International Journal of Transportation Engineering and Technology
2021; 7(4): 97-103
http://www.sciencepublishinggroup.com/j/ijtet
doi: 10.11648/j.ijtet.20210704.12
ISSN: 2575-1743 (Print); ISSN: 2575-1751 (Online)

Extremely Fast "Solution" to the Large-Scale and Very Large-Scale Vehicle Routing Problem

James Riechel

Center for Information Systems & Technology (CISAT), Claremont Graduate University (CGU), Claremont, the United States

Email address:
james.riechel@cgu.edu

To cite this article:
James Riechel. Extremely Fast "Solution" to the Large-Scale and Very Large-Scale Vehicle Routing Problem. *International Journal of Transportation Engineering and Technology*. Vol. 7, No. 4, 2021, pp. 97-103. doi: 10.11648/j.ijtet.20210704.12

Received: October 21, 2021; **Accepted**: November 8, 2021; **Published**: November 17, 2021

Abstract: A solution to the vehicle routing problem (VRP) is presented that takes only quadratic space, $O(n^2)$, and quadratic time, $O(n^2)$, if n is the number of stops on a route. The input is assumed to be a list of stops of length n in longitude, latitude format. The output is an origin-destination (OD) matrix of size $O(n^2)$, which takes $O(n^2)$ time to build. The element (i, j) in the matrix is the approximate driving distance between stop i and stop j on the route. Each approximate driving distance takes constant or $O(1)$ time to compute. (The approximate driving distance appears in previous work by the author, published in URISA GIS-Pro '19 and CalGIS 2020.) This OD matrix is well-suited for solving large-scale and very large-scale VRP problems, since computing approximate driving distances is lightning fast. For instance, using real-world data, it took less than one (1) second to produce a route with 5,156 stops. The OD matrix can be used with any exact or approximation algorithm to find a route, including the nearest-neighbor approximation algorithm: Starting at an origin, the next closest stop is visited repeatedly, ending at the destination once all stops have been visited. Determining the next stop to visit takes linear or $O(n)$ time to compute, and this is done $O(n)$ times. This solution to the VRP is a polynomial-time, $O(n^2)$, approximation; it is not exact, but is extremely fast.

Keywords: Vehicle Routing Problem (VRP), Approximate Driving Distance, Manhattan Distance, Equirectangular Projection, Nearest-neighbor Approximation Algorithm

1. Introduction/Literature Review

The vehicle routing problem (VRP) is defined as follows: Beginning at a depot, a route is formed by visiting all stops before returning to the depot [1]. The VRP is a generalization of the famous "traveling salesman" problem (TSP) [4]. Finding an optimal route in which travel distance is minimized is known to be NP-hard [5]. Both the VRP and TSP are NP-complete problems [7]. An approximate solution is presented which uses the polynomial-time (P-time) nearest-neighbor approximation algorithm [12]. The contribution in this work is that an origin-destination (OD) matrix is built on the fly with approximate driving distances [8-10]. The OD matrix is usually a pre-computed input to the VRP algorithm.

Here are several related works in the VRP published by the Transportation Research Record (TRR): Wang et al. (2013) consider the vehicle routing problem of simultaneous deliveries and pickups with split loads and time windows (VRPSDPSLTW) [15]. They formulate the problem as a mixed-integer programming problem and use a hybrid heuristic algorithm. Ripplinger (2005) considers the case of the rural school vehicle routing problem. Ripplinger develops a mathematic model, and a new heuristic. His results are superior to existing VRP for the pickup and drop-off of students in rural areas [11]. Tang and Miller-Hooks (2006) define the VRP with solution shape constraints. They propose an interactive heuristic which when coupled with effective shape measures, produces solutions with significantly improved layout [14]. Figliozzi (2010) considers the case of VRP for emissions minimization (EVRP). He provides both a formulation and solution to EVRP [2].

Large-scale routes have hundreds of stops, such as the routes for package carriers like UPS, FedEx, DHL, OnTrac, Amazon, and USPS. Very large-scale routes have thousands

of stops, such as the routes for garbage and recycling trucks. The presented algorithm is a GIS solution to a GIS problem. The math is close to trivial. A distance or an approximate distance is really more of a concept or an idea than a mathematic number or equation.

The presented algorithm makes possible solutions to the VRP which were previously impossible. Computing routes with a large or very large number of stops in a fraction of a second forever changes route planning. Stops can be inserted and deleted, and a new route produced in real-time. Feedback from the client in the City of Chamblee, Georgia, indicates that the new routes are superior to previous routes, which were done partly "by hand." Adding a stop changes the route, but so can deleting one or more stops (it can change the order of the remaining stops).

The rest of the paper is organized as follows:

1. Section 2: Materials and Methods
2. Section 3: Case Study
3. Section 4: Results
4. Section 5: Limitations/Discussion
5. Section 6: Conclusion/Future Work
6. Appendix: The source code used to find the route in the case study for the City of Chamblee, Georgia

2. Materials and Methods

2.1. Approximate Driving Distance

Each line of the input file has a longitude, latitude pair (in degrees) of a stop in the route. First, convert these from degrees to radians. Then, compute the *x, y, z* coordinates of each stop:

$$x = \text{longitude} * r * \cos(\ _0)$$

$$y = \text{latitude} * r$$

$$z = \text{elevation}$$

where r is the radius of the Earth, and $_0$ is a centrally located latitude in the dataset. This forms an equirectangular projection [3].

Let (x_i, y_i, z_i) and (x_j, y_j, z_j) be two stops in a route. The approximate driving distance between them is:

$$\text{distance}(i, j) = \text{Abs}(x_i - x_j) + \text{Abs}(y_i - y_j) + \text{Abs}(z_i - z_j)$$

This approximate distance, the Manhattan distance [13], is both a better approximation of the actual driving distance than the Euclidean distance, and an order of magnitude faster to compute than the Euclidean [8-10]. See Zeager and Stitz (2016) for a description of Euclidean distance [16].

The Manhattan distance is extremely fast to compute:

Table 1. *Execution speed of Manhattan distance, aka "approximate driving distance".*

	Running time of 100 million calls (milliseconds)	Operations per second
Manhattan distance, aka "approximate driving distance"	296	337,837,838

A pilot study was performed to determine how accurate Manhattan distances are compared to actual driving distances. A nonrepresentative study of 200 green taxi cabs rides in New York City on January 1, 2016, starting at 12 AM EST, had the following distribution:

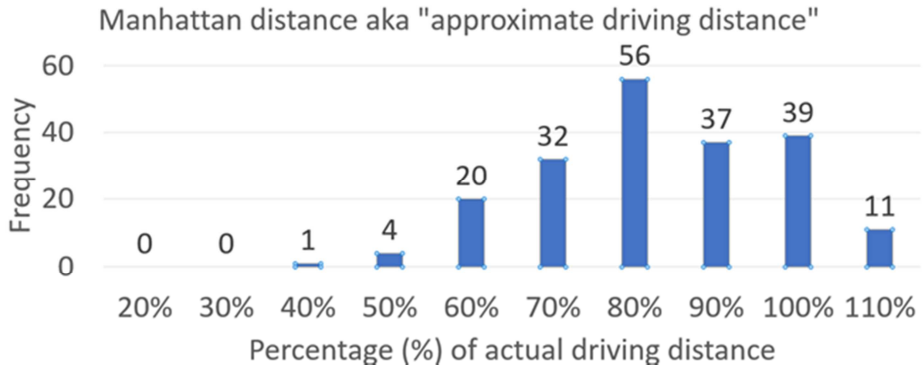

Figure 1. *Accuracy of Manhattan distance in NYC pilot study.*

It was confirmed that this distribution is normal. On average, the Manhattan distance is about 80% of the actual driving distance, and the Manhattan sometimes overestimates actual driving distance.

There are many reasons why Manhattan distance might underestimate actual driving distance, including:

1. Traffic controls (U-turns, one-way streets)
2. Manmade obstacles (bridges)
3. Natural obstacles (lakes, hills, mountains)
4. Equirectangular projection

2.2. OD Matrix and the Nearest-Neighbor Approximation Algorithm

As mentioned in the Abstract and Introduction, an OD matrix is built using approximate driving distances. This is important because computing approximate driving distances is lightning fast [8-10]. Building the OD matrix takes $O(n^2)$

space and $O(n^2)$ time, where n is the number of stops on the route.

Using this OD matrix, any exact or approximate algorithm can be used to find a route but for the sake of simplicity, the standard nearest-neighbor approximation algorithm is used. First, find the first stop (the one closest to the depot), and then successively the next closest stop, returning to the depot after the final stop is visited. This takes $O(n^2)$ time, where n is the number of stops on the route.

3. Case Study

Using data from the City of Chamblee, Georgia, a route with 5,156 stops was computed in under one (1) second. It was, however, necessary to expand the stack size of the executable, because of the size of the OD matrix which was over 26.5 million cells (5,156 rows, and 5,156 columns). The client was satisfied with the route computed.

If actual driving distances were used instead in the case study, the computation would take around 308 virtual days instead of less than one second (1s), if we assume the computation of each actual driving distance takes around one virtual second:

5,156 rows times 5,156 columns

= 26,584,336 cells

= 26,584,336 virtual seconds

= 443,072 virtual minutes

= 7,385 virtual hours

= 308 virtual days

4. Results

The map on the following page shows the produced route for the case study. The truck depot is located at the red star, stops are shown as blue dots, and the visitation order starting at the depot is given in unclassed colors (yellow to orange to red):

No truck could visit all 5,156 stops in one day. From what is understood, daily routes for each truck visit around 1,100 to 1,200 stops.

But the master route of 5,156 stops is a good test case for the presented algorithm. It is amazing that it completes in less than one second (<1s).

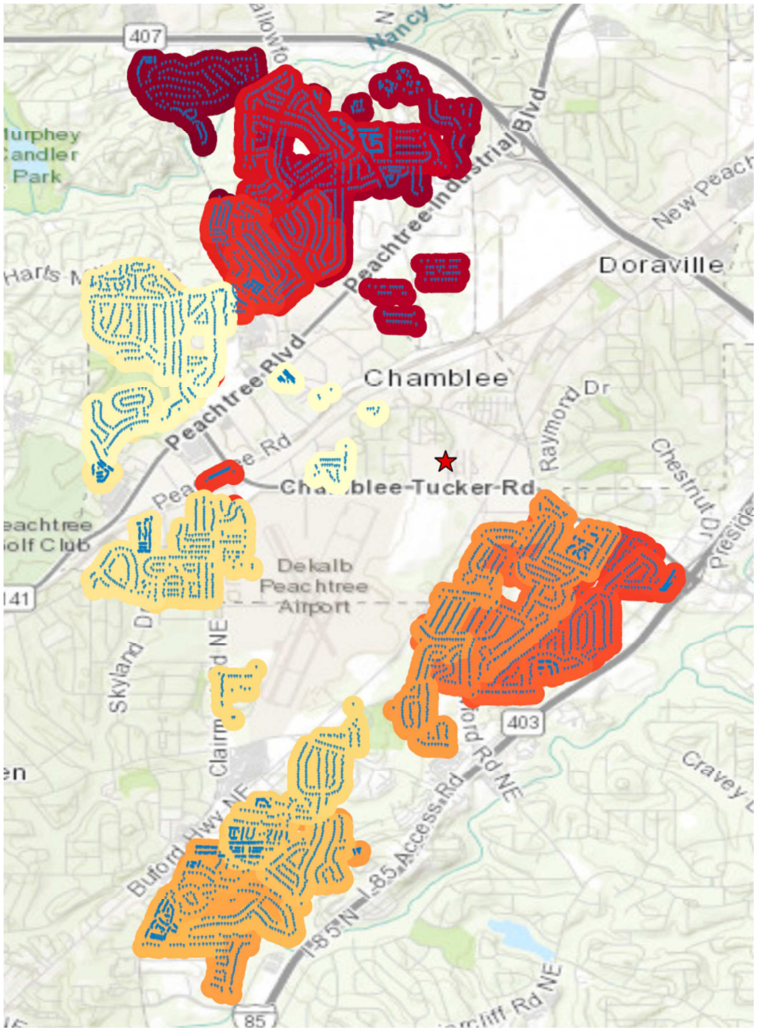

Figure 2. *Truck depot (red star), stops (blue dots), visitation order (yellow to orange to red).*

5. Limitations/Discussion

The main advantage of this solution to the large-scale or very large-scale VRP is its improved speed. Its main drawback is its 20% lower accuracy. Approximate driving distances are retained only to find the order of stops on a route. Once the route is determined, it is important to use other tools, such as driving directions in ArcGIS, Waze, or Google Maps to fine tune the route.

Li et al. (2016) introduce the "six Vs" of Geospatial Big Data: volume, variety, velocity, veracity, visualization, and visibility [6]. In a world filled with Big Data, where the *volume* of data points to compute distances between, and the velocity at which these distances are expected to be computed, are both extremely high, a fast algorithm for computing approximate distances may be the only choice. This introduces the issue of *veracity*: How reliably accurate these approximate distances are.

A number of optimizations were skipped in the case study because they were unnecessary. Since the client's route was produced in less than one (1) second, compiler optimization flags were not used. Also, software floating-point operations were used, not hardware which are faster. There might be one or two optimizations in the code itself that were not implemented because it was deemed unnecessary.

6. Conclusion/Future Work

It is impossible to gauge how "good" the presented solution to the VRP is because it cannot be compared to the exact solution, which would require *Factorial (5,156)* or *5,156!* time. This amount of time triggers an *Overflow* in the Calculator program built into Windows 10 Pro. The exact solution would likely take millions of years to compute. However, additional quantitative and qualitative data can be collected, such as:

1. Gallons of fuel saved using the new routes
2. Time saved using the new routes
3. Interviews with managers, truck drivers, and other employees

Such data has not yet been collected, but if the opportunity arises it will be collected in the future.

A serial algorithm has been developed to "solve" the VRP (see Appendix). A next step could be to parallelize this serial algorithm to get good speedup on larger problems than the one presented here, which computed a "master route" with 5,156 stops in under one second (<1s).

It should be possible to "solve" other NP-complete problems using this solution to the VRP. All it takes is a translation or transformation from another NP-complete problem space to this VRP problem space, and back again after a solution to the VRP is computed.

The accuracy of the VRP algorithm should be explored, perhaps with a smaller problem with a known exact solution.

Finally, the VRP algorithm can be implemented as a Python toolkit within ArcGIS Pro.

Acknowledgements

Carlo Frate of the City of Chamblee, Georgia, who provided wonderful test data for the presented algorithm

Professors Brian Hilton and Warren Roberts of the Center for Information Systems & Technology (CISAT) at Claremont Graduate University (CGU) for all their help and guidance.

Appendix

A1. First Source Code File, "stop.h"

```
#include <iostream>
#include <fstream>
#include <string>
#include <sstream>
using namespace std;

class Stop {

  public:

    int OBJECTID;
    string Name; // address
    double Duration;
    string RouteName;
    string Sequence;
    string CurbApproach;
    string Driver;
    double POINT_X; // longitude in degrees
    double POINT_Y; // latitude in degrees
    int Zone2020;

    double longitude, latitude; // in radians

    double x, y, z; // in meters

  void read_row_from_file(ifstream&);

  void set_xyz(double, double, double);

  void output_row();

  double approximate_distance(double, double, double);
};
```

A2. Second Source Code File, "stop.cpp"

```
#include "stop.h"

#define PI (3.14159265359)
#define DEG2RAD(deg) (deg/180.0*PI)
#define RAD2DEG(rad) (rad/PI*180.0)

void Stop::read_row_from_file(ifstream& input_file) {
```

```
    string line;

    getline(input_file, line);

    stringstream s(line);
    string word;

    getline(s, word, ',');
    OBJECTID = stoi(word);

    getline(s, Name, ',');

    std::string::size_type pos;

    //getline(s, Duration, ',');
    getline(s, word, ',');
    Duration = stod(word, &pos);

    getline(s, RouteName, ',');

    getline(s, Sequence, ',');

    getline(s, CurbApproach, ',');

    getline(s, Driver, ',');

    getline(s, word, ',');
    POINT_X = stod(word, &pos);

    getline(s, word, ',');
    POINT_Y = stod(word, &pos);

    getline(s, word, '\n');
    Zone2020 = stoi(word);

    longitude = DEG2RAD(POINT_X);
    latitude = DEG2RAD(POINT_Y);

  }


  void Stop::set_xyz(double x_mult, double y_mult, double
elevation) {

    x = x_mult * longitude;
    y = y_mult * latitude;
    z = elevation;
  }

  void Stop::output_row() {

    std::cout.precision(20);

    std::cout << OBJECTID << ',' << Name << ',' <<
Duration
      << ',' << RouteName << ',' << Sequence << ',' <<
CurbApproach << ',' <<
      Driver << ',' << POINT_X << ',' << POINT_Y << ','
<< Zone2020 <<
      std::endl;
  }

  double Stop::approximate_distance(double x2, double y2,
double z2) {

    return(fabs(x - x2) + fabs(y - y2) + fabs(z - z2));
  }
```

A3. Third Source Code File, “main.cpp”

```
  #include "math.h"
  #include "stop.cpp"

  #define EARTHRADIUS (6371000)

  main() {

    const int number_of_stops = 5156;

    const double depot_longitude = DEG2RAD(-84.291861);
    const double depot_latitude = DEG2RAD(33.887694);
    const double elevation = 304.8; // meters

    ifstream input_file("ServiceLocations2020.csv");
    string line;
    getline(input_file, line);

    //Stop Stops[number_of_stops];
    Stop* Stops = new Stop[number_of_stops];

    for (int i = 0; i < number_of_stops; i++) {

      Stops[i].read_row_from_file(input_file);
    }

    input_file.close();

    // compute Phi_0
    double Phi_0 = 0.0;
    for (int i = 0; i < number_of_stops; i++) {

      Phi_0 += Stops[i].latitude;
    }
    Phi_0 = Phi_0 / ((double) number_of_stops);

    double cos_Phi_0 = cos(Phi_0);
    double x_mult = EARTHRADIUS * cos_Phi_0;
    double y_mult = EARTHRADIUS;

    for (int i = 0; i < number_of_stops; i++) {

      Stops[i].set_xyz(x_mult, y_mult, elevation);
    }
```

```
    // Build Origin-Destination Matrix M:

    //double M[number_of_stops][number_of_stops];
    double** M = new double* [number_of_stops];
    for (int i = 0; i < number_of_stops; i++) {
       M[i] = new double[number_of_stops];
    }

    for (int i = 0; i < number_of_stops; i++) {

      for (int j = 0; j < number_of_stops; j++) {

        double delta_x = fabs(Stops[i].x - Stops[j].x);
        double delta_y = fabs(Stops[i].y - Stops[j].y);
        double delta_z = fabs(Stops[i].z - Stops[j].z);

        M[i][j] = delta_x + delta_y + delta_z;
      }
    }

    double depot_x = depot_longitude * x_mult;
    double depot_y = depot_latitude * y_mult;
    double depot_z = elevation;

    // Compute the route:

    int visited[number_of_stops];
    for (int i = 0; i < number_of_stops; i++) {

      visited[i] = 0; // False
    }

    // Find first stop:

    int first_stop = 0;

    double shortest_distance =

      Stops[0].approximate_distance(depot_x,        depot_y,
depot_z);

    for (int i = 1; i < number_of_stops; i++) {

      double approx_distance =

        Stops[i].approximate_distance(depot_x,      depot_y,
depot_z);

      if (approx_distance <  shortest_distance) {

        first_stop = i;
        shortest_distance = approx_distance;
      }
    }

    // Okay, first_stop is our first node to visit

    int Route[number_of_stops];

    for (int i = 0; i < number_of_stops; i++) {

      Route[i] = 0;
    }

    Route[0] = first_stop;
    visited[first_stop] = 1; // True

    int previous_stop = first_stop;

    // Find the remaining stops:

    int first_j = 0;

    for (int i = 1; i < number_of_stops; i++) {

      int found_first = 0; // False

      while (!found_first) {

        if (!(visited[first_j])) found_first = 1; // True
        else first_j++;
      }

      double shortest_distance = M[previous_stop][first_j];

      int next_stop = first_j;

      for (int j = (first_j + 1); j < number_of_stops; j++) {

        if (!visited[j]) {

          double approx_distance = M[previous_stop][j];

          if (approx_distance < shortest_distance) {

            shortest_distance = approx_distance;
            next_stop = j;
          }
        }
      }

      Route[i] = next_stop;
      visited[next_stop] = 1; // True
      previous_stop = next_stop;

    }

    // Output route to standard output:

    std::cout                 <<                 "Visit
#,OBJECTID,Name,Duration,RouteName,Sequence,CurbAp
proach,Driver,POINT_X,POINT_Y,Zone2020" << std::endl;

    for (int i = 0; i < number_of_stops; i++) {
```

```
      std::cout << i << ',';
      Stops[Route[i]].output_row();
    }

    delete[] Stops;
    for (int i = 0; i < number_of_stops; i++) {
       delete[] M[i];
    }
    delete[] M;
  }
```

References

[1] Dantzig, G. B., & Ramser, J. H. (1959). The Truck Dispatching Problem. *Management Science*, *6* (1), 80–91. https://doi.org/10.1287/mnsc.6.1.80

[2] Figliozzi, M. (2010). Vehicle Routing Problem for Emissions Minimization. *Transportation Research Record: Journal of the Transportation Research Board*, *2197* (1), 1–7. https://doi.org/10.3141/2197-01

[3] Hargitai, H., Willner, K., & Hare, T. (2019). Fundamental Frameworks in Planetary Mapping: A Review. In H. Hargitai (Ed.), *Planetary Cartography and GIS* (pp. 75–101). Springer International Publishing. https://doi.org/10.1007/978-3-319-62849-3_4

[4] Karp, R. M. (1972). Reducibility among Combinatorial Problems. In R. E. Miller, J. W. Thatcher, & J. D. Bohlinger (Eds.), *Complexity of Computer Computations: Proceedings of a symposium on the Complexity of Computer Computations, held March 20–22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, and sponsored by the Office of Naval Research, Mathematics Program, IBM World Trade Corporation, and the IBM Research Mathematical Sciences Department* (pp. 85–103). Springer US. https://doi.org/10.1007/978-1-4684-2001-2_9

[5] Lenstra, J. K., & Kan, A. H. G. R. (1981). Complexity of vehicle routing and scheduling problems. *Networks*, *11* (2), 221–227. https://doi.org/10.1002/net.3230110211

[6] Li, S., Dragicevic, S., Castro, F. A., Sester, M., Winter, S., Coltekin, A., Pettit, C., Jiang, B., Haworth, J., Stein, A., & Cheng, T. (2016). Geospatial big data handling theory and methods: A review and research challenges. *ISPRS Journal of Photogrammetry and Remote Sensing*, *115*, 119–133. https://doi.org/10.1016/j.isprsjprs.2015.10.012

[7] Papadimitriou, C. H. (1977). The Euclidean travelling salesman problem is NP-complete. *Theoretical Computer Science*, *4* (3), 237–244. https://doi.org/10.1016/0304-3975(77)90012-3

[8] Riechel, J. (2019). A fast algorithm for computing approximate distances in the Cartesian plane. *Proceedings of URISA GIS-Pro '19, New Orleans, LA*. https://urisa.library.esri.com/cgi-bin/koha/opac-detail.pl?biblionumber=183148&query_desc=kw%2Cwrdl%3A%20riechel

[9] Riechel, J. (2020a). *Comparing Manhattan, Euclidean, and Actual Driving Distances*. Proceedings of CalGIS 2020, Long Beach, CA. https://drive.google.com/file/d/1_wgjePJH6LXM6OAYHg-PJkr2o-wVr8AK/view?usp=sharing

[10] Riechel, J. (2020b). *Extending Manhattan, Euclidean, and Actual Driving Distances into 3D*. Unpublished. https://drive.google.com/file/d/16rkw9Ysn8BWfT7CpfvlnlaSwUgBw4Y4v/view?usp=sharing

[11] Ripplinger, D. (1922). Rural School Vehicle Routing Problem. *Transportation Research Record*, 6.

[12] Rosenkrantz, D. J., Stearns, R. E., & Lewis, I., Philip M. (1977). An Analysis of Several Heuristics for the Traveling Salesman Problem. *SIAM Journal on Computing*, *6* (3), 563–581. https://doi.org/10.1137/0206041

[13] Singh, A., Yadav, A., Block, A. E., Rana, A., Block, E., & Floor, G. (2013). K-means with Three different Distance Metrics. *International Journal of Computer Applications*, *67* (10), 13–17. https://doi.org/doi:10.5120/11430-6785

[14] Tang, H., & Miller-Hooks, E. (1964). Interactive Heuristic for Practical Vehicle Routing Problem with Solution Shape Constraints. *Transportation Research Record*, 10.

[15] Wang, Y., Ma, X., Lao, Y., Wang, Y., & Mao, H. (2013). Vehicle Routing Problem: Simultaneous Deliveries and Pickups with Split Loads and Time Windows. *Transportation Research Record: Journal of the Transportation Research Board*, *2378* (1), 120–128. https://doi.org/10.3141/2378-13

[16] Zeager, J., & Stitz, C. (2016). *College Algebra*. http://dspace.calstate.edu/handle/10211.3/180387