	296.793	18776.10	283.548	18776.10	290.569	18776.10	281.411
pr05	VNS	APD	9780.66	8137.73	157.952	8129.20	172.616	7954.24	148.545	7954.24	147.078
pr05	VND	APD	9780.66	8137.73	156.283	8108.92	183.318	8108.92	191.804	8108.92	184.020
pr05	VND	Voronoi	24196.30	18372.10	8.237	17398.50	2467.977	17235.50	2788.436	17399.40	4609.273
pr06	VND	APD	24778.50	14108.80	17959.474	14108.80	17795.797	-	-	-	-
pr07	VNS	APD	4026.09	2140.68	1.435	2098.52	1.419	2087.64	1.311	2087.64	1.279
pr07	VNS	Voronoi	10006.30	6104.24	0.218	5105.09	0.484	4884.46	0.530	4827.39	0.889
pr07	VND	APD	4026.09	2151.59	1.186	2090.29	1.248	2090.29	1.295	2090.29	1.357
pr07	VND	Voronoi	10006.30	6271.34	0.171	5224.70	0.421	5175.43	0.764	5175.43	0.811
pr08	VNS	APD	9340.81	7385.68	94.069	7385.68	98.032	7143.78	15.273	7133.24	29.453
pr08	VNS	Voronoi	14959.80	10765.70	64.257	9395.85	72.447	8566.96	81.963	8357.23	34.976
pr08	VND	APD	9340.81	7385.68	97.485	7385.68	92.899	7312.80	54.491	7312.80	56.784
pr08	VND	Voronoi	14959.80	10765.70	62.478	8907.27	77.720	8689.09	97.922	8659.87	21.185
pr09	VNS	APD	12207.20	7881.07	2.012	5758.65	25.928	5417.64	30.825	5417.64	31.512
pr09	VNS	Voronoi	16921.70	12113.80	340.661	7097.05	21.638	4543.79	118.234	4422.08	108.281
pr09	VND	APD	12207.20	6325.20	4.945	5706.90	461.125	5598.21	179.558	5598.21	191.975
pr09	VND	Voronoi	16921.70	7489.55	14.804	5272.41	54.694	4983.53	40.966	4767.24	60.279

Table 5-1 Continued

				Best 1 Combo		Best 2 Combo		Best 3 Combo		Best 4 Combo	
Instance	**Scheme**	**Partition**	**Start Cost**	**Impr Cost**	**Secs Taken**	**Impr Cost**	**Secs Taken**	**Impr Cost**	**Secs Taken**	**Impr Cost**	**Secs Taken**
pr10	VNS	APD	9119.63	6922.19	0.468	6027.47	1.029	4701.49	2.028	4701.49	6.973
pr10	VNS	Voronoi	30491.90	30473.40	0.655	30468.60	76.145	30332.70	1.669	30332.70	27.877
pr10	VND	APD	9119.63	4044.72	1.389	4044.72	9.282	4044.72	728.606	4044.72	737.623
pr10	VND	Voronoi	30491.90	30491.90	0.109	30491.90	25.365	30491.90	25.335	30491.90	25.241