# An ACO-Inspired, Probabilistic, Greedy Approach

# to the Drone Traveling Salesman Problem

Jessica Houseknecht

A Senior Thesis submitted in partial fulfillment
of the requirements for graduation
in the Honors Program
Liberty University
Spring 2019

Acceptance of Senior Honors Thesis

This Senior Honors Thesis is accepted in partial fulfillment of the requirements for graduation from the Honors Program of Liberty University.

______________________________
Andy Ham, Ph.D.
Thesis Chair

______________________________
Robert K. Rich, M.S.
Committee Member

______________________________
Scott Long, Ph.D.
Committee Member

______________________________
David Schweitzer, Ph.D.
Assistant Honors Director

______________________________
Date

## Abstract

In recent years, major companies have done research on using drones for parcel delivery. Research has shown that this can result in significant savings, which has led to the formulation of various truck and drone routing and scheduling optimization problems. This paper explains and analyzes a new approach to the Drone Traveling Salesman Problem (DTSP) based on ant colony optimization (ACO).

The ACO-based approach has an acceptance policy that maximizes the usage of the drone. The results reveal that the pheromone causes the algorithm to converge quickly to the best solution. The algorithm performs comparably to the MIP model, CP model, and EA of Rich & Ham (2018), especially in instances with a larger number of stops.

# An ACO-Inspired, Probabilistic, Greedy Approach to the Drone Traveling Salesman Problem

## Introduction

### Integrated Drone Delivery Applications

An unmanned aerial vehicle (UAV) or drone is an aircraft that operates autonomously, via remote control, or both (Guilmartin & Taylor, 2018). In 2013 during a broadcast on *60 Minutes*, Amazon's CEO, Jeff Bezos (as cited in Murray & Chu, 2015), announced the company had developed a fleet of drones. However, Carlson (2013) had reported that the so-called *Amazon Prime Air* service would not be available "for many years" (para. 2). In December of 2016, the first short unmanned aerial flight of Amazon Prime Air was made in Britain (Amazon, n.d.). It has now been five years since Bezos made this claim, but he has not given up on the prospect of using drones for delivery (Barrabi, 2018). Studies show that utilizing a drone in-tandem with a truck to make deliveries can save costs in time and fuel. Other companies including Alibaba and UPS have also experimented with drones for small parcel "last-mile" delivery (Popper, 2013).

This notion of using drones to make deliveries has helped inspire the Drone Traveling Salesman Problem (DTSP), where a drone works in-tandem with a truck to deliver parcels to customers on its route. The DTSP is an extension of the thoroughly researched traveling salesman problem (TSP). Research on various truck-drone problems has shown evidence of savings from using a drone. This paper seeks to, first, show that a new probabilistic, greedy approach to the DTSP that is based on ant colony optimization (ACO) can produce comparable results to those in the literature. Second, it seeks to show that while ACO appears to have been only theoretically discussed in the literature, its inspiration for the successful implementation in this study reveals that it is a conceivable method for application to truck-drone problems. For this reason, it is worth further investigating extensions to the ACO-based algorithm in this paper as well as utilizing ACO-based approaches in other truck-drone problems.

## The Traveling Salesman Problem (TSP)

The traveling salesman problem (TSP) is one of the most researched NP-hard combinatorial optimization problems in the literature. An NP-hard problem is one such that “there is no exact algorithm to solve it in polynomial time” and in which one cannot know whether the answer obtained is correct in polynomial time (Brezina & Čičková, 2011, p. 1). The objective of the TSP is to minimize the time or distance that it takes a traveling salesman to visit each city along his route. The basic TSP consists of a structure called a graph which consists of nodes, representing the different stops in his route, and edges, representing the paths between the cities that he can take. The objective of the TSP is to minimize the distance that a traveling salesman takes to visit every city exactly once, and then return to the starting city. This problem has been solved using several optimization techniques, including ant colony optimization, the technique motivating the approach to the DTSP explored in this study.

## The Drone Traveling Salesman Problem (DTSP)

Like the TSP, the DTSP is an NP-hard problem (Ponza, 2016). In this problem, a drone flies in-tandem with the truck to help deliver parcels. The drone rides with the truck when it is not delivering a parcel. The goal of the DTSP is to minimize the amount of time it takes to service all customers by either the drone or by the truck, visiting each customer exactly once (Rich & Ham, 2018). The drone and truck start at a depot and must return to the depot after delivering to each of the customers. A sample routing of the truck and drone is shown in Figure 1, with the solid black lines representing the truck route and the broken blue lines representing the drone route.

The drone has a limited range of $\kappa$ kilometers and moves by a speed factor $\alpha_{ds}$ of the truck's speed. While not considered in this construction, it may not be feasible for the drone to deliver a package to a customer due to other factors such as the parcel weight, "parcels requiring a signature, or customer locations not amenable to safely landing the UAV" (Murray & Chu, 2015, p. 90).

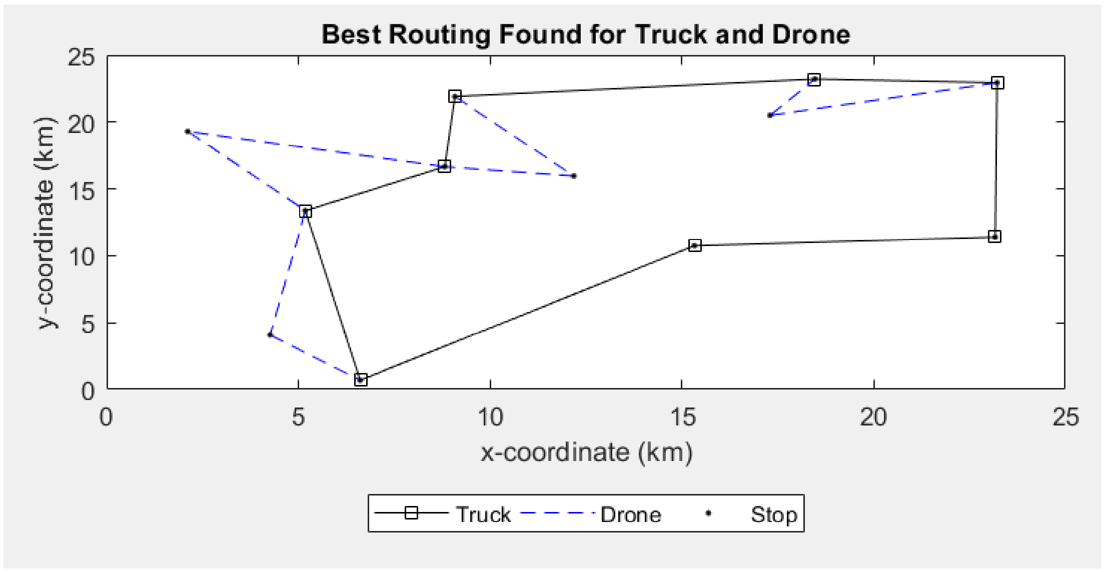

*Figure 1*. Sample routing plot output from MATLAB for the DTSP problem.

When separated from the truck, the drone travels in short trips called *sorties*, consisting of three nodes (Murray & Chu, 2015). The DTSP given by Rich & Ham (2018) is virtually the same problem as the *Flying Sidekick Traveling Salesman Problem*

(FSTSP) coined by Murray & Chu (2015), except that the problem addressed by Rich & Ham (2018) and in this paper has two additional restrictions. First, it does not allow the truck to stop at more than one location while separated from the drone. Thus, only three-tuple configurations as shown by the sortie in Figure 2 are allowed. In a sortie, the drone departs from the truck at the node *i*, which can either be the depot (where the truck and drone start their route) or the current customer; it then drops off a parcel at the second node *l*; finally, it rendezvouses with the truck at a new unvisited customer location *j* (Murray & Chu, 2015). Using this notation, sorties shall be denoted as <*i*, *l*, *j*>. The second restriction of the DTSP forces the drone to rendezvous with the truck at a customer node rather than the depot.

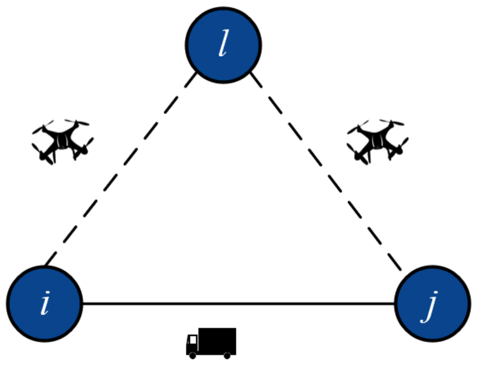

*Figure 2*. Illustration of a sortie <*i*, *l*, *j*>.

## Ant Colony Optimization (ACO)

Swarm intelligence is a field of research that examines intelligent multi-agent systems, which use autonomous agents that individually are not intelligent, but, collectively, can solve complex problems (Selvi & Umarani, 2010). Independent agents follow certain rules to cause swarm-like behavior (Marzolla & Babaoglu, 2014).

Ant colony optimization (ACO) is a type of swarm optimization metaheuristic inspired by ants in nature. It has been shown by a series of experiments called the *Double Bridge Experiments* that foraging ants, by releasing pheromone, will converge to the shortest path between their food source and nest (Dorigo & Stützle, 2004).

Ants find the shortest path between their nest and food source by releasing a chemical called pheromone. This pheromone attracts other ants, causing them to take paths with more pheromone. Over time, the pheromone accumulates on the shorter paths, due to ants moving from point A to point B in less time. In contrast, pheromone on longer paths tends to get depleted since ants take longer to move between the nest and the food source on these trails. Over several iterations, the amount of pheromone evaporating on these paths is higher than the amount of pheromone that gets laid down.

Similarly, for ant colony optimization in the TSP, artificial *ants* leave *pheromone* along the edges in a network and use other information to construct tours. The pheromone *evaporates* at a certain rate according to a learning rule, which can help discourage search in unpreferable directions (Ellabib, Calamai, & Basir, 2007). This is modeled by decreasing the values of pheromone on trails. Ant movement is stochastic, allowing for a few individuals to take normally unfavorable paths, even as pheromone builds up, to see whether a better solution may be found (Brezina & Čičková, 2011).

Several ACO algorithms were applied to the TSP. The first version of ACO was Ant System. *Ant Colony System* (which considers the best-so-far solution) and *MAX-MIN Ant System* (which considers the best-this-iteration) are the best performing ACO algorithms for the TSP. The implementation in this study uses extensions on the classical Ant System, modifying the acceptance criteria used in the TSP to solve the DTSP.

## Literature Review

### Previous Related Work on the DTSP

Several truck and drone assignment and routing problems are present in the literature. This paper solves the DTSP as presented by Rich & Ham (2018). In their paper, they use a mixed integer programming (MIP) model, a constraint programming (CP) model, and an evolutionary algorithm (EA). This problem is a more specific version of the FSTSP given by Murray & Chu (2015). The FSTSP allows the truck to visit multiple customers while the drone is in flight. Murray & Chu (2015) use an MIP model, and a *route and re-assign* heuristic that solves the TSP and, subsequently, determines savings for reassigning customers to the drone. Ponza (2016) implements a simulated annealing approach to solve the FSTSP.

Agatz, Bouman, & Schmidt (2016) model the TSP with Drone, or TSP-D, a similar problem to the FSTSP using integer programming (IP) and several route-first cluster-second approaches based on local search and dynamic programming. Unlike the FSTSP, a customer node can be visited more than once in the TSP-D (Agatz et al., 2016). Additionally, the drone can return to the node where it was launched (Ha, Deville, Pham, & Hà, 2018). The results of Agatz et al. (2016) have shown significant savings: by a factor of time on average between 1.4 and 2. Ha et al. (2018) solve a variant of the TSP-D that seeks to minimize operational costs: transportation costs plus the time one vehicle must wait for the other. They address this problem with two heuristics: 1) a greedy heuristic and 2) a heuristic adapted from Murray & Chu (2015) that solves the TSP and locally searches for a TSP-D solution. Ferrandez, Harbison, Weber, Sturges, & Rich (2016) utilize *K*-Means clustering and a genetic algorithm to approach an in-tandem truck-drone delivery problem.

Murray & Chu (2015) explore a second problem in their paper called the parallel drone scheduling traveling salesman problem (PDSTSP). In this problem, upon dropping off a parcel, the drone has the option to go directly back to the depot for another parcel or to pick up a parcel at a customer location. In their problem, there is one truck, one depot, and *m* drones. They use a heuristic approach to solve this problem. Ham (2018) solves an *m*-truck, *m*-depot, and *m*-drone version of this problem using constraint programming.

In his thesis, Ponza (2016) provides a proposal for approaching the DTSP with ACO and Naïve approaches. The ACO approach proposed would include different types of pheromone, one for the truck route and the other for the drone route. Ponza (2016) remarks that "[ACO] is the second [behind simulated annealing] most interesting approach in need of analysis: it [has] never been tried before for the FSTSP or drone-related problems, has very promising characteristics, but it is slightly more difficult to approach as a metaheuristic than SA" (p. 21).

**Two-Pheromone Approaches**

Members of George Mason University's Evolutionary Computation Laboratory and the GMU Center for Social Complexity built a multi-agent simulator called MASON (George Mason University), for which Panait & Luke (2004a, 2004b, 2004c) modeled ant foraging behavior. Their simulations were unique in that they considered the use of two types of pheromone: one deposited from the ant home to the food source, and the second, from the food source to the ant home. They use an approach that mirrors dynamic programming. Ants use the maximum pheromone deposited to determine the direction in which the ant travels: towards the food source or home.

Ponza (2016) notes that to approach the DTSP, “the ACO approach would most likely…have two kinds of pheromone, one for truck and one for drone paths, and then follow the basic framework of the metaheuristic” (p. 21). The approach explored under study in this paper uses two pheromones in the sense that Ponza (2016) describes. The Ant System approach for the TSP uses only one type of ant (to construct the truck’s route). Analogously, this problem uses *ant pairs* since each construction of a tour requires a truck route and drone route, the latter of which consists of sorties. Each ant in the pair lays down a different type of pheromone when constructing its path, and each type of ant is attracted to its respective type of pheromone.

## Research Questions

This study seeks to answer the following questions:

1. What is the effect of using two pheromones, one for the truck route and the other for the drone route, on the algorithm’s results?
2. Are the rules that govern the truck ant’s and drone ant’s movements (i.e., the acceptance criteria) effective?

The second question is especially important because the literature appears to show that ACO-based techniques have not been implemented to solve the DTSP or related truck-drone problems before.

## Ant System Model and Explanation

Before explaining the ant colony optimization-based method used in this study to solve the DTSP, it is important to understand the mathematical formulation of ACO for the TSP. This study used Ant System (AS), the most basic ACO algorithm, as the basis for the DTSP algorithm developed. The equations presented are part of the mathematical formulation of AS.

### Acceptance Criteria

In the TSP, the parameters $\alpha_p$, $\beta$, and $\rho$ are established at the beginning of the algorithm as well as the number of ants, number of iterations, and number of stops along the route. The parameters $\alpha_p$ and $\beta$ are part of the probability equation that determines the likelihood of the ants visiting certain cities. In Equation 1, $p_{ij}^{k}$ gives the probability that ant $k$ will take the path from its current city $i$ to city $j$, where $j$ is in the neighborhood $N_i^k$. The neighborhood $N_i^k$ is the feasible set of cities that can be visited by ant $k$ from its current city $i$, that is, the remaining cities in the route of ant $k$.

Parameter $\alpha_p$ regulates the influence of pheromone $\tau_{ij}$ on the edge from city $i$ to city $j$, and $\beta$ regulates the influence of the visibility (or proximity) $\eta_{ij}$ of city $j$ to city $i$ to determine the desirability of choosing the next city. A higher value of pheromone $\tau_{ij}$, results in a greater probability of choosing the path from city $i$ to city $j$. The parameter $\eta_{ij}$ is equal to the inverse of the distance $d_{ij}$ as expressed by Equation 2.

$$p_{ij}^{k} = \frac{[\tau_{ij}]^{\alpha_p}[\eta_{ij}]^{\beta}}{\Sigma_{l \in N_i^k}[\tau_{il}]^{\alpha_p}[\eta_{il}]^{\beta}}, \text{if } j \in N_i^k \tag{1}$$

$$\eta_{ij} = \frac{1}{d_{ij}} \tag{2}$$

Thus, a greater distance between city $i$ and city $j$ gives a smaller value in $\eta_{ij}$, resulting in a lesser probability of choosing city $j$. Conversely, a shorter distance $d_{ij}$, indicates closer proximity $\eta_{ij}$, giving a greater probability of choosing city $j$.

A higher value of the exponent $\alpha_p$ gives greater magnitude to the value of $\tau_{ij}$, and a higher value of the exponent $\beta$ gives greater magnitude to the value of $\eta_{ij}$. If $\alpha_p$ were equal to zero, then the probability $p_{ij}^{k}$ would be purely greedy and consider only the proximity in determining the next city. If $\beta$ were equal to zero, then the probability would only consider the amount of pheromone along the trail. This tends to lead to poor results. It is important to set these parameters appropriately. Experimental results show that setting $\alpha_p$ equal to one and $\beta$ between two and five produce good performance when using Ant System (without local search) to solve the TSP (Dorigo & Stützle, 2004). Note that the $\alpha_p$ used in Equation 1 for affecting the influence of pheromone is distinguished from $\alpha_{ds}$, which is the ratio of the drone speed to the truck speed, discussed later in the paper. Optimizing the parameters $\alpha_p$ and $\beta$ are beyond the scope of this paper.

After these parameters have been initialized, the first iteration begins. At the beginning of an iteration, the ants are assigned to random start cities. During the iteration all ants construct their tours. Equation 1 is used to select the next city in the tour until all cities have been visited, at which point the truck and drone ants return to the starting depot.

## Calculating Change in Pheromone

If an ant $k$ takes an edge from city $i$ to city $j$, the pheromone level along that path will change. The change in pheromone $\Delta\tau_{ij}^{k}$ on an edge from city $i$ to city $j$ that an ant $k$ takes is a function of that ant's tour length. The length $C^k$ of the tour $T^k$, the tour of ant $k$, is used to determine the amount of pheromone to deposit on these edges; specifically, ants that create shorter tours, will leave greater amounts of pheromone on the edges that are elements of their tours, than ants that take longer routes. This is shown by Equation 3.

$$\Delta\tau_{ij}^{k} = \begin{cases} \frac{1}{C^k}, & \text{if arc } (i,j) \text{ belongs to } T^k \\ 0, & \text{otherwise} \end{cases} \tag{3}$$

## Pheromone Evaporation/Subtraction

After all ants have constructed their tours in an iteration, the amount of pheromone on each trail is updated with a two-phase process: pheromone evaporation and deposit. Equation 4 represents the pheromone left on all paths after evaporation:

$$\tau_{ij} \leftarrow (1-\rho)\tau_{ij,}\ \forall(i,j) \in L \tag{4}$$

where $0 < \rho \leq 1$ is the evaporation constant (Dorigo & Stützle, 2004).

The set $L$ consists of the edges connecting the nodes of the graph. Equation 4 causes the current amount of pheromone $\tau_{ij}$ on the paths to decrease. It has been found experimentally that setting $\rho$ to 0.5 results in good performance when using Ant System without local search to solve the TSP (Dorigo & Stützle, 2004).

## Pheromone Deposit/Addition

After pheromone has been evaporated from all paths in the network, the pheromone values for all paths are updated according to the amount of pheromone "deposited" by the ants in the current iteration. If $m$ is the number of ants used in the algorithm, then Equation 5 is used to deposit pheromone:

$$\tau_{ij} \leftarrow \tau_{ij} + \sum_{k=1}^{m} \Delta\tau_{ij}^{k}, \quad \forall (i,j) \in L \tag{5}$$

Equation 3 shows that $\Delta\tau_{ij}^{k}$, the change in the amount of pheromone on the path from city $i$ to city $j$, has an inverse relationship with the length of an ant $k$'s tour. Thus, a greater amount of pheromone will be deposited on the edges $(i,j)$ by ants that took shorter tours.

## Pseudocode

The pseudocode below describes one implementation of Ant System. The pseudocode for the DTSP algorithm builds off this structure.

```
Part  Procedure Ant System Metaheuristic
1     Initialize
2         Set number of cities, number of ants, number of iterations, $\alpha_p$, $\beta$, and $\rho$, distance matrix
          Distance$_{ij}$, $\kappa$, and $\alpha_{ds}$
3         Initialize all pheromone trails (values must be > 0)
4         minCost = infinity // Set minimum cost
5     Body
6     Initialize canChoose vector to contain all cities
      While iteration budget remains
7         Randomize ant start cities
          Set all $\Delta\tau_{ij} = 0$
8         For $k$ = 1 to number of ants
9             Route($k$, 1) = start city for $k$
              $i$ = Route($k$, 1)
              numCitiesVisited = 1
10            While numCitiesVisited $<$ number of cities
11                Calculate all $p_{ij}^k$ in $N_i^k$ using Equation 1
                  [maxProb, destIndex] = max($p_{ij}^k$)
                  probAcceptance = rand()
                  probCumulative = maxProb
                  probReached = false

                  While probReached == false
                      If probCumulative ≥ 1 – probAcceptance
                          probReached = true
                      Else
                          Remove city at destIndex from canChoose vector
                          [maxProb, destIndex] = max(remaining cities under consideration)
                          probCumulative = probCumulative + maxProb
                      End If
                  End While Loop

                  Visited(city at destIndex) = true
                  numCitiesVisited = numCitiesVisited + 1
```

```
            Route($k$, numCitiesVisited) = city at destIndex
            cost($k$) = cost($k$) + Distance$_{ij}$($i$, city at destIndex)
            Reset canChoose vector to contain any unvisited cities

            $i$ = city at destIndex // Make the destination city the new current city
12       End While Loop
         cost($k$) = cost($k$) + Distance$_{ij}$($i$, 1) // Return truck to depot

         For each path $(i, j)$ traversed by ant $k$
            $\Delta\tau_{ij}$ = $\Delta\tau_{ij}$ + 1/cost($k$)
         End For Loop
13    End For Loop

      min(cost array) < minCost // If the best solution this iteration is better than global minimum
         minCost = min(cost array)
         bestRoute = route of minimum length tour
      End If
      Update Pheromone Levels using Equations (4) and (5)
14 End While Loop
15 End Body
```

## Method

### Algorithm and Rationale

The algorithm implemented is based off the work of Rich & Ham (2018) and takes concepts of Ant System, applying them to the DTSP.

**Algorithm assumptions.** A few assumptions were made. First, the truck only delivers to one customer while the drone takes a sortie. Second, like Rich & Ham (2018), the drone must rendezvous with the truck at the last customer node in the route. Unlike the problem presented by Murray & Chu (2015), it cannot rendezvous with the truck at the depot. Third, the algorithm assumes the network used in the DTSP is symmetric like the TSP: the distance $d_{ij}$ from city node $i$ to $j$ is equal to the distance $d_{ji}$. Fourth, delivery is considered instantaneous. Finally, the program allows for the distance matrix to have entries of “0.” This was done in order to be able to test the algorithm on the 90-instance stop of the distance matrices used by Rich & Ham (2018) in their data set. To avoid division by zero, neighbors that were zero units away from the current city are automatically picked next.

**Cost function.** The cost of ant $k$’s tour is based on the cost function used in the evolutionary algorithm of Rich & Ham (2018). An operation consists of at least two nodes: a start node and an end node. If there are feasible sorties the drone could take to meet up with the truck at the selected end node, then the drone will be forced to travel to an intermediary node and the operation will consist of three nodes. For every operation, the maximum of the two values of (1) the truck distance traveled in the sortie and (2) the drone distance traveled in the sortie divided by the speed factor $\alpha_{ds}$ is added to the cumulative distance traveled so far. However, for an operation where the truck and drone move together, the distance traveled by the truck (moving together with the drone) is added onto the cumulative distance traveled so far. The cost of a tour taken by ant pair $k$ can be found using Equation 6 iteratively until all cities are visited by the truck or drone:

$$C^k \leftarrow C^k + \max(\mathrm{truckDistance}_o^k, \frac{\mathrm{droneDistance}_o^k}{\alpha_{ds}}) \quad (6)$$

where $\mathrm{truckDistance}_o^k$ is the truck’s distance during operation $o$ and $\mathrm{droneDistance}_o^k$ is the drone’s distance during operation $o$. The value of $\mathrm{droneDistance}_o^k$ is zero if the drone rides with the truck. The number of operations can vary between ant pairs and iterations. The best tour found has the minimum cost of all tours created by the ant pairs.

**Conformance to Ant System benchmarks.** The algorithm was designed to conform to the basic rules of Ant System used in the TSP with modifications for solving the DTSP. It was deemed appropriate to start the ant pairs at random starting stops in order to conform with the Ant System algorithm as much as possible and increase the chances of finding the best solution. This did not seem to be a problem, since the evolutionary algorithm of Rich & Ham (2018) created several strings of routes with different starting stops as well. Each of the paths of the network was initialized with a pheromone intensity equal to the number of ants divided by the nearest neighbor heuristic path length. This is in conformance with a good pheromone initialization heuristic in Ant System: that is, to set the pheromone value “slightly higher than the expected amount of pheromone deposited by the ants in one iteration” (Dorigo & Stützle, 2004, p. 70).

**Acceptance criteria.** To reduce the computation time for larger data sets, the probability equations used to route the drone only consider the closest 10 cities when determining how to route the truck and drone. The rationale for considering only the closest 10 was that the algorithm would converge on better solutions more quickly, and, thus, be more competitive with the current algorithms in the literature in terms of processing time. Equation 1 used in Ant System for the TSP is also used for choosing the next city for the truck in this approach; it has been reformulated, however, as Equation 7, with $\tau_{ij}^{t}$ representing the amount of truck pheromone along the path from *i* to *j*.

$$p_{ij}^{k} = \frac{\left[\tau_{ij}^{t}\right]^{\alpha_p}\left[\eta_{ij}\right]^{\beta}}{\Sigma_{l\epsilon \mathrm{N}_i^k}[\tau_{il}^{t}]^{\alpha_p}[\eta_{il}]^{\beta}}, \qquad \text{if } j \in N_i^k \tag{7}$$

If the drone can create a sortie from the start node to that end node that is within range, Equation 8 is used to determine where to route the drone from the start city $i$ to the midpoint city $l$ to the end city $j$.

$$p_{ilj}^{k} = \frac{\left[\tau_{il}^{d} + \tau_{lj}^{d}\right]^{\alpha_p}\left[d_{il} + d_{lj}\right]^{\beta}}{\Sigma_{q\epsilon N_i^k}\left[\tau_{iq}^{d} + \tau_{qj}^{d}\right]^{\alpha_p}\left[d_{iq} + d_{qj}\right]^{\beta}}, \qquad \text{if } l \in N_i^k \tag{8}$$

Equation 8 is designed to maximize the distance that the drone travels separately from the track; in other words, the equation helps increase the probability of choosing in-range neighbors that make for a longer sortie. However, moving the drone to the city furthest away from the truck's destination will not always be optimal. If there is no feasible sortie, the drone moves with the truck to its destination city.

**Finishing tour construction.** The chosen cities for the truck and drone are taken out of the pool of the remaining cities, and then the process of choosing the city the truck and drone will visit next repeats. After all cities have been visited by either the drone or by the truck, the next ant constructs its tour.

**Updating pheromone.** The change in pheromone $\Delta\tau_{ij}^{k}$ caused by ant pair $k$ to an edge $(i, j)$ is like Equation 3, except that the cost of a tour $C^k$ is now given by Equation 6 rather than by the length of the truck's tour in the TSP. After all ants have constructed their tours, the process of updating pheromone occurs. The evaporation equation for the truck in the DTSP algorithm is like Equation 4 used in the Ant System algorithm for the TSP. However, an evaporation equation must now also be applied to the drone pheromone. Equations 9 and 10 represent the evaporation of truck pheromone and drone pheromone, respectively, from all paths in the network:

$$\tau_{ij}^{t} \leftarrow (1-\rho)\tau_{ij}^{t}, \ \forall (i,j) \in L \tag{9}$$

$$\tau_{ij}^{d} \leftarrow (1-\rho)\tau_{ij}^{d}, \ \forall (i,j) \in L \tag{10}$$

where $\tau_{ij}^{t}$ and $\tau_{ij}^{d}$ are the amount of truck pheromone and drone pheromone on the path from city $i$ to city $j$, respectively. Recall that $L$ consists of all edges in the network.

The pheromone deposit equation for the truck is like that in the TSP, but pheromone is only applied to the paths the truck takes. An additional equation is needed to model deposit of drone pheromone on its paths. The pheromone deposit equations for the truck and drone are given by Equations 11 and 12, respectively:

$$\tau_{ij}^{t} \leftarrow \tau_{ij}^{t} + \sum_{k=1}^{m} \Delta\tau_{ij}^{k}, \quad \forall (i,j) \in L_{t}^{k} \tag{11}$$

where $L_{t}^{k}$ is the set that consists of the edges that are in the tour of the truck in ant pair $k$.

$$\tau_{ij}^{d} \leftarrow \tau_{ij}^{d} + \sum_{k=1}^{m} \Delta\tau_{ij}^{k}, \quad \forall (i,j) \,\epsilon\, L_d^k \tag{12}$$

where $L_d^k$ is the set that consists of the edges that are in the tour of the drone in ant pair $k$.

**Pseudocode**

The pseudocode for the DTSP algorithm is shown below. The MATLAB source code is also available in Appendix A. The nearest-neighbor heuristic used to initialize the amount of pheromone on all edges is given in Appendix B.

```
Procedure ACO-Inspired Probabilistic Greedy Approach
Initialize
    Set number of cities, number of ant pairs, number of iterations, $\alpha_p$, $\beta$, and $\rho$, distance matrix
    Distance_ij, $\kappa$, and $\alpha_{ds}$
    Initialize all pheromone trails for both truck and drone (values must be > 0)
    minCost = infinity // Set minimum cost
    Find the top 9 closest cities to each of the NUM_CITIES and store in array top10dMat
Body
While iteration budget remains
    Randomize ant pair start cities
    Set all $\Delta\tau_{ij}^{t} = 0$
    Set all $\Delta\tau_{ij}^{d} = 0$
    For $k$ = 1 to number of ants
        TruckRoute($k$, 1) = start city for $k$
        $i$ = TruckRoute($k$, 1)
        numCitiesVisited = 1
        tMove = 2 // Position in TruckRoute vector
        dMove = 0 // Position in DroneRoute vector, assume no sorties until we start building
        them
        While numCitiesVisited < number of cities
            If the number of cities remaining > 10 then
```

```
                Fill up currentTop10 array to have 10 unvisited cities
            Else
                Fill up currentTop10 array with remaining cities
            End if
            Choose truck destination city: Follow procedure found in Part 11 of the Pseudocode
            section in Ant System Model and Explanation

            // destStop contains the city the truck ant chose
            truckMoveDist = Distance_ij($i$, destStop)

            Calculate all $p_{ilj}^{k}$ using Equation 6 from the currentTop10 array (which can now only
            contain up to 9 cities, since the truck chose the first city of this operation)

            If there is at least one $l$ that creates a feasible sortie between $i$ and $j$
                [maxProb, midIndex] = mid-point $l$ that maximizes the distance from $i$ to $l$ to $j$
                probAcceptance = rand()
                probCumulative = maxProb
                probReached = false

                While probReached == false
                    If probCumulative ≥ 1 – probAcceptance
                        // Store the drone sortie
                        DroneRoute(dMove + 1) = $i$
                        DroneRoute(dMove + 2) = city at midIndex
                        DroneRoute(dMove + 3) = destStop
                        probReached = true
                        numCitiesVisited = numCitiesVisited + 1
                        Visited(city at midIndex) = true
                        droneMoveDist = Distance_ij($i$, city at midIndex) + Distance_ij(city at
                        midIndex, destStop)
                        dMove = dMove + 3
                    Else
                        Remove city at midIndex from consideration
                        [maxProb, midIndex] = max(remaining cities under consideration)
                        probCumulative = probCumulative + maxProb
                    End if
                End While Loop
            Else
                droneMoveDist = 0
            End If

            If truck and drone moved together then
                cost($k$) = cost($k$) + truckMoveDist
            Else If drone took a sortie then
                cost($k$) = cost($k$) + max(truckMoveDist, droneMoveDist/$\alpha_{ds}$)
            End If
            tMove = tMove + 1
            $i$ = destStop // Make the destination city the new current city
        End While Loop
        cost($k$) = cost($k$) + Distance$_{ij}$($i$, 1) // Return truck and drone to depot

        For each path ($i$, $j$) traversed by ant pair $k$
            $\Delta\tau_{ij}^{t}$ = $\Delta\tau_{ij}^{t}$+ 1/cost($k$)
            $\Delta\tau_{ij}^{d}$ = $\Delta\tau_{ij}^{d}$+ 1/cost($k$)
        End For Loop
    End For Loop
    If min(cost array) < minCost // If the best solution this iteration is better than global minimum
        Update the minCost, best truck route, and best drone route
    End If
    Update Truck and Drone Pheromone Levels using Equations 9, 10, 11, and 12
End While Loop
End Body
```

## Experiment Benchmarking

The experiments done for this study are based off the work of Rich & Ham (2018). The experiments use the same data set as Rich & Ham (2018). The data set consists of distance matrices for 10 to 100 stops. Each distance matrix includes an additional row and column for the depot starting location. The range limit $\kappa$ of the drone is 14 miles, and the speed ratio $\alpha_{ds}$ of the drone to the truck is two, which were the parameters used in the experiments done by Rich & Ham (2018). To mirror the trials done by Rich & Ham (2018), particularly those for the EA, three trials were run on each of the ten data sets.

Time measurements were taken to calculate the efficiency of the algorithm in speed and accuracy. Elapsed times for improved solutions were taken after constructing all the tours in an iteration. If the minimum cost found in an iteration was lower than the global minimum cost found up to that iteration, then the elapsed time was output to the screen.

The settings for the pheromone tuning parameter, $\alpha_p = 1$; proximity tuning parameter, $\beta = 3$; and the evaporation constant, $\rho = 0.5$, were all held constant. These parameter settings were experimentally found to result in good performance for Ant System in the TSP (Dorigo & Stützle, 2004). However, the author does not assume that these are necessarily the best values for the DTSP.

Table 1 shows the settings of parameters used at each level of the experiment. In general, greater numbers of ants and iterations were used for the trials that involved more stops. The rationale for this is that a greater number of ants and iterations increases exploration, which is especially important for many stops. Using an excessive number of ants and iterations for a smaller number of jobs can increase computation time unnecessarily.

The number of iterations was not always greater for a greater number of stops, however. The number of iterations and ants were chosen so that they produced a runtime close to those used for the evolutionary algorithm of Rich & Ham (2018). For example, the 80-stop instance was run for more iterations than the 90 and 100-stop instances because more iterations were needed to reach a time span of 120 seconds, which was the runtime given by Rich & Ham (2018) for all three of these test instances.

Table 1

*Parameters Used for Test Instances of the Proposed Algorithm*

| **Number of Stops** | **Number of Ants** | **Number of Iterations** |
|---|---|---|
| 10 | 100 | 110 |
| 20 | 150 | 160 |
| 30 | 275 | 175 |
| 40 | 300 | 225 |
| 50 | 300 | 275 |
| 60 | 300 | 305 |
| 70 | 300 | 390 |
| 80 | 300 | 460 |
| 90 | 300 | 410 |
| 100 | 300 | 360 |

## Results

Table 2 shows the results of the trials run on the data sets of Rich & Ham (2018). The iteration during which the best solution was found is denoted by the *Iter* column for each trial. The approximate time when it was found is recorded in the column marked *Elapsed (s)*. This time was taken after all tours had been constructed for an iteration, and when it was determined that the best solution so far was found in that iteration. Elapsed time was calculated in MATLAB to a precision of ± 0.000001 seconds, but the results were reported in Table 2 to the nearest hundredth place. In the last column, *Error* refers to the percent variation across the three trials run at each level.

Table 2

*Results for three trials of each test instance (job) level*

| | *Trial 1* | | | *Trial 2* | | | *Trial 3* | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Jobs | *f* | Iter | Elapsed (s) | *f* | Iter | Elapsed (s) | *f* | Iter | Elapsed (s) | Error |
| 10 | 228 | 4 | 0.04 | 229 | 9 | 0.08 | 229 | 1 | 0.01 | 0.4% |
| 20 | 287 | 140 | 4.11 | 288 | 116 | 3.44 | 290 | 49 | 1.46 | 1.1% |
| 30 | 373 | 28 | 2.48 | 373 | 58 | 4.91 | 375 | 113 | 9.50 | 0.5% |
| 40 | 433 | 110 | 14.49 | 431 | 124 | 16.53 | 433 | 36 | 4.83 | 0.5% |
| 50 | 487 | 91 | 14.79 | 486 | 124 | 20.68 | 486 | 8 | 1.40 | 0.2% |
| 60 | 429.5 | 293 | 57.72 | 430.5 | 184 | 35.49 | 430 | 121 | 23.87 | 0.2% |
| 70 | 516.5 | 89 | 20.45 | 529 | 316 | 72.10 | 524.5 | 9 | 2.18 | 2.4% |
| 80 | 534 | 21 | 5.68 | 542 | 26 | 7.08 | 544 | 114 | 30.12 | 1.9% |
| 90 | 585.5 | 370 | 107.77 | 591 | 378 | 110.86 | 586 | 307 | 90.34 | 0.9% |
| 100 | 620 | 207 | 68.73 | 621.5 | 105 | 35.58 | 628 | 76 | 25.51 | 1.3% |

Table 3 shows the results of the ACO-based algorithm in comparison to the MIP, CP, and EA models implemented by Rich & Ham (2018). In this table, the values given for *f* were the minimum values obtained from several trials. The code was implemented in MATLAB, and trials were run on a personal computer with an Intel Core i7-8750H CPU @ 2.20 GHz processor and 16 GB of RAM. For their experiments, Rich & Ham (2018) used a personal computer with an Intel Core i5-3537 @ 2.5 GHz processor and 8 GB of RAM. The *Gap* column in the table shows the percentage difference obtained from the best solution of the probabilistic, greedy approach compared to the best solution of each approach used by Rich & Ham (2018).

Results for the MIP and CP models report the time at which the best solution was reached. In contrast, the EA results report the total time given to find the best solution. Tests for the ACO-based approach were run for a total time close to the runtimes used for the corresponding EA instances (shown in Table 3).

Table 3

*Results of four DTSP optimization techniques*

| | MIP | | | CP | | | EA | | | ACO-Based | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Jobs | Time (s) | *f* | Gap (%) | Time (s) | *f* | Gap (%) | Time (s) | *f* | Gap (%) | Time (s) | *f* | Gap (%) |
| 10 | 1 | 228.0 | 0.0 | 1 | 228.0 | 0.0 | 1 | 228.0 | 0.0 | 0.04 | **228.0** | 0.0 |
| 20 | 54 | 285.5 | 0.0 | 20 | 285.5 | 0.0 | 5 | 285.5 | 0.0 | 4.11 | 287.0 | 0.5 |
| 30 | 1834 | 400.0 | 8.4 | 10 | 369.0 | 0.0 | 15 | 374.0 | 1.4 | 2.48 | **373.0** | 1.1 |
| 40 | 1692 | 621.0 | 44.6 | 55 | 429.5 | 0.0 | 30 | 434.0 | 1.0 | 16.53 | **431.0** | 0.3 |
| 50 | | – | | 118 | 466.0 | 0.0 | 45 | 478.5 | 2.7 | 1.40 | 486.0 | 4.3 |
| 60 | | – | | 422 | 415.5 | 0.0 | 60 | 419.5 | 1.0 | 57.72 | 429.5 | 3.4 |
| 70 | | – | | 1398 | 511.5 | 1.5 | 90 | 504.0 | 0.0 | 20.45 | 516.5 | 2.5 |
| 80 | | – | | 1745 | 523.0 | 0.0 | 120 | 546.5 | 4.5 | 5.68 | **534.0** | 2.1 |
| 90 | | – | | | – | | 120 | 638.0 | 9.0 | 107.77 | **585.5** | 0.0 |
| 100 | | – | | | – | | 120 | 655.0 | 5.6 | 68.73 | **620.0** | 0.0 |

*Note*. Adapted from "The truck-drone scheduling problem with a theoretical insight into system configuration," by R. Rich and A. Ham, 2018, p. 10.

## Discussion

It should be noted that the number of ants, number of iterations, and the subjective changes in computer processing time may have affected the time it took to execute the source code from trial to trial. However, the order of trials was performed randomly, and as was shown in Table 2, consistently low variation (< 2.5%) between the three trials at each level suggests consistency in the results. Table 3 shows that results obtained from the new algorithm are comparable to those reached in the MIP, CP, and EA. Specifically, the bolded values in Table 3 indicate any values that were equal or better than at least one of the other three techniques.

The first research question addressed in this paper stated, *What is the effect of using two pheromones, one for the truck route and the other for the drone route, on the algorithm's results?* This probabilistic greedy approach based on Ant System produced comparable results to the MIP, CP, and EA. The greatest percent error (gap) reached by the ACO-based approach across all levels was 4.3% at 50 stops. This is lower than the gap obtained by the EA at 70 stops, 4.5%. The algorithm also produced results better than the EA at 80, 90, and 100 stops. By inspection, it appears that the use of two pheromones in this ACO-based algorithm is the best of the four approaches for handling solution sizes of at least 90 stops.

However, the use of two pheromones works well up to a point. For most trials, the algorithm converged quickly, finding the best solution in early iterations of the run. In most cases, adding extra time did not result in significant improvement per unit of time added. Towards the beginning of the algorithm, the intensity of the pheromone on the path is significantly lower compared to the amount of pheromone accumulated by the last iteration. Figure 3 shows a typical convergence plot output by the program. The solution converges quickly for the first 27 iterations, but then the solution does not improve until iteration 159. The long horizontal line before the improvement at iteration 159 may indicate that the current parameter settings at that point were ineffective. That is, the algorithm may have performed better at this point had the parameters been optimized. Recall that $\alpha_p$, $\beta$, $\rho$, and the number of ants were held constant in these experiments.

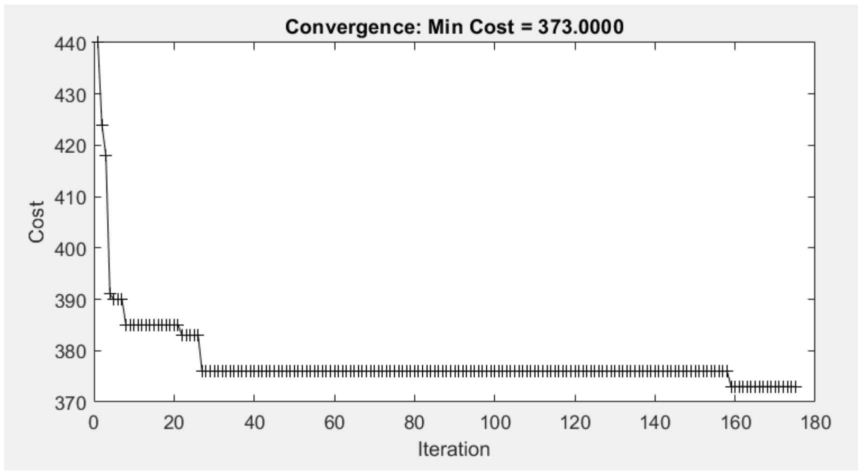

*Figure 3*. Convergence Plot for Instance of DTSP Algorithm.

One general pattern that was observed was that the pheromone intensity on underutilized paths approached zero as iterations increased. In general, better overall solutions were found if found in early iterations. This could discourage ants from following paths that may have led to a better solution and instead redundantly choose paths with the most pheromone. The influence of the pheromone could have been decreased or the number of ants constructing paths could have been decreased to prevent pheromone from becoming too great.

Figure 4 shows the best routing found in a 20-stop instance (including the depot), and Figure 5 is the corresponding convergence plot. In Figure 6, red and green lines show the relative amount of truck and pheromone on each path, respectively. The order in which the truck completed its route is as follows: 1, 14, 19, 9, 13, 17, 6, 20, 18, 3, 12, 15, 1. The path with the most truck pheromone, from stop 9 to stop 13, was part of the best solution found. However, the path from stop 1 to stop 12, which has the second greatest amount of pheromone, was not part of the best solution found. Part of the reason why this occurred is related to acceptance criteria, which leads into the next research question: *Are* *the rules that govern the truck ant's and drone ant's movements (i.e., the acceptance criteria) effective?*

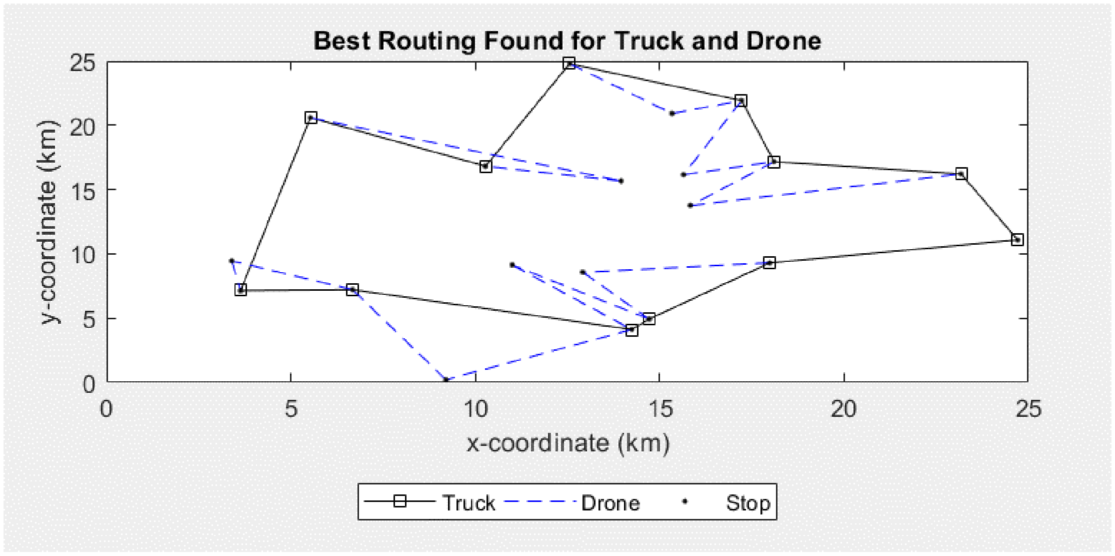

*Figure 4*. Plot showing the best truck and drone routing found for 20-stop instance (including depot) of the DTSP algorithm.

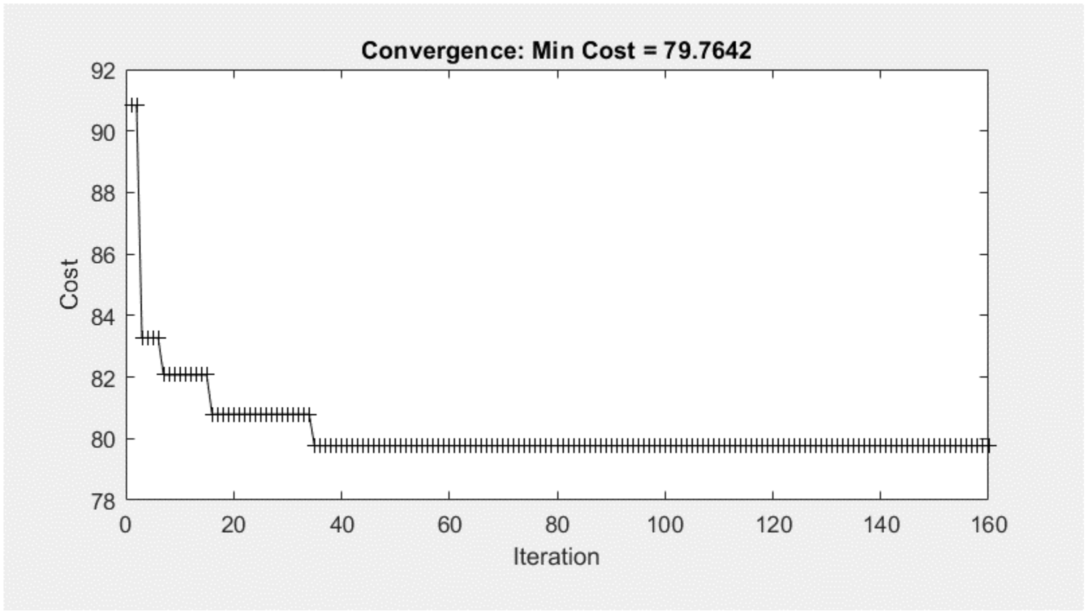

*Figure 5*. Convergence plot for 20-stop (including depot) instance of DTSP algorithm.

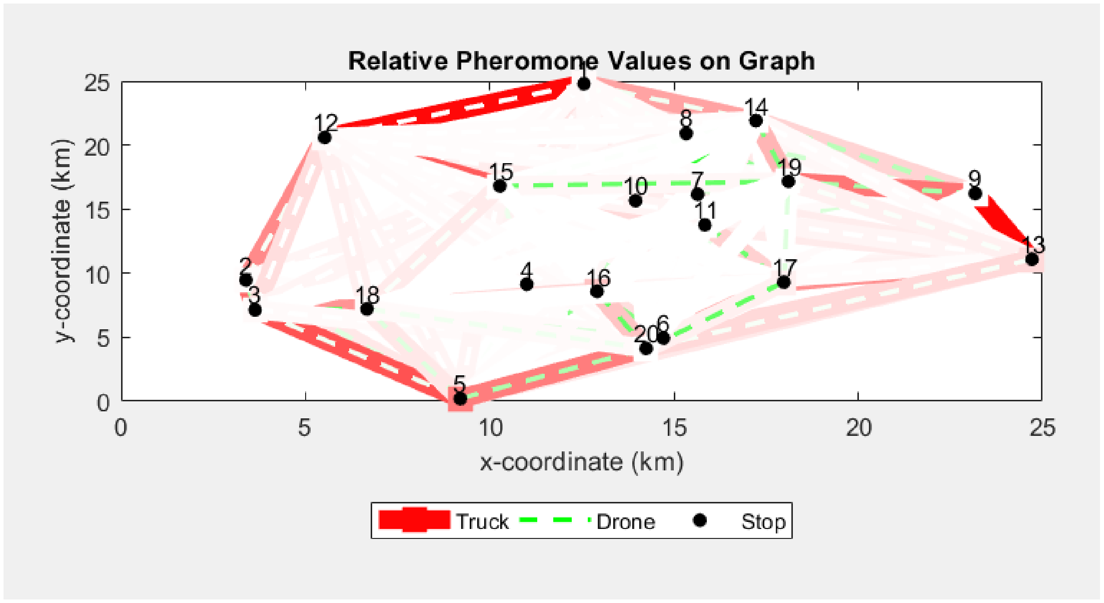

*Figure 6*. Plot showing the relative levels of truck and drone pheromone on the paths between all cities in 20-stop (including depot) instance of the DTSP algorithm.

The acceptance criteria for the truck in this ACO-based approach was the same as that for the truck in the classical TSP: amount of pheromone on a path $(i, j)$ and proximity $\frac{1}{d_{ij}}$ to city $j$ increased the probability of selecting a city. The drone acceptance criteria probability $p_{ilj}^{k}$most likely affected the quality of results more than the truck's probability $p_{ij}^{k}$. Equation 7 was designed to maximize the utilization of the drone by increasing the probability for selecting a middle sortie node that would cause the sortie to be longer. This could have negatively affected the way pheromone was laid down by the drone. For example, in Figure 6, there is more drone pheromone laid down on the sortie created by edges (14, 10) and (10, 19) than on the sortie created by (14, 7) and (7, 19), even though the latter edges form the sortie (i.e., <14, 7, 19>) that is part of the best solution found. The drone may lay down too much pheromone on inferior edges.

Another case where the acceptance criteria may have suffered was due to the interaction between the choice of the truck's next city and the drone's next city. The truck was designed to visit the closest city and the drone was designed to lengthen the sortie as much as possible. The next city for the truck was always chosen first. This may have led to the drone traveling inefficiently: moving in one direction for a great distance only to rendezvous at a node that was in an opposite direction (greater than 90° from its initial direction). The acceptance criteria would likely have been more effective if the probabilities for choosing the next cities for the truck and drone were designed to minimize wait time of the truck and drone rather than maximize drone utilization.

## Future Work

The ACO-based algorithm presented has contributed a new perspective to truck and drone routing problems in the literature, as no other drone-related problems have used ACO. The results show that ACO techniques have promise, especially for addressing data sets with a greater number of cities, since it had superior performance at higher levels to the EA presented by Rich & Ham (2018).

Modifying the acceptance criteria is one direction in which this could be explored further. Currently, the probability $p_{ilj}^{k}$ of a drone taking sortie $< i, l, j >$ is designed to maximize the utilization of the drone. It would be worthwhile to modify $p_{ilj}^{k}$ to be set up to minimize the waiting time of the truck and the waiting time of the drone. In fact, Ha et al. (2018) consider the cost of waiting in their variant of the TSP-D. In this case, the probability minimizes the gap between (1) the distance traveled by the truck in an operation and (2) the distance traveled by the drone divided by its speed factor in that operation. Results may improve if this policy is adopted.

Tuning the parameters such as $\alpha_p$, $\beta$, and $\rho$ is another important area of research. The recommended settings for Ant System in the context of the TSP were adopted for the DTSP, but it would be beneficial to optimize the parameters specifically for the DTSP. Another consideration related to tuning is that the drone probability equation for $p_{ilj}^{k}$ may have different values for $\alpha_p$ and $\beta$ than the acceptance equation for the truck $p_{ij}^{k}$. These could be optimized using a full factorial or fractional factorial experimental design approach. Doing this should help address the issues that arise when the solution converges too quickly and levels out for the remaining iterations.

Another area of exploration is performing the ant algorithm in a broader neighborhood than the 10 closest unvisited, feasible cities. This may increase the quality of results since the neighborhood is expanded. Another way that a greater solution space can be achieved is by changing the problem: it would be worthwhile to consider the possibility of having the truck visit more than one city while separated as Murray & Chu (2015) assume.

Anti-pheromone, which has been used in three Ant Colony System variants, might also be useful in solving this problem (Montgomery & Randall, 2002). Once the solution has converged and the convergence graph has leveled out for several iterations with pheromone built up to a high degree on several paths, anti-pheromone can be applied to the worst paths in the iterations to discourage taking paths that contribute to suboptimal solutions.

Another area of research would be to perform a comparative analysis between the author's approach and other heuristic approaches. A paired *t*-test could be conducted on larger test instances (of at least 80 stops) to provide statistical evidence as to whether the approach performs better on larger data sets than the EA of Rich & Ham (2018).

Finally, the basic framework could be extended by exploring other ant colony optimization techniques used to solve the DTSP. Other techniques applied to solve the classical TSP such as Elite Ant System, Ant Colony System, and MAX-MIN Ant System could potentially be used to solve the DTSP. They may perform better since they are superior to AS in the TSP. Since the literature has shown that local search techniques coupled with ACO have improved the performance of ACO techniques such as MAX-MIN Ant System in the TSP, local search techniques may also be applied to this problem to see whether they improve the quality of the solution.

## References

Agatz, N., Bouman, P., & Schmidt, M. (2015). Optimization approaches for the traveling salesman problem with drone. *SSRN Electronic Journal*. doi:10.2139/ssrn.2639672

Amazon (n.d.). *First Prime Air Delivery* [Video file]. Retrieved from https://www.amazon.com/Amazon-Prime-Air/b?ie=UTF8&node=8037720011

Barrabi, T. (2018, December 03). Amazon drone delivery? Jeff Bezos' timeline is past due. Retrieved from https://www.foxbusiness.com/business-leaders/amazon-drone-delivery-jeff-bezos-timeline-is-past-due

Brezina, I., Jr., & Čičková, Z. (2011). Solving the travelling salesman problem using the ant colony optimization. *Management Information Systems, 6*(4), 10-14. Retrieved from http://www.ef.uns.ac.rs/mis/

Calik, S. K., Kugu, E., Birtane, S., & Sahingoz, O. K. (2016). A multi agent solution for UAV path planning problem with NetLogo. *International Journal of Applied Engineering Research, 11*(15), 8397-8401.

Carlson, N. (2013, December 02). The real reason Amazon announced delivery drones last night: $3 million in free advertising on Cyber Monday. Retrieved from http://www.businessinsider.com/why-amazon-announced-delivery-drones-2013-12

Dorigo, M., & Gambardella, L. M. (1997). Ant colony system: A cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation, 1,* 53-66. doi:10.1109/4235.585892

Dorigo, M., & Stützle, T. (2004). *Ant colony optimization*. Cambridge, MA: MIT.

DTSP_Test_Instances. (2018). *The truck-drone scheduling problem with a theoretical insight into system configuration* [Zip file and data files]. Retrieved from https://drive.google.com/file/d/19ZZ9ukEwSSfjCqNID1P1kPsGMOyMefXC/view

Ellabib, I., Calamai, P., & Basir, O. (2007). Exchange strategies for multiple Ant Colony System. *Information Sciences, 177*, 1248-1264. doi:10.1016/j.ins.2006.09.016

Ferrandez, S. M., Harbison, T., Weber, T., Sturges, R., & Rich, R. (2016). Optimization of a truck-drone in tandem delivery network using k-means and genetic algorithm. *Journal of Industrial Engineering and Management*, *9,* 374-388. doi:10.3926/jiem.1929

George Mason University. (n.d.). *MASON Multiagent Simulation Toolkit*. Retrieved from https://cs.gmu.edu/~eclab/projects/mason/

Guilmartin, J. F., & Taylor, J. W. (2018). Military aircraft. In *Encyclopedia Britannica*. Retrieved February 3, 2019, from https://www.britannica.com/technology/military-aircraft/Unmanned-aerial-vehicles-UAVs

Ha, Q. M., Deville, Y., Pham, Q. D., & Hà, M. H. (2018). On the min-cost Traveling Salesman Problem with Drone. *Transportation Research Part C: Emerging Technologies, 86*, 597-621. doi:10.1016/j.trc.2017.11.015

Ham, A. M. (2018). Integrated scheduling of *m*-truck, *m*-drone, and *m*-depot constrained by time-window, drop-pickup, and m-visit using constraint programming. *Transportation Research Part C: Emerging Technologies*, *91*, 1-14. doi:10.1016/j.trc.2018.03.025

Laftello. “Other drone.” *Openclipart*. Retrieved from openclipart.org/detail/305390/other-drone.

Luke, S., Balan, G. C., Sullivan, K., Panait, L., Cioffi-Revilla, C., Paus, S., Kuebrich, D., Harrison, J., & Desai, A. (n.d.). MASON Multiagent Simulation Toolkit [Computer software]. Retrieved from https://cs.gmu.edu/~eclab/projects/mason/

Malakar, G. (Producer). (2017, October 15). *Tutorial - Introduction to ant colony optimization algorithm n* [sic] *how it is applied on TSP* [Video file]. Retrieved from https://www.youtube.com/watch?v=wfD5xlEcmuQ

Marzolla, M., & Babaoglu, O. (2014). *Agent-based systems* [PowerPoint presentation].

Montgomery, J., & Randall, M. (2002). Anti-pheromone as a tool for better exploration of search space. *Ant Algorithms Lecture Notes in Computer Science, 2463*, 100-110. doi:10.1007/3-540-45724-0_9

Murray, C. C., & Chu, A. G. (2015). The flying sidekick traveling salesman problem: Optimization of drone-assisted parcel delivery. *Transportation Research Part C: Emerging Technologies*, *54*, 86-109. doi:10.1016/j.trc.2015.03.005

Panait, L., & Luke, S. (2004a). A pheromone-based utility model for collaborative foraging. *Proceedings of 2004 Conference on Autonomous Agents and Multiagent Systems*.

Panait, L., & Luke, S. (2004b). Ant foraging revisited. *Proceedings of the Ninth*

*International Conference on the Simulation and Synthesis of Living Systems (ALIFE9).*

Panait, L., & Luke, S. (2004c). Learning foraging behaviors. *Proceedings of the Ninth International Conference on the Simulation and Synthesis of Living Systems (ALIFE9).*

Pierini, D. (2015, February 5). Alibaba one-ups Amazon with drone delivery of tea in China. Retrieved from https://www.cultofmac.com/311193/alibaba-one-ups-amazon-drone-delivered-tea/

Ponza, A. (2016). *Optimization of Drone-Assisted Parcel Delivery* (Unpublished master's thesis). Università Degli Studi Di Padova.

Popper, B. (2013, December 03). UPS researching delivery drones that could compete with Amazon's Prime Air. Retrieved from https://www.theverge.com/2013/12/3/5169878/ups-is-researching-its-own-delivery-drones-to-compete-with-amazons

Resnick, M. (2002). *Turtles, termites, and traffic jams: Explorations in massively parallel microworlds*. Cambridge, MA: MIT Press.

Rich, R. K. (2017). dtsp_ga_basic(nStops, popSize, numIter, xy, range, speed ) [Computer software]. Retrieved from https://www.mathworks.com/matlabcentral/fileexchange/60640-dtsp_ga_basic-nstops-popsize-numiter-xy-range-speed?s_tid=prof_contriblnk

Rich, R. K. (2017). tsp_ga_basic(nStops, popSize, numIter, xy ) [Computer software]. Retrieved from https://www.mathworks.com/matlabcentral/fileexchange/60640-dtsp_ga_basic-nstops-popsize-numiter-xy-range-speed?s_tid=prof_contriblnk

Rich, R. K., & Ham, A. M. (2018). *The truck-drone scheduling problem with a theoretical insight into system configuration*. Manuscript submitted for publication.

Selvi, V., & Umarani, D. (2010). Comparative analysis of ant colony and particle swarm optimization techniques. *International Journal of Computer Applications, 5,* 1-6. doi:10.5120/908-1286

Stützle, T., & Hoos, H. (1999). Max-Min Ant System. *Elsevier Science,* 1-39. doi:10.1007/978-3-7091-6492-1_54

Vaughan, R. T., Støy, K., Sukhatme, G. S., & Matarić, M. J. (2000). Whistling in the dark. *Proceedings of the Fourth International Conference on Autonomous Agents – AGENTS 00*. doi:10.1145/336595.337351

Wodrich, M., & Bilchev, G. (1997). Cooperative distributed search: The ants' way. *Controls and Cybernetics, 26*(3), 413-445.

Yan, X., Zhang, C., Luo, W., Li, W., Chen, W., & Liu, H. (2012). Solve traveling salesman problem using particle swarm optimization algorithm. *International Journal of Computer Science Issues, 9*(6), 264-271.

Yang, J., Shi, X., Marchese, M., & Liang, Y. (2008). An ant colony optimization method for generalized TSP problem. *Progress in Natural Science, 18,* 1417-1422. doi:10.1016/j.pnsc.2008.03.028

## Appendix A

```
function [ bestMinCost ] = DTSP_ACOTop10(nStops, nAntPairs, numIter, ALPHA,
BETA, RHO, dMat, DRONE_DIST_LIMIT, DRONE_SPEED)
    % Notes about inputs:
    % -- nStops INCLUDES the depot.
    % -- dMat should be formatted as a symmetrical matrix. The matrix may
    % have 0s in spots NOT along the diagonal as well as on the
    % diagonal of the matrix.

    % This algorithm forces the drone and truck to be together by the last
    % city visited. It allows for the truck to wait for the drone AND
    % vice-versa.
    tic % Start timing the algorithm


%************************************************************************
    %    Title: dtsp_ga_basic(nStop?s, popSize, numIter, xy, range, speed )
    %    Author: Robert Rich
    %    Date: October 12, 2017
    %    Code version: 1.4.0.0
    %    Availability:
https://www.mathworks.com/matlabcentral/fileexchange/60640-dtsp_ga_basic-
nstops-popsize-numiter-xy-range-speed?s_tid=prof_contriblnk
    %

%************************************************************************
    if nargin < 9  % If the user provided less than 9 input variables, use
      defaults.
      nStops=20;  nAntPairs=150; numIter=160; ALPHA = 1; BETA = 3; RHO = 0.5;
          DRONE_DIST_LIMIT = 14; DRONE_SPEED = 2;
      xy = 25*rand([nStops, 2]);
      numColors = 100;
        nPoints = size(xy,1);
        meshg = meshgrid(1:nPoints);
        dMat = reshape(sqrt(sum((xy(meshg,:)-
xy(meshg',:)).^2,2)),nPoints,nPoints);

        % Preparations for making the pheromone graph
        scale = [1 1 1];
        scale2 = scale;
        factor = 1 / numColors;
        controller = 1;
        for i = 2:numColors
            scale = cat(1, scale,[1 controller controller]);
            scale2 = cat(1, scale2,[controller 1 controller]);
            controller = controller - factor;
        end
    end

    % The algorithm
    NU = zeros(nStops,nStops);  % This contains the inverses of the distance
matrix values.

    for i = 1:nStops
        for j = 1:nStops
            if dMat(i,j) ~= 0
                NU(i,j) = 1 / dMat(i,j);
            elseif i == j % If it is along the diagonal
                NU(i,j) = inf;
                dMat(i,j) = inf;
             % The only way neither of these if statements are executed is
             % if a non-diagonal cell value == 0.
             % The non-diagonal 0s should not be changed to inf.
            end
        end
    end

    % Find the top 10 closest stops to each of the nStops.
    % Output from for loop: in every row (i) in the closestCities array, there
      will be the
    % top 10 closest cities for each respective city i
    tempdMat = dMat;

    if nStops > 10
        closestCities = zeros(nStops, 9);
        for j=1:9
          [~, cityIdx] = min(tempdMat,[],2);
          closestCities(:,j) = cityIdx;
          % Remove for the next iteration the last smallest value:
          for i = 1:nStops
            tempdMat(i,cityIdx(i)) = inf;
          end
        end
    end

    % Find length of the greedy heuristic path so we know how to initialize
      pheromone
    % level on paths in the network (must start at value > 0).
    Cnn = NearestNeighbor(dMat, nStops);

    % Initialize pheromone along paths
    TauT = zeros(nStops,nStops);
    TauD = zeros(nStops,nStops);
    for i = 1:nStops
        for j = 1:nStops
            if i ~= j
                TauT(i,j) = nAntPairs ./ Cnn; % Contains truck pheromone on
                    path from city i to city j
                TauD(i,j) = nAntPairs ./ Cnn; % Contains drone pheromone on
                    path from city i to city j
            end
        end
    end
```

```
% Initialize Best Solution Variables
bestMinCost = inf; % This contains the global minimum cost.
bestTruckRoute = 0; % Vector containing order truck visits cities
bestDroneRoute = 0; % Vector containing order drone visits cities
distHistory = zeros(numIter,1); % Contains best solutions so far (found
    during or before the current iteration)
droneMoveDist = 0;

% Construct tours for k ant pairs, numIter times
for iter = 1:numIter
    % Initialize start stops for each ant pair (random)
    startStops = zeros(nAntPairs,1);
    RouteT = zeros(nAntPairs,1);
    RouteD = zeros(nAntPairs,1);
    for k = 1:nAntPairs
        startStops(k) = randi(nStops);
        RouteT(k, 1) = startStops(k);
    end

    % These variables are necessary for determining by how much the
    % pheromone on the paths change at the end of an iteration.
    sumOfChangeInTauT = zeros(nStops, nStops);
    sumOfChangeInTauD = zeros(nStops, nStops);

    % The route costs for antPairs
    cost = zeros(nAntPairs,1);

    % Initialize moves for all ants
    moveT = zeros(nAntPairs);
    moveD = zeros(nAntPairs);

    % Find feasible route for ant pair k
    for k = 1:nAntPairs
        % -------------------------------------------------------------
        % Preliminary Initialization for ant pair k
        % -------------------------------------------------------------

        % Initialize the remainingStops to all stops except for the
        % first (ant pair is already at the first)
        remainingStops = 1:nStops;
        remainingStops(startStops(k)) = [];

        % Initialize the current move the truck and drone are on
        moveT(k) = 2;
        moveD(k) = 0; % no sorties until we start building them

        % Set the current stop equal to the starting stop
        currentStop = startStops(k);
        currentTop10 = zeros(1,9);

        % Determine (based on the number of cities) whether there is
        % need to keep track of any cities that have not yet been added
        % to the top 10 (this would only happen if nStops > 10)
        % Any cities that will not be in the top 10 initially need to be
          rotated in
        % (and will be tracked using the dMatLookUp variable)
        if nStops > 10
            dMatLookUp = dMat; % This helps us keep track of the cities
                that are not already part of our top 10 (which ones that
                we will need to eventually add to our currentTop10 to be
                visited)
            % Copy over the 10 closest cities we calculated earlier
            for t = 1:length(closestCities(currentStop,:))
                currentTop10(t) = closestCities(currentStop,t);
            end
        else % There are no more than 10 cities so there is no need to
               rotate in unvisited stops (all of the stops will be in the
               currentTop10 immediately)
            currentTop10 = remainingStops;
        end

        % Exclude the cities that are already in the top 10, to prevent
          possibility of them being added again
        if nStops > 10
           dMatLookUp(1:nStops, currentStop) = inf;
           for i = 1:nStops
                for c = 1:length(currentTop10)
                    dMatLookUp(i, currentTop10(c)) = inf;
                end
            end
        end

        % ---------------------------------------------------------------
        % Body of algorithm
        % ---------------------------------------------------------------

        while ~isempty(remainingStops) % While not all cities have been
            visited
            % Build the route
            if length(remainingStops) >= 10 && nStops > 10 % This helps us
                ensure there are at least 10 cities under consideration in
                Pijk

                while length(currentTop10) < 10
                    [~, cityIdx] = min(dMatLookUp(currentStop,:));
                    currentTop10(length(currentTop10) + 1) = cityIdx;
                    % keep the value from being rotated in again
                    dMatLookUp(1:nStops,cityIdx) = inf;
                end

            elseif nStops > 10

                while length(currentTop10) ~= length(remainingStops)
                    [~, cityIdx] = min(dMatLookUp(currentStop,:));
                    currentTop10(length(currentTop10) + 1) = cityIdx;
                    % keep the value from being rotated in again
                    dMatLookUp(currentStop,cityIdx) = inf;
                end

            end

            % If there is a stop 0 units away from the current stop, move
            % to it next
            if min(dMat(currentStop,currentTop10)) == 0
                [~, cityIdx] = min(dMat(currentStop,currentTop10));
                RouteT(k,moveT(k)) = currentTop10(cityIdx);
                remainingStops(remainingStops==currentTop10(cityIdx)) =
                    [];
                currentTop10(cityIdx) = [];
                % move on to choose next city
                currentStop = RouteT(k,moveT(k));
                moveT(k) = moveT(k) + 1;
            else

                % Generate all possible probabilities
                % First, calculate the denominator of the probabilities
                sumOfTauNu = 0.0;
                for j = 1:length(currentTop10)
                    sumOfTauNu = sumOfTauNu +
                       (TauT(currentStop,currentTop10(j)) ^ ALPHA *
                       NU(currentStop,currentTop10(j)) ^ BETA);
                end

                % Calculate the individual probability for ant k to move
                % from city i to city j
                Pijk = zeros(1,length(currentTop10)); % Vector containing
                    probabilities

                    for j = 1:length(currentTop10)
                        Pijk(j) = ((TauT(currentStop, currentTop10(j))) ^
                            ALPHA * (NU(currentStop, currentTop10(j))) ^
                            BETA) / sumOfTauNu;
                    end

                % Determine the most likely city
                maxProb = max(Pijk);
                maxStopIndex = find(Pijk==maxProb); % Get the index where
                    the max probability was found
                while length(maxStopIndex) > 1 % In the chance that two
                    cities had equal probability, choose 1
                    maxStopIndex(randi([1 length(maxStopIndex)])) = [];
                end

                % Accept one of the cities as the next city
                consideration = currentTop10; % These are the city numbers
                    under consideration.
                probAcceptance = rand(); % Probability that the city
                   corresponding to the cumulative probability will be
                   accepted
```

```
                    probCumulative = maxProb; % Initialized to the most likely
                        city; this probCumulative is added on to when the most
                        likely city is not accepted
                    probReached = false; % Indicates whether probCumulative is
                        high enough to accept the last considered city.

                    % Assume the first city is not accepted until shown
                        otherwise.
                    while probReached == false

                        if (probCumulative >= 1 - probAcceptance) % If the
                            probability of acceptance is high enough
                            RouteT(k, moveT(k)) = consideration(maxStopIndex);
                            % Add the max stop to the route.

                            % Remove the stop just added from the
                            % remainingStops and the currentTop10, and exit
                              the
                            % loop.

remainingStops(remainingStops==consideration(maxStopIndex)) = [];

currentTop10(currentTop10==consideration(maxStopIndex)) = [];
                            probReached = true;
                        else
                            % Remove this stop from consideration this move
                            Pijk(maxStopIndex) = [];
                            consideration(maxStopIndex) = [];
                            maxProb = max(Pijk);
                            maxStopIndex = find(Pijk==maxProb);
                            while length(maxStopIndex) > 1
                                maxStopIndex(randi([1 length(maxStopIndex)]))
                                     = [];
                            end
                            probCumulative = probCumulative + maxProb;
                        end

                    end

                    % Calculate move distance for the truck
                    destStop = RouteT(k, moveT(k));
                    truckMoveDist = dMat(currentStop, RouteT(k,moveT(k)));
                    % Move the drone if possible
                        % If after the truck is moved to the destStop (i.e.,
                          potential "rendezvous node"), there is at
                        % least one location unvisited, then see if it can
                        % be serviced by the drone.
                    if length(currentTop10) >= 1
                        sumOfTauD = 0.0; % Calculate the denominator we will
                            use to determine the probabilities of choosing
                            different
                                         % cities as the middle node in a
                                         % sortie.
                        % See if it is possible to move the drone in a
                        % sortie configuration.
                        for l = 1:length(currentTop10)
                            if dMat(currentStop, currentTop10(l)) +
                                dMat(currentTop10(l), destStop) <=
                                DRONE_DIST_LIMIT
                                sumOfTauD = sumOfTauD + ((TauD(currentStop,
                                    currentTop10(l)) + TauD(currentTop10(l),
                                    destStop)) ^ ALPHA * (dMat(currentStop,
                                    currentTop10(l)) + dMat(currentTop10(l),
                                    destStop)) ^ BETA);
                            end
                        end

                        if sumOfTauD > 0 % If there are any cities that were
                            feasible for the drone to visit
                            % Find max drone probability
                            consideration = currentTop10;
                            Piljk = zeros(1,length(currentTop10));

                            % Calculate individual drone probabilities
                            for l = 1:length(currentTop10) % l is the
                                intermediate node between the node of
                                departure and rendezvous node with the truck
                                if dMat(currentStop, currentTop10(l)) +
                                        dMat(currentTop10(l), destStop) <=
                                        DRONE_DIST_LIMIT
                                    Piljk(l) = ((TauD(currentStop,
                                        currentTop10(l)) +
                                        TauD(currentTop10(l), destStop)) ^
                                        ALPHA * (dMat(currentStop,
                                        currentTop10(l)) +
                                        dMat(currentTop10(l), destStop)) ^
                                        BETA) / (sumOfTauD);
                                end
                            end

                            % Find the most likely event
                            maxDroneProb = max(Piljk);
                            maxInterStopIndex = find(Piljk==maxDroneProb);
                            while length(maxInterStopIndex) > 1
                                maxInterStopIndex(randi([1
                                    length(maxInterStopIndex)])) = [];
                            end

                            probReached = false;
                            probAcceptance = rand();
                            probCumulative = maxDroneProb;

                            while probReached == false

                                if (probCumulative >= 1 - probAcceptance)
```

```
                                    RouteD(k,moveD(k)+1) = currentStop;
                                    RouteD(k,moveD(k)+2) =
                                        consideration(maxInterStopIndex);

remainingStops(remainingStops==consideration(maxInterStopIndex)) = [];

currentTop10(currentTop10==consideration(maxInterStopIndex)) = [];
                                    RouteD(k,moveD(k)+3) = destStop;
                                    probReached = true;
                                    droneMoveDist = dMat(currentStop,
                                        RouteD(k, moveD(k) + 2)) +
                                        dMat(RouteD(k, moveD(k) + 2),
                                        destStop);
                                    moveD(k) = moveD(k) + 3;
                                else
                                    % remove this stop from consideration this
                                      move
                                    Piljk(maxInterStopIndex) = [];
                                    consideration(maxInterStopIndex) = [];
                                    maxDroneProb = max(Piljk);
                                    maxInterStopIndex =
                                        find(Piljk==maxDroneProb);
                                    while length(maxInterStopIndex) > 1 %
                                        maxInterStopIndex might return a
                                        vector but take only one instance
                                        maxInterStopIndex(randi([1
                                            length(maxInterStopIndex)])) = [];
                                    end
                                    probCumulative = probCumulative +
                                        maxDroneProb;

                                end

                            end

                            cs = 2;

                        else % There is not a feasible sortie for the drone,
                               so
                             % just move the drone straight to where the truck
                             % went (move the drone with the truck in this
                             % move).

                            cs = 1;

                        end
                        moveT(k) = moveT(k) + 1;
                        currentStop = destStop; % Reset the currentStop to the
                            stop where the truck and drone just arrived.

                    elseif isempty(currentTop10) % The truck has visited the
                        last stop, so the drone should join it there.
                        moveT(k) = moveT(k) + 1;
```

```
                cs = 1;
            end

            switch cs
                case 1 % The truck and drone moved together
                    cost(k) = cost(k) + truckMoveDist;
                case 2 % The drone took a sortie, so take the max of
                         the individual distances
                    cost(k) = cost(k) + max(truckMoveDist,
                        droneMoveDist / DRONE_SPEED);
            end

        end
    end

    % Complete the circuit from the last city to the first
    RouteT(k, moveT(k)) = RouteT(k, 1);  % Return the truck home
    cost(k) = cost(k) + dMat(RouteT(k, moveT(k) - 1), RouteT(k,
        moveT(k)));

    % Calculate the amount of change in truck pheromone along the
      paths taken by truck k (but don't apply pheromone yet)
    for i = 1:moveT(k) - 1
        currentCity = RouteT(k, i);
        nextCity = RouteT(k, i + 1);
        sumOfChangeInTauT(currentCity, nextCity) =
            sumOfChangeInTauT(currentCity, nextCity) + 1 / cost(k);
        sumOfChangeInTauT(nextCity, currentCity) =
            sumOfChangeInTauT(currentCity, nextCity); % Do the same in
            other direction
    end

    % Calculate the amount of change in drone pheromone along the
      paths taken by drone k (but don't apply pheromone yet)
    for sortie = 1:moveD(k)/4
        s1 = 3*sortie - 2;
        s2 = s1 + 1;
        s3 = s2 + 1;
        sumOfChangeInTauD(RouteD(k,s1), RouteD(k,s2)) =
            sumOfChangeInTauD(RouteD(k,s1), RouteD(k,s2)) + 1 /
            cost(k);
        sumOfChangeInTauD(RouteD(k,s2), RouteD(k,s1)) =
            sumOfChangeInTauD(RouteD(k,s1), RouteD(k,s2)); % Do the
            same in other direction
        sumOfChangeInTauD(RouteD(k,s2), RouteD(k,s3)) =
            sumOfChangeInTauD(RouteD(k,s2), RouteD(k,s3)) + 1 /
            cost(k);
        sumOfChangeInTauD(RouteD(k,s3), RouteD(k,s2)) =
            sumOfChangeInTauD(RouteD(k,s2), RouteD(k,s3)); % Do the
            same in other direction
    end

    % Ant pair k is done constructing its tour this iteration
```

```
        end

        % All ant pairs have constructed their tours

        % Check to see if the best truck and drone route found this iteration
        % is better than the bestMinCost so far.
        [minIterCost, bestIdx] = min(cost);
        if minIterCost < bestMinCost
            toc % Output the elapsed time to find this solution
            bestMinCost = minIterCost;
            bestTruckRoute = nonzeros(RouteT(bestIdx,:));
            bestDroneRoute = nonzeros(RouteD(bestIdx,:));
        end

        % Keep Track of Best Cost Each Iteration for the Convergence Plot

%*****************************************************************************
        %    Title: tsp_ga_basic(nStops?, popSize, numIter, xy )
        %    Author: Robert Rich
        %    Date: October 6, 2017
        %    Code version: 1.0.1.0
        %    Availability:
https://www.mathworks.com/matlabcentral/fileexchange/60640-dtsp_ga_basic-
nstops-popsize-numiter-xy-range-speed?s_tid=prof_contriblnk
        %

%*****************************************************************************
          distHistory(iter) = bestMinCost;

        % Update the pheromone levels in the network
            for i = 1:nStops
                for j = 1:nStops
                    if (i ~= j)
                        %Pheromone evaporation
                        TauT(i, j) = (1 - RHO) * TauT(i, j);
                        TauD(i, j) = (1 - RHO) * TauD(i, j);

                        % Pheromone deposit
                        TauT(i, j) = TauT(i, j) + sumOfChangeInTauT(i, j);
                        TauD(i, j) = TauD(i, j) + sumOfChangeInTauD(i, j);
                    end
                end
            end

          iter

    end   % Do another iteration of pheromone updating
   toc % Output the elapsed time to complete the algorithm

    % The algorithm has finished all of its iterations.

    % Make the Convergence Plot
```

```
%***************************************************************************
    %    Title: tsp_ga_basic(nStops?, popSize, numIter, xy )
    %    Author: Robert Rich
    %    Date: October 6, 2017
    %    Code version: 1.0.1.0
    %    Availability:
https://www.mathworks.com/matlabcentral/fileexchange/64653-tsp_ga_basic-
nstops-popsize-numiter-xy
    %

%***************************************************************************
    iterHist = 1:iter;
    subplot(2,2,1)
    plot(iterHist, distHistory(iterHist),'k-+');
    title(sprintf('Convergence: Min Cost = %1.4f',bestMinCost));
    xlabel('Iteration'); ylabel('Cost');

    % If the program generated its own coordinates and distance matrix
    if exist('xy','var')
       % Update graph with new pheromone levels
        % This gives us the rgb values.
        % Color the pheromone paths for the truck -- these paths are black.
        colormap(scale)
        v = TauT; % my matrix
        map = colormap;
        minv = min(v(:));
        maxv = max(v(:));
        ncol = size(map,1);
        s = round(1+(ncol-1)*(v-minv)/(maxv-minv));
        rgb_image = ind2rgb(s,map);

        % Color the pheromone paths for the drone -- these paths are blue.
        colormap(scale2)
        v2 = TauD; % my matrix
        map = colormap;
        minv = min(v2(:));
        maxv = max(v2(:));
        ncol = size(map,1);
        s = round(1+(ncol-1)*(v2-minv)/(maxv-minv));
        rgb_image2 = ind2rgb(s,map);

        % Plot the graphs showing the relative pheromone levels (brighter
        % colors indicate more intense pheromone levels).
        subplot(2,2,3);
        gMinimum1 = min(min(rgb_image(:,:,2)));
        bMinimum1 = min(min(rgb_image(:,:,3)));
        rMinimum2 = min(min(rgb_image2(:,:,1)));
        bMinimum2 = min(min(rgb_image2(:,:,3)));
        for num = 1:nStops
            for j = 1:nStops - num
                i=num:j:num+j;
                if (rgb_image(num, num+j, 2) == gMinimum1) &&
                       (rgb_image(num, num+j, 3) == bMinimum1)
                    p1 = plot(xy(i,1), xy(i,2),'ks-', 'Color', [rgb_image(num,
                    num+j, 1) rgb_image(num, num+j, 2) rgb_image(num,
                    num+j,3)], 'LineWidth',8); hold on;
                elseif (rgb_image2(num, num+j, 1) == rMinimum2) &&
                       (rgb_image2(num, num+j, 3) == bMinimum2)
                    p2 = plot(xy(i,1), xy(i,2), 'k--', 'Color',
                        [rgb_image2(num, num+j, 1) rgb_image2(num, num+j, 2)
                        rgb_image2(num, num+j,3)], 'LineWidth',2); hold on;
                else
                    plot(xy(i,1), xy(i,2),'ks-', 'Color', [rgb_image(num,
                        num+j, 1) rgb_image(num, num+j, 2) rgb_image(num,
                        num+j,3)], 'LineWidth',8);
                    plot(xy(i,1), xy(i,2), 'k--', 'Color', [rgb_image2(num,
                        num+j, 1) rgb_image2(num, num+j, 2) rgb_image2(num,
                        num+j,3)], 'LineWidth',2); hold on;
                end
            end
        end

       p3 = plot(xy(:,1), xy(:,2),'k.','MarkerSize',20); hold on; % Show the
           cities

       for i = 1:nStops

      text(xy(i,1),xy(i,2),num2str(i),'VerticalAlignment','bottom','Horizontal
          Alignment','center')
       end

       title('Relative Pheromone Values on Graph')
       xlabel('x-coordinate (km)')
       ylabel('y-coordinate (km)')
       legend([p1 p2 p3],{'Truck', 'Drone', 'Stop'},
           'Location','bestoutside','Orientation','horizontal')

       % Plot the best cost drone and truck routes found

%*****************************************************************************
       %    Title: dtsp_ga_basic(nStop?s, popSize, numIter, xy, range, speed )
       %    Author: Robert Rich
       %    Date: October 12, 2017
       %    Code version: 1.4.0.0
       %    Availability:
https://www.mathworks.com/matlabcentral/fileexchange/60640-dtsp_ga_basic-
nstops-popsize-numiter-xy-range-speed?s_tid=prof_contriblnk
       %

%*****************************************************************************
       subplot(2,2,2);

       % Plot the truck route
       p1 = plot(xy(bestTruckRoute,1),  xy(bestTruckRoute,2),'ks-', 'Color',
           'k'); hold on;
       % Plot the drone route
       for startNode = 1:3:length(bestDroneRoute) % number of sorties
           vectorToPlot = bestDroneRoute(startNode:startNode+1);
           plot(xy(vectorToPlot,1),  xy(vectorToPlot,2), 'k--', 'Color', [0 0
               1]); hold on;
           vectorToPlot = bestDroneRoute(startNode+1:startNode+2);
           p2 = plot(xy(vectorToPlot,1),  xy(vectorToPlot,2), 'k--', 'Color',
              [0 0 1]); hold on;
       end

       % Plot the cities
       p3 = plot(xy(:,1), xy(:,2),'k.');

       title('Best Routing Found for Truck and Drone');
       xlabel('x-coordinate (km)');
       ylabel('y-coordinate (km)');
       legend([p1 p2 p3],{'Truck', 'Drone', 'Stop'},
           'Location','bestoutside','Orientation','horizontal')
    end

end % end of function
```

## Appendix B

```
function [Cnn] = NearestNeighbor(dMat, nStops)
% This greedy algorithm constructs a route starting at stop 1. The function
  returns
% the total distance of the route, which is constructed by consecutively
  choosing the next
% closest stop ("nearest neighbor") to the currentStop until all nStops have
  been visited.
    Cnn = 0.0; % Length of the tour formed by NearestNeighbor Heuristic
    firstStop = 1;
    currentStop = firstStop;
    remainingStops = 2:nStops;

    while ~isempty(remainingStops)
        minValue = inf;
        for destCity = 1:length(remainingStops)
            if dMat(currentStop,remainingStops(destCity)) < minValue
                minCityIndex = destCity;
                minValue = dMat(currentStop,remainingStops(destCity));
            end
        end
        Cnn = Cnn + minValue;
        currentStop = remainingStops(minCityIndex);
        remainingStops(minCityIndex) = [];
    end
    Cnn = Cnn + dMat(currentStop,firstStop);
end
```