Goods transportation problem solving via routing algorithm

Mikhail Shchukin, *CS 820 Graduate Student, University of Regina*[*]
Aymen Ben Said, *CS 820 Graduate Student, University of Regina,*[†]
and André Lobo Teixeira, *CS 820 Graduate Student, University of Regina*[‡]
Email: [*]vladimmi@uregina.ca, [†]abb549@uregina.ca, [‡]teixeira@cs.uregina.ca

Abstract

This project report outlines the ideas behind developing a graph-based heuristic-driven routing algorithm designed for a particular instance of a goods transportation problem with a single good type. The proposed algorithm solves the optimization problem of satisfying the demand of goods on a given undirected transportation graph with minimizing the estimated cost for each traversed segment of the delivery path. The operation of the routing algorithm is discussed and overall evaluation of the proposed problem solving technique is given.

Index Terms

Algorithm, graph, report, logistics.

I. Introduction

THE transportation problem is one of the well-known and hot topics both in mathematics and economics. It was first conceptualized by Gaspard Monge back in 1781 [1]. Since then, it was analyzed with various approaches, including linear programming and graph-based algorithms. In this project, a graph-based interpretation of the problem is used. This project strives to deal with this problem using graph theory approach and a particular decision-making algorithm in an attempt to acquire an acceptable solution, if it exists, or give partial solution otherwise.

From our point of view, the transportation problem can be represented by a graph with nodes of 3 types: *store*, *warehouse* and a *joint*. The edges between nodes represent roads to traverse with associated *distance* and *time*. The stores and warehouses have a certain associated amount of *demand* and *supply* of goods respectively. Joints act solely as transfer points (major cities, towns, etc.). For simplicity, all goods are assumed to be of the *single type*. The problem is finding the way to provide the stores with goods available in the warehouses, *maximizing* the amount of delivered goods, while *minimizing* the transportation costs. It is assumed that the shipping is being done by a *single agent*, a truck with a certain *maximum carry capacity*, that is traversing a certain path while loading and unloading goods where needed.

In this project, we provide an algorithm that takes a pre-generated transportation problem graph with the described characteristics and returns the following key elements of the proposed solution:

- Truck path with shipping and restocking segments
- Algorithm log with the following details:
  - shipping and restocking segments explained
  - routing decisions made for next segment (if applicable)
  - decision cost (heuristic value)
  - Remaining total amount of demand or supply at the termination of algorithm
- Total runtime of the algorithm

In the following sections we describe the testing environment for such algorithm, the mode of operation of such algorithm and the experimental results.

II. Testing environment - graph generator

First of all, in order to provide a capability of reproducing the experimental results of using the suggested algorithm, it is important to have a testing environment that can produce a set of various transportation problem instances (graphs) of a varying scale to serve as an input to the solving algorithm. The suggested testing environment attempts to pseudorandomly generate a transportation problem graph instance on each run. This allows to create a wide variety of sample cases that are applicable to the real-life instances of goods transportation problem.

The graph generator requires the following parameters to operate with:

- Total number of nodes to generate
- Maximum number of edges to spawn per node
- Total number of stores to generate

- Total number of warehouses to generate
- Types of goods to generate (in scope of this project, this defaults to 1)
- Total supply and total demand per each good type

Given these parameters, the graph generator proceeds to create a requested number of nodes. Each node is created as a joint by default. Then, the algorithm pseudorandomly decides which nodes are converted into stores and warehouses. To keep the graph realistic, each node receives pseudorandomly assigned coordinates of X,Y, where X,Y $\in$ [0, MAX_MAP_SIZE], where MAX_MAP_SIZE corresponds to the scale of the square space where the transportation problem is defined, it defaults to 1000. Originally, it is assumed that the units of map scale are expressed in *kilometers*.

The generator then distributes goods to be supplied one by one between each warehouse until all supplies are allocated. Similarly, each store keeps incrementing the amount of demanded goods one by one until the demand is equally distributed among all stores. Equal distribution of supply and demand among stores and warehouses is assumed to avoid degenerate cases to happen (i.e. most goods are concentrated in a warehouse too close to the stores, one store demands much more goods than the others, etc.).

After the node generation is done, the algorithm spreads the required amount of edges. Each time an edge is to be created, there is a 50% chance for that to happen. To avoid disconnected graphs, the algorithm forces at least one edge to be spread. The destination node is picked pseudorandomly among the rest of the generated nodes. When the edge is created, the X,Y coordinates of the source and destination are used to obtain the *distance between nodes*. Then, an *average assumed velocity* is picked pseudorandomly within the range of [40,100]. These bounds are assuming that the velocity is expressed in *kilometers per hour*, where lower bound corresponds to speed limits of bigger cities and upper bound resembles the highways. Based on the average assumed velocity of the edge, the time is then calculated as

$$time\_in\_minutes = (distance\_between\_nodes / assumed\_average\_velocity) \times 60$$

with floating point of it truncated. The time and distance are the two key parameters that affect the later decision-making, so they are kept in the graph as the background information.

When edges between nodes are created, the graph generator produces a simple text file, that has the problem-related information associated with nodes and edges. Notably, the syntax for such text file is specifically designed to follow the syntax of WebGraphViz, the online graph visualization tool that can produce and display the graph, based on the plain text graph description with all embedded information as a picture [2]. A particular example of such visualization can be seen on Fig. 1. The source file for that graph is available in Appendix A of this report.

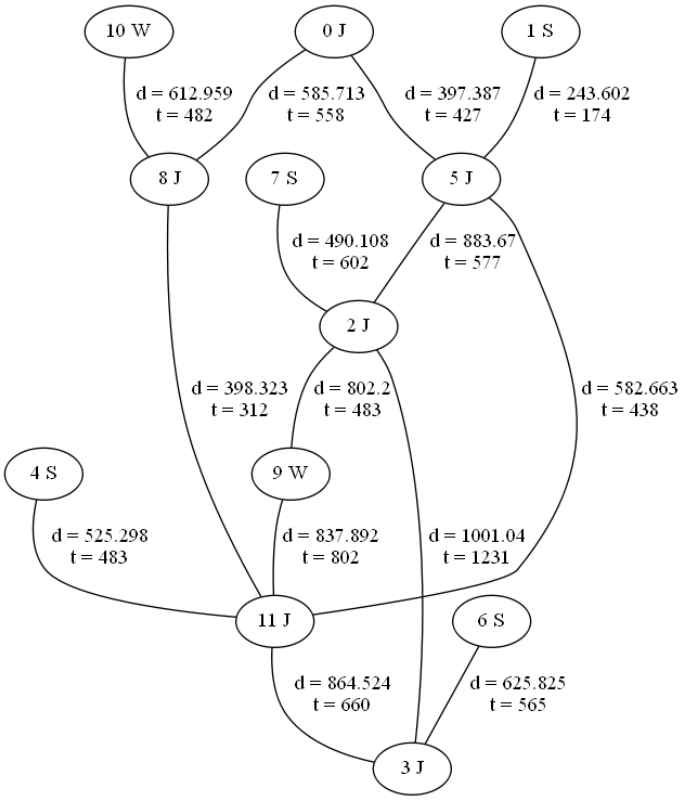

Fig. 1. That is our graph!

This testing environment combines the commodity of reproducable results, configurable problem scaling and the power of visualization through external tools. In the following sections, we refer to Fig. 1 as a particular case of using this testing environment.

III. The Routing Algorithm

Given an instance of the goods transportation graph, the routing algorithm requires the following parameters to attempt at solving the problem:

- Truck initial position (defaults to node 0, which is the first generated node)
- Truck initial load of supplies (defaults to 0, we assume here that the truck always starts empty)
- Truck maximum carry capacity
- Resupply threshold **T** (level of current supply below which a truck restocks at a warehouse, not proceeding to stores)

With these parameters set, the truck attempts to follow the least expensive path, which is recorded as a solution. The heuristic cost that is used in determination of the least expensive edge to traverse is defined as follows:

$$edge\_cost = time + distance$$

Accordingly, at the beginning of each segment of the traversed path the truck driver decision-making works as follows:

- If the position of the truck is at a warehouse node with a capacity $<$ **T** $\times$ MAX_CAPACITY : it goes to the nearest warehouse which is the current node and restocks right there.
- If the position of the truck is at a joint node with a capacity $<$ **T** $\times$ MAX_CAPACITY: it goes to the nearest warehouse to restock with goods.
- If the position of the truck is at a store node with a capacity $<$ **T** $\times$ MAX_CAPACITY: it goes to the nearest warehouse to restock with goods.
- If the position of the truck is at a warehouse node with a capacity $>=$ **T** $\times$ MAX_CAPACITY: it goes to the nearest store to drop goods.
- If the position of the truck is at a joint node with a capacity $>=$ **T** $\times$ MAX_CAPACITY: it goes to the nearest store to drop goods.
- If the position of the truck is at a store node with a capacity $>=$ **T** $\times$ MAX_CAPACITY: it goes and drops goods at the current node.

Note that for the sake of simplicity, we assume **T** = 0.5 for the particular instance of the graph shown in Fig. 1.

The algorithm treats each node according to their type with 2 parameters: supply and demand. For warehouse nodes, the demand is always 0; for stores, the supply is always 0; for joints, both supply and demand are 0.

The truck keeps going back and forth between the stores and the warehouses until either the warehouses run off of supply or the stores stops asking for goods.

IV. Conclusion

After observing the fundamenal principles of the operation of the proposed algorithm, it is important to say that it is capable of functioning with varieties of transportation problem instances. Consequently, with doing some experiments and applying different values to truck capacity we have compared the results and realized that the algorithm also helps companies by giving them an idea about the requirement of the truck capacity to perform better by fulfilling the entire demand and minimizing the overal path cost.

For our project, we could have done a better job in choosing the right parameters for the threshold value, because it is a very important variable that can make a change in the decision making, as a result affecting the subpaths and the overal trip that the truck makes while visiting stores and warehouses. Also, we used the summation of both time and distance in regards of distinguishing the better path which can still be limited and definately can be improved by either choosing a better relation between time and distance (not just simply summing them) or by picking more accurate constraints/heuristics.

Below is an experimental table for an instance of our problem where we each time changed the truck capacity value and stored the results:

Truck Capacity	Number of path segments	Total cost achieved
10	22 segments	68360.5
15	16 segments	48142.7
20	14 segments	42781.1
22	13 segments	40764.5
23	9 segments	26577.2

Appendix A
Source codes

Appendix one text goes here.

APPENDIX B
TOOLS USED

Appendix two text goes here.

REFERENCES

[1] H. Kopka and P. W. Daly, *A Guide to LaTeX*, 3rd ed. Harlow, England: Addison-Wesley, 1999.