Modeling High Volume Networks With Dynamically Assigned Node Structures

Introduction

- Current transportation networks utilize inflexible algorithms to determine how to route vehicles
  - This is why buses and trains operate on specific, rigid schedules
  - This is also why red lights may halt several cars, even when no cars are using the intersecting road
- Current algorithms cannot be used in networks such as automobile systems because
  - They fully define the path of the vehicle before runtime
  - They cannot adapt to changes in the network (drivers changing destination, or the addition or removal of roads)
- Dynamic algorithms, which route vehicles while the network is running, are needed for automobile systems.
- Dynamic algorithms can account for vehicles changing destination and roads being added or blocked, because they operate in runtime
- Dynamic algorithms would trade optimality of routes for computation speed and flexibility

Engineering Goals

1. Create a model of a network in Java using paths, nodes, and vehicles
2. Create a program which accepts an initial network and outputs network data
3. Create an algorithm which can schedule multiple vehicles to a single node without collision
4. Create an algorithm which can route a vehicle incrementally closer to a given node, using path weights given by the algorithm in goal 3
5. Using the algorithms created, use the program created in goal 2 to model a small network and route vehicles, displaying network data in text format
6. Create an animated model for the program
7. Extend the routing algorithm to create a partial route of multiple nodes
8. Determine the program's ability to respond to dynamic changes
9. Test the program's efficiency using several randomly generated networks

Program Overview

- The program is written in Java
- Networks are composed of **nodes, roads, and vehicles**
- Nodes serve as the intersection points of roads – All vehicle routing and scheduling is done at nodes
- Roads are unidirectional paths between two nodes
- Vehicles are points which travel on roads. They originate at a specific node and have a destination node.

- Low-level algorithms which control the path of vehicles are **schedule** and **route**, each of which are used when a vehicle enters a node
- The program runs each vehicle along roads and schedules vehicles at nodes until all vehicles are at their destinations

- Schedule is used to determine how long it will take for a vehicle to traverse a road, using the length of the road between the nodes, the maximum speed that the vehicle can travel on the road, and times during which the output node is occupied.
- Schedule returns the soonest time that the vehicle may reach the node

- Route determines all of the sequences of roads that consist of *cstep* road traversals from the current node. It then determines how long it would take to fully traverse each sequence, and determines the best sequence to take by comparing the distance travelled towards (absolute geographic position) the end destination and dividing this by the time it would take to traverse the sequence. It ignores routes that it has already taken.

- Dynamic changes during the "run" method allow vehicle rescheduling, and clear all pre-determined route data from nodes. This allows other vehicles to use the time slot otherwise used by the vehicle.

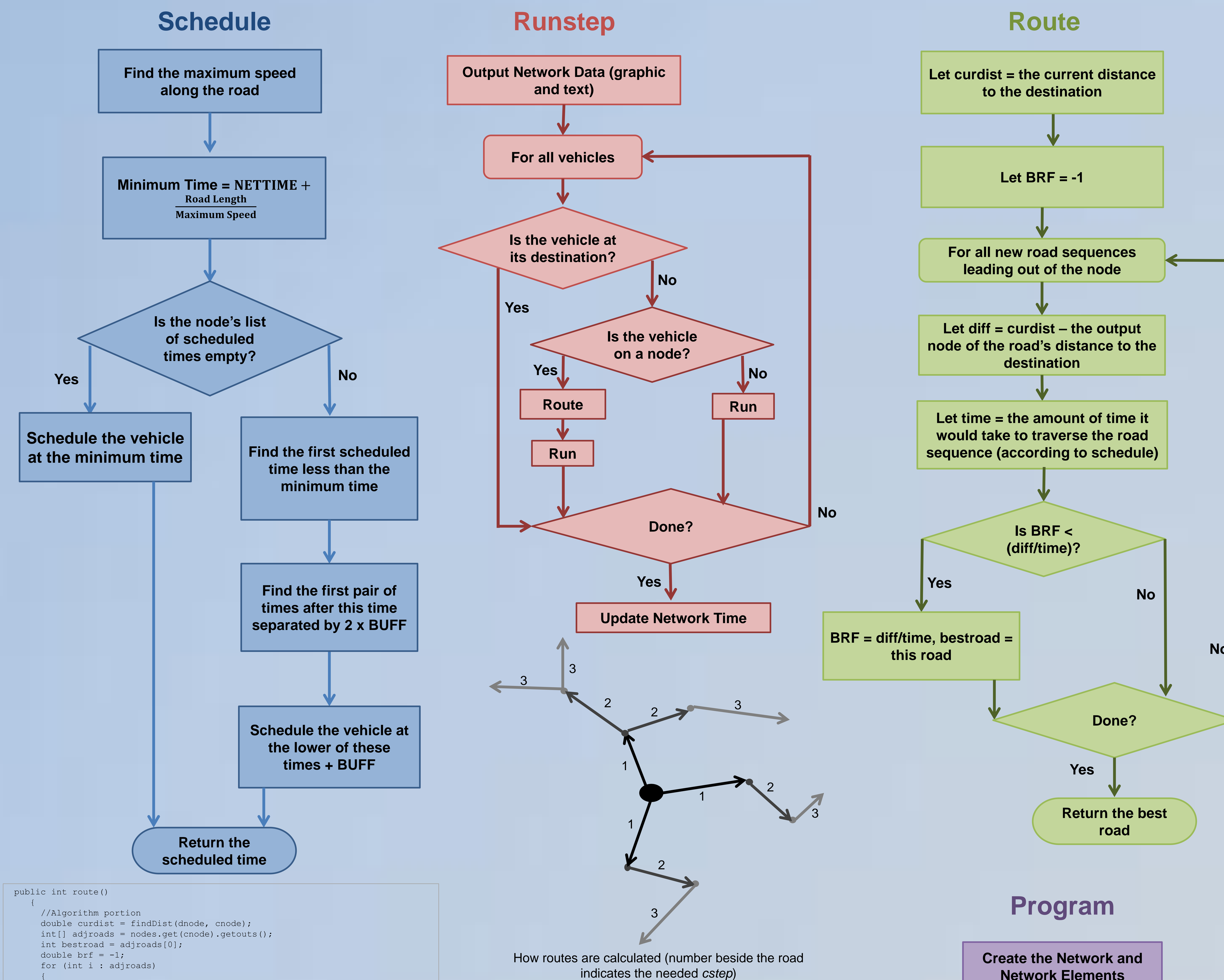
Results

- A program was created which can model networks with real-world constraints and dynamically route vehicles.
- The program was demonstrated as being capable of routing and running several large and complex networks.
- No collisions occurred on the network, as demonstrated by data from the program.
- Random networks were used for testing. With a cstep of at least 5, almost all networks could be routed successfully.
- The program responded successfully to dynamic changes in the destination of vehicles

Summary

Current traffic and rail networks depend largely on pre-defined routes which are optimal for given start conditions. As networks become more complex, and as network parameters are liable to change during runtime (rail breakdowns, wrecks, change of destination, etc.); future networks will need flexible algorithms which can adapt to changes in network parameters. This research attempts to address the flexibility problem of networks by demonstrating a network with dynamically routed and run packets, using a simple greedy algorithm to determine to which nodes to travel. The algorithm demonstrates capabilities to adapt each vehicle's route in accordance with traffic patterns, which static algorithms are incapable of doing. The algorithm is tested using a Java program which allows for network modeling and testing. Output from the network is represented textually and graphically, and demonstrates collision-free operation of the network.

Methods and Algorithms



How routes are calculated (number beside the road indicates the needed *cstep*)

```java
public int route()
    {
      //Algorithm portion
      double curdist = findDist(dnode, cnode);
      int[] adjroads = nodes.get(cnode).getouts();
      int bestroad = adjroads[0];
      double brf = -1;
      for (int i : adjroads)
      {
        double nextdist = findDist(dnode, roads.get(i).getbnode());
        double diff = curdist - nextdist;
        double time = schedule(id, i, false) - NETTIME;
        if (brf < (diff/time))
        {
          brf = diff/time;
          bestroad = i;
        }
      }
      //Non-Algorithm
      reachtime = schedule(id, bestroad, true);
      road = bestroad;
      nnode = roads.get(road).getbnode();
      cnode = -1;
      rate = roads.get(road).getlength() / (reachtime - NETTIME);
      ret public int route()
    {
      //Algorithm portion
      double curdist = findDist(dnode, cnode);
      buildroutes();
      int bestroute = 0;
      double brf = -1;
      int n = 0;
      for (ArrayList<Integer> i : myroutes)
      {
        double nextdist = findDist(dnode, roads.get(i.get(i.size() -
1)).getbnode());
        double diff = curdist - nextdist;
        double time = NETTIME;
        for (int k : i)
        {
          time = schedule(k, false, time);
        }
        time -=NETTIME;
        if (brf < (diff/time))
        {
          brf = diff/time;
          bestroute = n;
        }
        n++;
      }
      //Non-Algorithm
      double time = NETTIME;
      for (int k : myroutes.get(bestroute))
      {
        time = schedule(k, true, time);
        route.add(k);
        reachtimes.add(time);
      }
      reachtime = reachtimes.get(0);
      reachtimes.remove(0);
      road = route.get(0);
      route.remove(0);
      nnode = roads.get(road).getbnode();
      cnode = -1;
      rate = roads.get(road).getlength() / (reachtime - NETTIME);
      myroutes.clear();
      return road;
    }
```

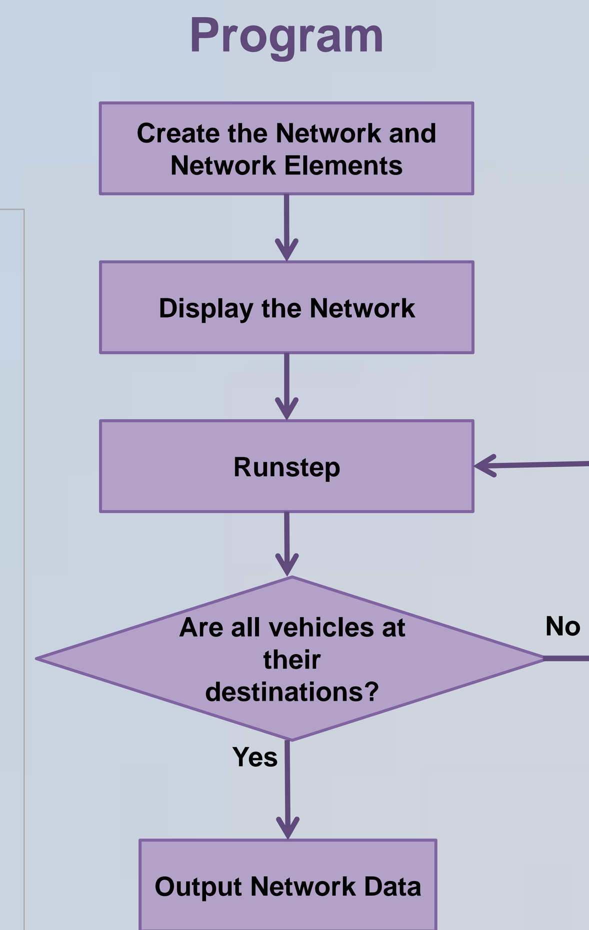
```java
public double schedule(double mintime, boolean cont, int vehicle)
    {
      double addtime = 0;
      int addindex = 0;
      if (times.size() == 0)
      {
        addtime = mintime;
        addindex = 0;
      }
      else
      {

        if (times.get(0) - BUFF >= mintime)
        {
          addtime = mintime;
          addindex = 0;
        }

        else
        {
          boolean done = false;
          for (int i = 0; i < times.size(); i++)
          {
            if (!done)
            {
              if (i<(times.size()-1))
              {
                if ((times.get(i) < mintime) && ((times.get(i+1) - times.get(i))
< 2*BUFF))
                {
                  addtime = Math.max(times.get(i) + BUFF, mintime);
                  addindex = i+1;
                  done = true;
                }
              }
              else
              {
                addtime = Math.max(times.get(i) + BUFF, mintime);
                addindex = i+1;
                done = true;
              }
            }
          }
        }
      }

      if (cont)
      {
        times.add(addindex, addtime);
        myvehicles.add(addindex, vehicle);
      }
      return addtime;
    }
```

Program



Algorithm Correctness and Complexity

As vehicles may temporarily occupy the same space on a road in real life applications (such as a car overtaking another car), collisions on roads were not taken into account in the analysis of algorithm correctness. Correctness was determined as representing no potential for collisions on nodes. The scheduling algorithm therefore must be correct, as a collision could only happen if two vehicles are scheduled for the same node simultaneously, which cannot occur because schedule() separates all vehicles by a time of BUFF, for BUFF > 0.

Complexity:

The average number of roads exiting any given node is

$$\frac{r}{n}$$

For a network with n nodes and r roads. Let c be the cstep of each car on the network, and v be the number of vehicles on the network. Thus, each vehicle must check an average of

$$\left(\frac{r}{n}\right)^c$$

Thus for the entire network, the amount of time, T that it takes a particular computer to determine how to route all of the vehicles is

$$v\left(\frac{r}{n}\right)^c$$

$$\frac{\partial T}{\partial c} = v\left(\frac{r}{n}\right)^c \cdot \ln(c)$$

This shows that algorithm complexity increases exponentially with cstep.

Program Limitations and Assumptions

- Routes are often non-optimal, as vehicles follow a greedy algorithm
- Starvation (a vehicle or group of vehicles being slowed down an unacceptable amount) may occur if a node receives a large amount of traffic from a different input road, causing the schedule algorithm to route these vehicles far later than their minimum time.
- The program assumes properly formatted input, and that all nodes and roads referenced when vehicles, nodes, and roads will be created
- In rare cases (closed loops), vehicles may be incapable of escaping a certain set of nodes. This is not a problem at higher cstep values

Future Work

- Implement a user interface which can accept dynamic changes to the network
- Create a random network generator which outputs more realistic networks than the truly random networks currently generated
- Statistically determine complexity and computation time variance with respect to each network parameter

Works Cited

"Java Platform SE 6." Oracle Documentation. Web. 25 Jan. 2012. <http://docs.oracle.com/javase/6/docs/api/index.>
O'Connor, Derek. "Derek O'Connor."Derek O'Connor. Web. 25 Jan. 2012. <http://www.derekroconnor.net/home/mms406.>
Lester, Patrick . "A* Pathfinding for Beginners." Almanac of Policy Issues. Web. 25 Jan. 2012. <http://www.policyalmanac.org/games/aStarTutorial.htm>
"Tree traversal - Rosetta Code."Rosetta Code. Web. 25 Jan. 2012.
" Tree and Graph Searches." Cross-Comp. Web. 25 Jan. 2012. <http://www.cross-comp.com/Pages/scipro/TreeSearch.aspx>.