[TYPE THE COMPANY NAME]

Evolutionary Methods for the Mutli-Vehicle Route Tolerant Routing Problem

Renxiong Wang[1], Domenico Amodeo[2], Aquia Richburg[3]

5/12/2016

1 University of Maryland Department of Physics.

2 George Washington University College of Engineering and Applied Science.

3 University of Maryland Department of Applied Mathematics

Motivation

The vehicle routing problem first posed in 1959 by G.B. Dantzig (Dantzig, 1959), has been studied extensively in large part due to its practical applications. Several good surveys exist detailing the development of solution approaches to this problem, over the last half century; we recommend "Fifty Years of Vehicle Routing" (LaPorte, 2009), for a comprehensive review of the topic. These authors categorize VRP approaches into *Classical Heuristics* and *Metaheuristics.* However, many researchers have combined elements from Classical and Meta methods to refine and improve upon solutions of canonical problems. Classical strategies such as C*luster-First, Route Second,* provide a method of decomposing the VRP into smaller sub-problems which can then be treated using TSP techniques. More modern methods such as the *Genetic Algorithm* allow for vehicle-location assignment to emerge as the algorithm progresses. Recent authors have developed variants of the *Genetic Algorithm* and tested it on a set of canonical problems with several variants of local search (Baker et al,2003). They noted that while their algorithm produces comparable results on fairly simple graphs, it does not outperform TABU search.

Researchers have developed solutions to a range of real world inspired nuances to the traditional VRP including various types of capacity and time constraints, or additions such as multiple depots. We intend to propose a solution to a VRP variant that addresses the dynamic challenges route planners and drivers face on a regular basis; probabilistic route disruption and the need to redirect and mid journey. One can easily imagine a convoy in a conflict zone faced with the need to assess an initial route based on some *a priori* probability of a threat, and the sudden need to adjust its route mid trip when faced with a sudden attack that degrades the planned route. Delivery companies may plan their daily routes with some insight into daily traffic patterns, but find themselves with the need to re-route delivery trucks when a traffic accident suddenly degrades the chosen route. Common to both of these scenarios is the assumption that planners performed a general assignment of locations to delivery trucks. One method for clustering is provided Fisher and Jaikumar, (Fisher and Jaikumar, 1981). We address a variant of the VRP that addresses the following element of the problem:

1) Multiple vehicles are available to delivery to D≤ (N-1). Where D is the delivery locations and (N-1) is the complete set of nodes, depot excluded.
2) After an initial route optimization, disruptions occur and the dispatcher has the ability to re-route deliveries.
3) The objective function is to minimize the time to deliver all packages.

The implication for these three elements is that a *Hamiltonian Cycle* is not required, and dispatchers have added flexibility in route design. The objective is also customer centric, since the aim is not to minimize cost in terms of time of money for the delivery company.

Given current security concerns this type problem has also received attention from engineers, infrastructure planners and policy makers. The concept of *infrastructure resiliency* has become increasingly relevant in recent years. Researchers at the Naval Post Graduate School in Monterey California have addressed a similar problem to the current VRP variant, coined the term *operational*

resiliency (citation). Their research incorporates elements of game theory (attacker-defender models), and network optimization to build a concept referred to as the *resiliency curve*. This research aims to measure the resiliency of a network, by finding the best delivery routes when faced with a worst case scenario of *k-edge* disruptions where: k∈ {1…K} and K maximum edges of interest. This analysis produces, a table listing the cost associated with the worst single edge through worst *k* edges, and a curve to visualize the cost impact at each level. This is a helpful tool for policy makers looking to make infrastructure investments.

Our research will contribute to both the theoretical development of the Vehicle Routing Problem, as well as the field of *Infrastructure Resiliency* and *Risk Management*. Application of the algorithm presented in this paper will allow planners and security experts to gain additional insights regarding the resiliency and flexibility of their plans. In this paper we present initial results for an algorithm designed for use in the FTVRP

Solution Overview

We approached the Fault Tolerant Vehicle Routing Problem (FTVRP) as three main sub-problems:

1) Assignment of trucks to delivery locations.
2) Determining shortest path from the depot to each delivery location.
3) Re-assessing the planned route and re-directing vehicles upon learning about a disruption.

We present four methods for addressing the assignment problem. Two of these methods are independent of the shortest path problem, and are akin to traditional *cluster-first route second* approaches. The other two assignment approaches, assign vehicles to delivery nodes, while searching for the shortest routes. We address the shortest path problem in two parts. First we present an intuitive *self-evolving algorithm* for finding the shortest from the i^{th} node to the j^{th} target node. We then present two algorithms which search for the best solution. The two algorithms we apply are the well-established genetic algorithm, and what we refer as the *reject or accept algorithm*. We also study a method for combining *self-evolution* and *reject and accept* methods. Finally, we present a method for quickly and intuitively dealing with route disruptions.

Data Preparation

The network of roads, depots and delivery locations is provided as G= (V, E), where G is graph composed of a set of nodes V ∈ (1…n), and E a set of edges defined on V, and represented by the n x n matrix G. Each entry, c_{ij}, represents the cost in time to travel between node i and j $\forall$ (i , j) ∈ E. For non-adjacent node c_{ij}=-1. The delivery location are represented by D, where D ⊂V. An example of the resulting network is shown in Figure 1.

Each edge defined on a pair of nodes (i, j) has the provability of disruption, $p_{ij.}$ These probabilities are presented in an n x n matrix. When a disruption occurs, a multiplier M, is applied to the edge which linearly increases the cost associated with traveling that edge. For this problem, we assume that there

are no capacity constraints on either the route or the vehicle. We also assume disruptions on each edge are binary over the run time of the experiment. At time t_0=0, we assume no disruptions and view all edge cost as expected cost. The edge cost can be viewed as a random variable X such that:

$$E[X] = (p_{ij})(c_{ij})(M) + (1 - p_{ij})(c_{ij}).$$

Shortest paths are determined based on these expected cost.

Intuitive Solution

Our intuitive solution, referred to as the *self-evolution method*, is based on insight from real world context; we expect to deliver to nodes closer to the depot earlier. Using this insight as a starting point we improve our solution through numerous iterations. This algorithm starts with two matrices, the first has dimension n x d where n is the total number of nodes, and d is the number of delivery locations. This matrix holds the shortest distance from each node, to each target node. These distances were calculated using the standard *Broad First Search* algorithm. The second matrix, also dimension n x d, holds the subsequent node to the i^{th} node on the shortest path from the i^{th} node to the j^{th} target node.

This algorithm begins by selecting current vehicle location from the set of total vehicles. Every current location of a vehicle has a path to a target node, and the selected vehicle must proceed to at least one target node. The solution evolves by assigning a probability of target node selection based the length of the path from a current vehicle location d_i target node D_k. The probability of selection is made higher for shorter paths by using the following probability distribution function:

$$p_k = \frac{e^{-di_k}}{\sum_{k=1}^{K} e^{-di_k}}$$

Once a D_k is selected, it is removed from the set of targets for subsequent probability calculations. The starting point for the m^{th} vehicle is removed from the set of starting points, the time is updated, and the process is repeated until all target nodes have been visited. The completion time is recorded for the entire run of the algorithm. The simulation is repeated N times, for some large value of N, and the solution that produces the best solution time is recorded. The time complexity of this method is $O(mn^2)$,

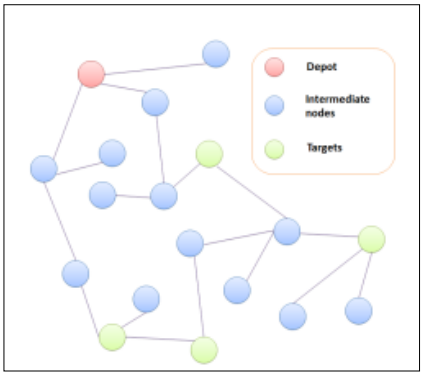

Figure 1: A graph with 1 depot, 4 delivery locations, and 13 total nodes.

since a shortest path must be determined for every target node, for every vehicle.

This method has two main advantages. First, the intuitive nature helps make the case for practical adoption for drivers and dispatchers. Second, this algorithm is easily adapted to the appearance of a route disruption. The *self-evolving* algorithm finds decent solutions for shortest on its own, however, we applied two searching algorithm for improve upon the solutions; the *genetic algorithm*, and the *reject and accept algorithm* similar to simulated annealing.

Searching Methods

Genetic Algorithm

The *Genetic Algorithm* starts with an initial population of solutions, followed by s set of random alterations that result in either an improvement of reduction in route time. In this case the fitness of each solution is calculated as $e^{-\lambda}$, where $-\lambda$ is the negative completion of the path from the depot to a

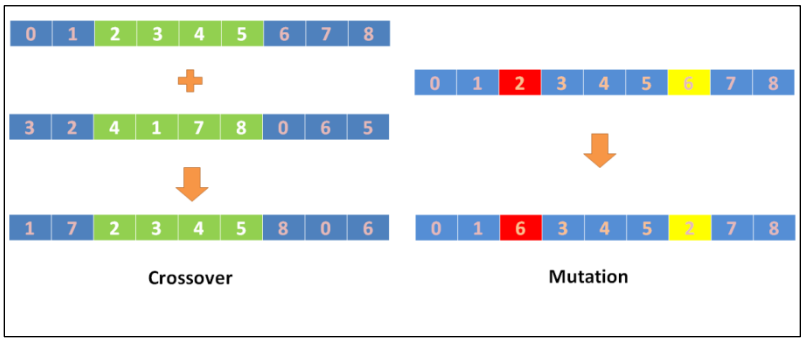

Figure 2: Demonstration of mutation and cross over transformation in the genetic algorithm.

given target node. From this population of initial paths, we select a subset. For each path the

$\mathbf{Pr}(\boldsymbol{mutation}) = \boldsymbol{\phi_1}$ and $\mathbf{Pr}(\boldsymbol{crossover}) = \boldsymbol{\phi_2}$. Where $\boldsymbol{\phi_i}$ is a threshold probability established by decision maker. A mutation involves changing the order of two edges within a single path, whereas a crossover, involves exchanging sub-paths between two existing solutions while simultaneously removing an existing sub path from each parent solution to create a new solution. The aim is to keep the population size consistent from one sample to the next. If a new solution has better fitness than its parent, the parent is replaced in the sampling domain. The fittest solution is maintained after N generations of mutation and cross over. Figure 2 demonstrates how the idea of mutation and crossover to create new solutions.

Reject and Accept Method

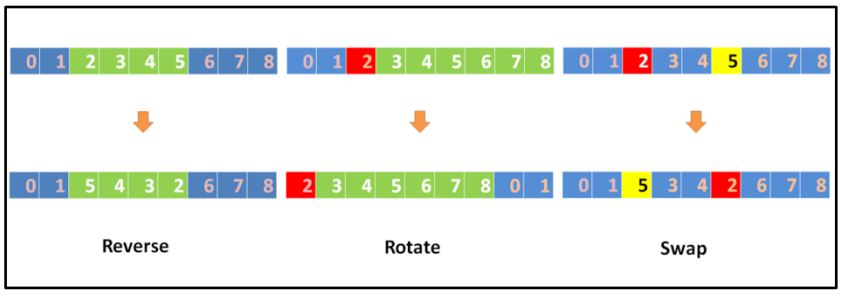

Figure 3: Reject and accept method

The *Reject and Accept Method* (*RAM*) is similar to both the *Genetic Algorithm* and *Simulated Annealing (SA)*. Similar to the *GA* this method performs a series of transformation on the initial paths to develop new solutions and evaluate the fitness, $\psi = e^{-\lambda}$, of these solutions. Similar to *SA,* the *RAM*, accepts solutions that are less fit with a given probability. This method performs three operations on an existing solution.

Reverse*: replaces a sub-path of an existing solution with the mirror image of that sub-path.*

Rotate*: Divides the path into one sub-path before a pivot node, and one sub-path after a given node. The first sub-path is rotated from before the pivot node, to the end of the second sub-path.*

Swap*: Randomly selected two nodes, and switch the order in which they appear in the path.*

Although all three operations are performed two main steps occur between each operation. First, a new fitness $\psi_{new} = e^{-\lambda_{new}}$ is calculated. Next we make an accept or reject decision for the new solution by comparing ψ_{new} and ψ. Improved solution will always be accepted. Worse solutions will be accepted with probability inversely related to the fitness, $\Pr(accept) = \frac{\psi_{new}}{\psi_{new} + \psi}$.

Randomly altering solutions is a structured way, allows us to quickly test a wide range of solutions, and retain the best ones we find. The rejection policy, allows us to accept a worse solution, with the understanding that this solution could lead to a locally sub-optimal solution with the opportunity for increased improvement after additional transformations.

Comparison

To test the quality of each solution we ran, a simple test case. The graph in question consisted of 45 nodes, with one truck and no disruptions. We can see that the *self-evolution algorithm* and *accept and reject method* produce better completion times than the well-established genetic algorithm, and converge to nearly the same quality of solution in a relatively few number of runs. According to (Baker

et a, 2003) TABU search is among the best methods for solving the VRP. Since the two new methods we propose out-perform the genetic algorithm, we expect the solution quality to be relatively close to TABU search.

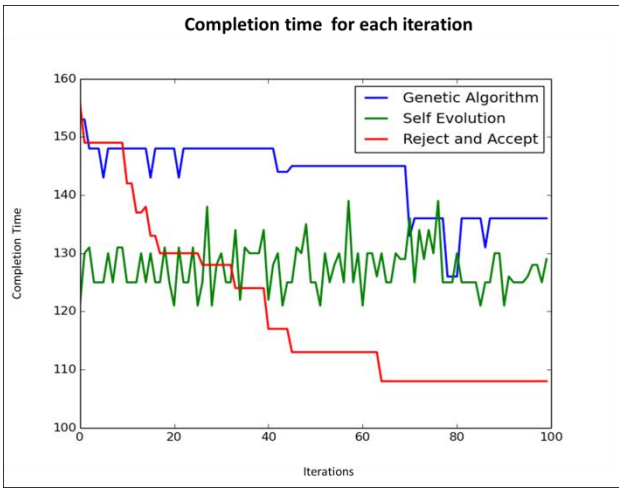

Figure 4: Comparision between 3 methods I

We can see from Figure 4 below that the *RAM* does accept some worse solutions early, however, as the algorithm progresses, no worse solutions are accepted. As the difference between high and lower quality solutions increases, and as lower quality solutions are removed the opportunity to that a low quality solution is sampled becomes increasing rare. The self-evolving algorithm produces random results, with no trend in either improvement or worsening. However, the self-evolving method produces relatively good solutions early on in the process. *GA* improves over time, but at a slower rate than *RAM.* This comparison provides several useful insights. First, we can see that the *RAM* will outperform the other two methods in the long run. Second, we can see that the *self-evolving method* is good at quickly finding a set of candidates for the population set, which in turn can be quickly searched

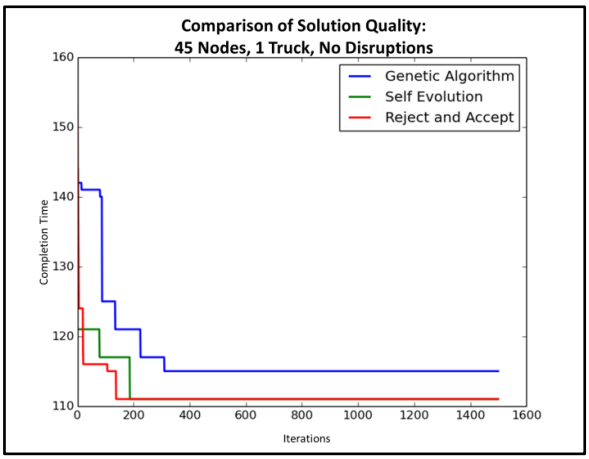

Figure 5: Comparision between 3 methods II

by *RAM* to search for the minimum.

Assignment

We explored four methods of assigning trucks to delivery locations. The first two methods are based on the concept of *cluster-first, route second* and are done independently of the shortest path and search algorithms. The first method, groups target nodes based on topological connections. A subset of nodes topologically connected to each other, but topologically disconnected from another group of targets nodes form a single cluster. Each cluster is assigned one vehicle. The best route is then found for each cluster. This method has the advantage of being computational easy to execute, and can quickly reduce the size of the graph, by eliminating sub graphs that are neither targets nor intermediate nodes.

The second method entails pre-determining the number of clusters desired. The target nodes are ranked based on their distance from the depot. We then assign clusters to each of the targets in decreasing order of their distance from the depot. Each node is assigned to a cluster based on its proximity to target node defining that cluster. In the event that one cluster has many targets, and another has very few, this technique will underutilize some of the vehicles and result in high completion times.

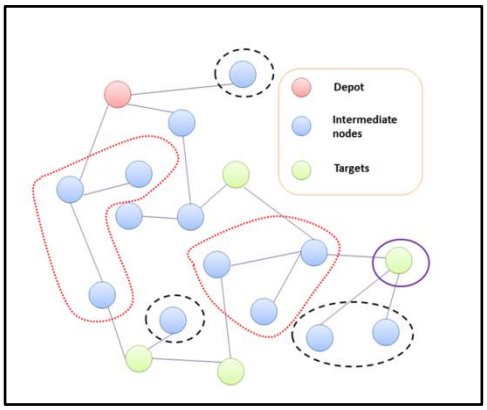

Figure 6: Grouping by topological connectivity

In the third method assignment of vehicles is achieved as part of the routing *self-evolution*. As discussed earlier, the self-evolving algorithm, randomly selects the next node visited for each vehicle based on a probability inverse to each path's length. This process advances until a vehicle reaches a target node. Once at a target node the evolution continues until the next target node is reached. Eventually all target nodes will be reached by a vehicle and the algorithm terminates, having found a near optimal solution and an assignment of locations to vehicles.

The final assignment method is based on our insight from the comparison of search methods. We previously observed that the solutions of the *self-evolution* method can be improved by the *reject and accept method*. The *reject and accept method* applies random transformations to the particular path followed by each vehicle. As each new path is created and tested for acceptance, each target node will

eventually fall on one of the near optimal solutions when the algorithm terminates. Again, this method assigns vehicles to target nodes while searching for the minimum path. As opposed to *cluster-first, route second* strategy, this approach is more robust and generalizable when target nodes are randomly distributed, and we are less likely to underutilize any given vehicle.

Figure 6 compares the completion time for each assignment method. Method 3 and Method 4 are the best performers. We selected Method 4 for our final algorithm because generalizability of the *self-evolution* plus *reject and accept method*. This method is also more capable of handling disruptions. The assignment process is more dynamic and flexible, allowing for vehicles to essentially aide each other when a disruption occurs. This algorithm has a complexity of O (n^2 x f), where f is the number of disruptions that occur. We know this method produces acceptable results for two reasons. First, it will always improve on the *self-evolving method*. This was empirically demonstrated. Second, we see that this method outperforms the *genetic algorithm,* a well-established method for solving the VRP. However, it must be admitted that this problem was applied to a relatively simple set of test cases, and not to a set of canonical test cases. It should also be noted that we conducted only a cursory literature review, and admit it is possible that modified genetic algorithms may exists which addresses the FTVRP.

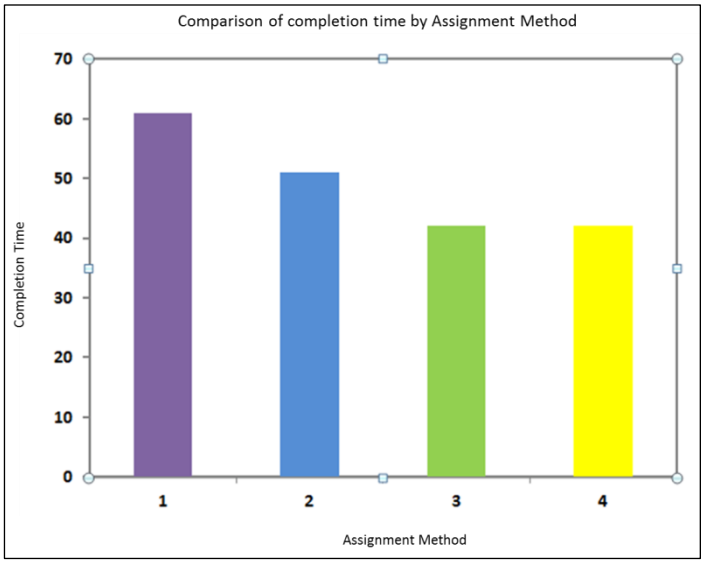

Figure 7: Comparison of assignment methods

Disruptions

Although dispatchers make their best efforts to optimize routes, situations often develop in real time which changes the cost of a pre-determined route. Traffic accidents, natural disaster, deliberate sabotage can all cause routes or segments of routes to become completely impassable, or extremely costly or risky to travel. Information about these disruptions can come from multiple sources. Traffic reports, intelligence reports, emergency responders, or firsthand witnesses. The expected cost may be useful for planning purposes, but in reality the vehicle will experience a particular realization of the random variable of the cost. In our example, this is somewhat binary. There will be no disruption and the cost of a segment will be the initial cost before expected cost, or a disruption will occur and the cost experienced will be a multiple, M>1 , of the original cost. We assume that, only one disruption can happen per route, and that disruptions on one route do not impact other routes.

This problem is in fact familiar to many commuters. The average commuter may have some preferred or optimal route they take to work, accounting for mileage, time of day, weather, and tolls. However, upon hearing a traffic report, many commuters typically assess other options and decide whether the cost or re-routing outweigh the cost of continuing on the planned route. This common experience provided the insight for our solution to dealing with disruptions.

A disruption occurs at some time, t>0, along the planned route. At this point, a pseudo-node is added to the current location of all vehicles. These pseudo-nodes serve the role of depots, in the initial routing problem. All delivery locations that have already been visited are transformed to intermediate nodes in the graph. New columns and rows are added to the matrix, and are adjacent exclusively with the nodes defining the current edge. The costs incurred on respective edges by the disruptions are updated, and the failure rate graph is updated to reflect 0% failure on currently disrupted routes. The new expected costs are updated, and the routing algorithm is re-executed. The algorithm used is the self-evolution plus reject and accept method. This allows for vehicles to be re-assigned if the situation calls for it.

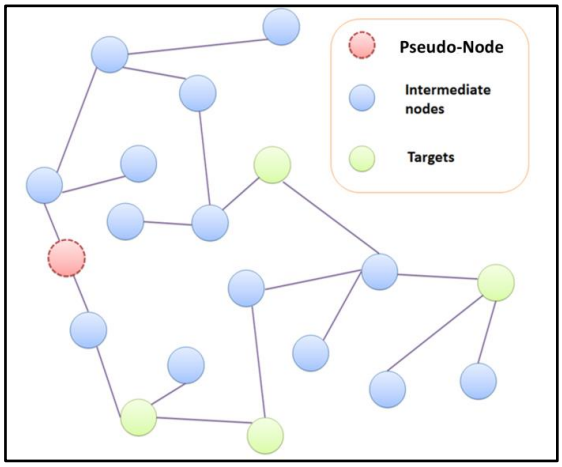

Figure 8: Pseudo -Node

Conclusions and Limitations

We set out to develop a solution for a variant of the traditional VRP. There were three main variations. First, only a subset of nodes required delivery. Second, disruptions occur to the network at some time $t_o>0$, which require the dispatcher to re-assess the initial routing. Third, the objective function is to minimize the delivery of the last package. The problem does not require return to the initial depot.

We decomposed the problem into three part; finding the shortest path to delivery locations, assigning vehicles to locations, and dealing with disruption. Our proposed solution uses a self-evolution algorithm to create an initial population of acceptable solutions, and a search algorithm similar to simulated annealing to searching and modifying these solutions to find a near-optimum. We deal with disruptions by adding new nodes to current vehicle locations, updating the matrix representation of the network, and re-applying the algorithm. Our method produces sufficiently good results to be considered for real world application. This is based on our comparison of our process to both the *Genetic Algorithm* and the *Self-Evolution Algorithm.* Our method always improves upon the *Self-Evolution Algorithm*, and in the basic test cases out performs the well-established *Genetic Algorithm*.

This solution is limited to situations where the product being delivered in not unique to each location. Beverage distribution would be a simple example of this case. Nor does our solution cover capacity constraints. Fuel delivery is one case where we might expect this to arise. In situation such as this, we might need to relax the constraint on one vehicle per delivery location to meet the current demand. In situations, such as parcel delivery, our solution would not adequately address the problem, since it allows for re-assignment. If vehicles are already configured for specific deliveries, we would need to re-solve the VRP by *cluster first-route second methods,* in which case this becomes a more traditional VRP problem.

Recommendations for Future Work

The current formulation of the FTVRP does not account for capacity constraints, or delivery priorities. These two constraints are prominent in the real world where parcel delivery companies guarantee delivery for delivery times to certain customers, and where delivery vehicles must be configured before leaving the depot. The priority constraint may be formulated more generally as minimizing the deviation from a promised delivery time given a disruption occurs in route to that location.

References

Baker, Barrie M., and M. A. Ayechew. 2003. “A Genetic Algorithm for the Vehicle Routing Problem.” *Computers & Operations Research* 30 (5): 787–800.

Dantzig, G. B., and J. H. Ramser. 1959. “The Truck Dispatching Problem.” *Management Science* 6 (1): 80–91. doi:10.1287/mnsc.6.1.80.

Fisher, Marshall L., and Ramchandran Jaikumar. 1981. “A Generalized Assignment Heuristic for Vehicle Routing.” *Networks* 11 (2): 109–124.

Laporte, Gilbert. 2009. “Fifty Years of Vehicle Routing.” *Transportation Science* 43 (4): 408–416.

Nagata, Yuichi. 2007. “Edge Assembly Crossover for the Capacitated Vehicle Routing Problem.” In *Evolutionary Computation in Combinatorial Optimization*, 142–153. Springer. http://link.springer.com/chapter/10.1007/978-3-540-71615-0_13.

Prins, Christian. 2004. “A Simple and Effective Evolutionary Algorithm for the Vehicle Routing Problem.” *Computers & Operations Research* 31 (12): 1985–2002. doi:10.1016/S0305-0548(03)00158-8.