Technical Report CS14-15

Optimizing Traffic Flow with a Genetic Algorithm

Adam Wechter

Submitted to the Faculty of
The Department of Computer Science

Project Director: Dr. Gregory Kapfhammer
Second Reader: Dr. Janyl Jumadinova

Allegheny College
2014

I hereby recognize and pledge to fulfill my responsibilities as defined in the Honor Code, and to maintain the integrity of both myself and the college community as a whole.

Adam Wechter

ADAM WECHTER. Optimizing Traffic Flow with a Genetic Algorithm. (Under the direction of Dr. Gregory Kapfhammer.)

ABSTRACT

Traffic gridlock and delay waste precious resources and is known to be worsening as populations increase. This makes alleviating the problem all the more important and drives research to find new, novel techniques to address the many causes of traffic congestion. This paper offers an evolutionary algorithmic approach to optimizing traffic control devices at intersections to feasibly and affordably reduce total time transit, total time delay, and total fuel consumption of the vehicles. Extensive configuration trials suggests that this process can contribute to optimize a given system, and points to future study and modification of these types of algorithms on larger and more realistic city plans. This research delivers a simple, ideal, discrete event simulator, which is able to solve for fitnesses of a traffic system with a user-defined traffic control system configuration, as well as the customized ECJ engine which solves for the highest fitness using the simulator as the evaluator.

I'd like to thank, firstly, my professors Kapfhammer, Jumidanova, Roos, and Cupper, for the help they provided me guiding and sparking the interest and passion I hold for programming today. My mother and father, for being there through the most rewardingly difficult years of my education, as well as my brothers for releasing the tension when it was needed most. If it weren't for my family, I would never have even made it to the comp so for their drive, I am grateful. I'd also like to thank my dear friends BJ Nelson, who was a constant pressure and egging on helped me preserveer, Katherine Murray, who's friendship and sense of humor got me through the days, and finally Bridgette McCauley, who always believed in me.

Contents

List of Figures

Chapter 1

Introduction

This chapter will overview the entirety of the project including motivation for research, thesis statement, goals of the project, contributions of this project, and finally a general outline to the format and construction of this paper. Primary and secondary sources are reserved for chapter 2 and will not be found in this section. Also, this segment will begin to introduce some of the language and definitions that the reader will find necessary to understand the entirety of this paper, however rarely used terms will be defined within their respective section if they are not commonly used.

1.1 Motivation

Traffic gridlock and delay waste precious natural, financial, and time resources, while contributing to local air pollution and global greenhouse gas accumulation. It is well established that this problem is worsening is well-established. Improved mass transit, increasingly fuel- efficient vehicles, and flexible work hours all have their places in addressing traffic gridlock and delay. Approaches to traffic intersection control itself are also being created, tested, evaluated, and implemented as potentially promising large-scale palliative solutions. The successful use of genetic algorithms in providing more optimal solutions to such problems as time tabling, scheduling, and other NP-Hard problems, and to some extent to the problems of traffic flow, supports expanding their use to further ameliorate traffic congestion. Traffic intersection control and genetic algorithms (GAs) are two subjects of deep personal interest to the author, and the notion of combining those interests to address a major problem motivates this research.

Of all the means of limiting the sequelae of traffic congestion, the most feasible and affordable in the short run is implementing traffic pattern changes through intersection control, using a system that minimizes investment in hardware and related equipment and their installation. Proposed systems that rely on universally installed internal automobile technology fail those criterion mentioned. Similarly, for reasons of cost alone, the ideal of restructuring an entire urban system and thus physically optimizing or overhauling the existing system is an essentially impossible one for existing cities.

Genetic algorithms (GA) are search heuristics which emulate natural selection and evolution to generate solutions to optimization or search problems; they are part of a larger collection of algorithms called evolutionary algorithms (EAs). Fundamental to these algorithms is their uses of genetic representations to describe solution domains, in this case binary arrays, as well as fitness functions that assign numeric representations to the quality of the performance of solution domains. Motivations for choosing this sort of algorithm for the process of optimizing traffic intersection configurations are threefold: first of all, GAs have been successfully implemented in the past for two specific types of problems with great success, namely time-tabling and scheduling. Both of these problems are known to be NP-Hard, which indicates that there are no known algorithms to effectively find solutions in polynomial time. Since traffic system optimization problems fall under similar constraints, genetic algorithms would seem to be a choice method for generating possible optimized configuration networks for traffic control. Secondly, the success of several other projects involving implementation of genetic algorithms encourages further exploration and study. Lastly, although many systems, such as the Sydney Coordinated Adaptive Traffic System (SCATS), work very well in real time, their development relies on initial layouts and timing configurations as standard starting points; determining those is already an expensive and time-consuming process. Using GA-determined configurations to establish the standard starting points may reduce the time and costs involved and thus encourage more widespread use of these expensive but proven systems.

1.2 Thesis Statement

Using a discrete event simulator as a testing and metric environment, it will be demonstrated that genetic algorithms are able to improve several benchmarking criteria, including decreasing time transit and time delay, while operating within the constraints of a realistic traffic experience.

1.3 Goals of the Project

This project has several goals primarily involving the creation of different components of the simulator and ECJ engine, followed by a series of objectives normally associated with empirical research. By running custom or randomly generated grid designs and gathering critical data, the simulator will enable determination of the fitness of the current traffic network setup, as well as gathering information about vital information for comparison with a previous iteration. Developing the simulator was the primary focus of this comprehensive project as subsequent phases depended on its functioning.

The next deliverable of this project involves the implementation and customization of the Java Evolutionary Computational genetic algorithm (ECJ). ECJ is a dynamic, open source, freeware algorithm, which has been proven to efficiently and effectively solve stochastic optimization problems, more specifically ones involving metaheuristics. Stochastic optimization is a class of techniques that use randomness to a degree in order to find the closest to optimal, and sometimes optimal, solution to NP-Hard problems. Metaheuristics are strategies used to guide a search process using heuristics to find a good enough solution to some sort of problem. Since metaheuristics are not problem specific, this project utilizes the framework provided by ECJ, along with the highly documented instruction manual to develop the algorithm described later in chapter 3[5][6].

Only after the completion of these major components can the final objective of this project be attempted: to execute the program in a series of tests and analyze the results from different scenarios. The downside to GAs is that "better" solutions are only in the realm of other solutions. This being said, GAs are generally compared to different optimization algorithms for analysis, such as a completely random solution-generating algorithm that essentially brute forces a solution, saving the best found every iteration. By comparing how the GA performs to some other algorithm, the reader will be able to easily judge its performance as a traffic control configuration optimizer.

1.4 Contributions

The contributions of this senior comprehensive project include several promising new advancements to traffic systems optimizations. Firstly, this paper will demonstrate through its results that the implemented genetic algorithm has, to a degree, successfully improved and optimized a idealized grid. Additionally, there will be the beginning and basic framework of a traffic grid simulator which will allow for the testing and optimization of future traffic systems. Finally, this paper will suggest improvements that future work could make to this study for a more successful configuration optimization.

1.5 Thesis Outline

Chapter 2 reviews a number of past works in traffic control optimization and addresses their strengths and weaknesses. It also covers basic information on genetic algorithms. Chapter 3 outlines the method of approach used to establish the results including details on the simulator and ECJ configuration. This chapter also discusses threats to validity and details pertaining to the techniques used for testing. Chapter 4 contains details as to the results of testing and draws conclusions based on those results. Chapter 5 discusses the possibility of future work and reviews some of the different aspects covered in this project.

Chapter 2

Related Work

2.1 Secondary Sources

2.1.1 SCATS

The Sydney Coordinated Adaptive Traffic System (SCATS) is one of the most commonly used traffic monitoring and regulating systems used today and has been deployed in 154 cities across 25 different countries, though under different names. It was developed in late 1979 by A.G. Sims and K. W. Dobinson and was used primarily on major arteries in urban environments. It is a dynamic on-line and real-time system which manages the timing of signal phases of different intersections to find the best fitting solution for the current traffic situation. It also allows and incorporates real-time pedestrian traffic flows, public vehicle transportation, as well as public vehicle priority systems. The priority system is generally associated with the tram and bus system where there are high, medium, and low priority levels, with low priority vehicles waiting their turn, while high priority vehicles skip all other phases to get to their call locations. The system supports three levels of control as well, a local level, a master level, and the control center. The control center is the central computer hub, which supplies all area-based traffic control and urban based traffic control(UTC) with information it keeps in its logs, and allows the entire system to be monitored from a central location with a holistic view of the system. The master level is one level below that and is in charge of regional areas; it has a small amount of local storage for data as well as a control system and a logging system. Finally, the lowest level of control is given to the local controller, which consists of a road side box at each intersection. These boxes contain the hardware to control the nearby intersections and allow for emergency updates as well as immediate control over particular intersections. All of these different sections of the system are in constant communication with each other through the telephone lines, resulting in a robustly dynamic system[8].

The drawback to this system is that it requires the purchase and installation of a large amount of hardware for the system to be implemented. Back in 1980, when the paper was written, the cost of installing a system, not including the additional costs associated with rerouting traffic, etc., averaged 2.5 million dollars. Additionally this system has an associated monthly cost for the use of the telephone wires, which the researchers estimated would be about 220 thousand dollars a year. Further, this system cannot be implemented immediately due to the fact that the city must take the time to install it[8].

This system boasts a 20 percent reduction in accidents and 20-48 percent reduction in delays, leading to an estimated 10 million dollars in fuel savings, and an 18 percent reduction in carbon monoxide(CO) emissions. One of the final benefits that these researchers saw in their implementation was related to directing the preferred traffic flow of commuters. They argued that by reducing delays and traffic congestion on the main arteries of routes into and out of popular areas, commuters were more likely to take the intended routes. This in turn caused a reduction of impatient commuters attempting to cut through neighborhood areas in order to escape the delays associated with the main routes [8].

2.1.2 Vehicle-to-Vehicle

Another recent topic of research in the field is the change from physical control devices to virtual, in-car devices, relying entirely upon the surrounding vehicles to gather data about the state of the intersection and to make timing decisions. Research was presented in 2010 by Ferreira, et al.[4] as a solution to overcome several problems which other traffic control systems have, such as the SCATS traffic system. The researchers cite that the motivation behind this study lies within the problems that traditional traffic methods have, and introduce through its simplicity of sensor systems. They say that the sensors in these physical systems, allow the controlling computers to make intelligent decisions based on their observations, however introduce the problem naivete of the rest of the environment, which can lead to possible large miscalculations of the actual state of the intersection. For example, if the sensors which decide the length of the queue are only 200 feet back from the light, if a car is on that sensor, there is a chance that the queue goes beyond that distance and extends for a significant distance[4].

To remedy this, the authors propose a traffic control system that is completely encapsulated within the car, from data collection, to decision making, and control signal indication, acting as a completely self-organized system. Ferreira et al. make the assumptions that firstly, all of the vehicles in the system are equipped with a dedicated short range communication (DSRC) device, tuned to a specific channel so that the cars will be able to find each other, the cars are equipped with an up to date roadmap and accurate global positioning system (GPS), and some sort of dedicated Application Unit (AU). The step process that this system proposes to implement involves firstly, establishing a Virtual Traffic Light (VTL) protocol with all the surrounding cars approaching the intersection to indicate that a traffic signal will be used at the upcoming intersection. Following this step, the vehicles approaching the intersection are responsible for electing a leader node, which will be responsible for creating the VTL and broadcast the traffic light message to all surrounding node, as well as be responsible to serve as the primary form of infrastructure for that instance of the intersection. This leader is elected using a predetermined algorithm so that there is no conflict; however, two rules apply to this leader node. First, it must be stopped at the intersection and remain there while it is leading that intersection, and second, it must be the closest to the center, or front, of the queue in its lane. This second criteria is strictly there to optimize communication. Following this, the leader will broadcast the lighting signals to all nearby vehicles on an individual basis, which is why an accurate GPS is required in order to ensure that vehicles are correctly interpreted as to lane location. The researchers conducted several simulations based on the major city of Porto, Portugal, and concluded that there was a significant benefit and case for viability[4]. While this system is a fascinating one, the researchers also made it clear that this system should primarily be used where traffic lights are not necessarily required, and that it should supplement the existing system. Additionally, this research presumed that all of the vehicles in the environment are outfitted with specific hardware to allow this sort of communication to work. Even if this were to be the protocol for a country, there are still several issues that could arise. For example, if some country in Europe were to adopt this system and implement it completely as a substitute for the physical system, if any other vehicle from some outside country were to come into the nation, there is a chance that their car may not have the hardware to support it. This could result in car accidents and other forms of delay for the entire system.

2.1.3 Vehicle-to-Infrastructure

In the case that a city does not wish to completely replace its existing system, there are cases where the idea of a real-time, intelligent system which uses information it gathers from nearby vehicles, instead of the environment like the SCATS case, to allow a traffic controller to make enlightened decisions about the signal selections. Vehicular Ad-Hoc Network (VANET) is a very commonly explored technology which is used to collect and interpret information about nearby cars' speeds and positions. Avzekar and Moon, 2014[1], propose a system which will use this VANET technology to interpret and gather information about the density of traffic approaching an intersection, and to make intelligent decisions about regulating this traffic; while favoring emergency vehicles, allowing them to run lights or giving them direct routes to their destinations. The premise of this system is encapsulated in 4.1, which illustrates exactly how the system will work. Note that in the diagram below, the RF receiver stands for a radio frequency receiver. The first methodology proposed by these researchers deals with optimizing traffic light timing based on the volume and density of the incoming traffic. Using VANET, they make the case that they will be able to manage the system and to allocate greater time to higher flow streets at an intersection and significantly less time to lower density streets. This is logical and generally the goal of any optimization of traffic control system. The second goal of their implementation is one of greater interest, however, and deals with the priority system for emergency vehicles. They propose that a city's emergency vehicles be outfitted with radio frequency transmitters to communicate with the traffic light controller as needed. In this situation, all other nearby lights besides those in the emergency vehicle's lane will be immediately turned red until that ambulance, fire truck, or police car passes, after which the lights will resume their regular operation. Using simulations, Avzekar and Moon were able to see that this approach to solving the traffic problem had no negative

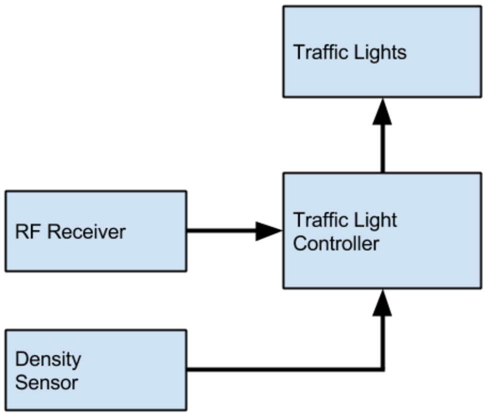

Figure 2.1: Block diagram of adaptive traffic system equipped with VANET

effects and resulted in consistent, smooth traffic flow. However, by failing to make a cost-benefit of the VANET system, they render their research slightly inconclusive[1]. There are problems with this system that are not make mention, yet are definitely present. First, VANET once again requires that *all* vehicles in a system have certain pieces of hardware installed in order for the system to work. Second, since security measures were not addressed, there is a chance that the radio frequency waves could be vulnerable to attacks.

2.1.4 Reinforced-Learning

Although the previous examples of traffic optimization covered methods which used sensors and real time techniques in order to adequately manage systems, there are several ways in which a traffic system can be optimized through the use of algorithms.

One paper that showed promising signs of ways to optimize this system used collaborative reinforced-learning techniques in order to support many of the unpredictable aspects associated with urban UTC. Salkham et al., 2008, propose using a collaborative reinforced learning system for the optimization process for several reasons, the most important being that reinforced learning is a great approach in unstable conditions where the flux of these systems are inevitable. Further, using the collaborative method over the traditional method allows for the implementation to affect the decentralized aspect of traffic optimization. Through a comparative study, they were able to show that both the traditionally reinforced learning technique and the collaborative reinforced learning technique reduce average wait time per arrived vehicle; however, they noticed that the collaborative technique was able to improve the overall quality of the optimization. Unfortunately, this study was done on a microscopic scale with regard to the simulation and detailed vehicles' behavior in a real city. The microscopic aspect of the simulator signifies that it tracks vehicles moving around the network on a second or sub-second basis, making it very accurate after the initial "warm-up" period. Thus the study's are very promising and might, in fact, translate well to real world application[7].

2.2 Primary Sources

Ceylan and Bell, 2004[2], cover two problems in one study: area traffic control optimization and user routing. They present the problem as an equilibrium network design problem rather than as a scheduling problem. Additionally, they call a known simulator named TRANSYT-7F to help evaluate the light timings of their genetically modified system. The network used for this simulation is small having only six control junctures, or intersections. Still, their findings are promising, suggesting that a genetic optimization can solve this type of problem. They arrive at three major conclusions: first, that this process shows an improvement of around 34 percent by the 75th generation; second, that the solution is strongly affected by the initial set of timings; and third, that depending on the model, there can be a huge disparity in total optimization time, from 16.5 seconds to 18.4 hours[2]. The implication of these conclusions for the research project at hand is that flow patterns can be repeated multiple times with different initial traffic timings. The traffic flow pattern the utilized were static, and tests were run multiple times with random starting traffic timing patterns determined by an algorithm. This provided sufficient results to determine whether an optimization process failure was due to the traffic flow set-up, or by the initial starting values of that optimization run.

Another example in which GAs not only worked but outperformed other stochastic algorithms is documented by Yun and Park, 2006[10], in a paper on the application of optimization methods applied to an urban corridor. They survey two existing optimization models for traffic timing, SYNCHRO(Husch and Albeck 2004) and CORISM. CORISM is the most recent addition to the TRANSYT-7F model, which uses a microscopic model and includes three different heuristics: a GA, simulated annealing, and an optimization engine called OptQuest. In their conclusion, the authors highlight that the recently introduced GA aspect of CORISM outperformed every other optimization method to which it was compared. These include the other heuristics of CORISM, as well as the commonly used SYNCHRO. While discussing future works, they caution future researchers that the performance of GAs is closely correlated to the original parameter settings. Further, they warn that running these optimizations can be time intensive. By showing that the GA outperformed the other heuristics by a rather large margin, around 15.2 percent in their measures of effectiveness, they influenced this researcher to focus on the implementations of a genetic algorithm rather than some other method[10].

While the time and scope of this research paper do not permit the use of a multi-objective GA, there have been occasions where this method has been applied to traffic control configuration problems with some success. One such example, completed by Li, Tao, and Chen, 2013[9], focuses on implementing a multi-objective GA on an oversaturated intersection. The environment scale varied between one- and two-intersection environments. Their study was motivated by the observation that many modern traffic control systems lack the ability to efficiently deal with circumstances where the amount of congestion overflows the normal boundaries. The variables they identify include queue length, delay, cycle length, and others. They determine that the number of variables, deems a multi-objective GA called NSGA-II to be the most appropriate approach. Their study emphasizes the importance of having accurate and realistic simulators. In their research, the authors were able to imitate several of the technologies available in real traffic systems, such as a NEMA controller. In their conclusion they acknowledge the difficulties inherent in increasing the number of genes that the algorithm needs to consider; increasing the number increases the computational complexity of the problem and the associated time demands[9]. Additionally this study had no more than two intersections in any simulation which is incredibly small and leaves information wanting, for example, how oversaturated intersections in a UTC would be optimized by a genetic algorithm.

Chapter 3

Method of Approach

This chapter will discuss and present an overview of the programs used for this study. It includes a detailed summary of the discrete event simulator, as well as descriptions of the ECJ tool used including parameter configurations. It will also remark on details on the logic of why certain algorithms were used for different mechanisms as well as architectural decisions of the program.

3.1 Test Environment

This project has two major sub-projects comprising the performance testing environment, the simulator and the ECJ evolutionary algorithm suite. The simulator has the sole purpose of receiving configurations of the light system, running a simulation based on a pre-determined traffic map, and evaluating the performance of an iteration and generating a correlating fitness value. The ECJ package is in charge of the optimization of the overall system; it utilizes the fitness values to best judge which configurations are superior.

3.1.1 Simulator

To begin with, this simulator is a discrete event simulator which takes four files and three "modifier values" as arguments and returns a fitness value based on several outcomes of this system, namely total time spent not moving at lights (time delay), total transit time, and approximate fuel used to travel through the system. The simulator is an idealized environment in which accidents do not happen and cars react promptly to external environmental changes. For example the cars react immediately to changes in traffic light color. Furthermore, the entire system is within a perfect square grid; each road has the same length and the same speed limit. Lastly, in this idealized system each road has two lanes, one in each direction. This places constraints on cars being able to turn left if traffic flow is heavy. The cars, as mentioned earlier, are relatively simplistic and all act the same way within the system. Though variations of the different cars' rates of fuel consumption are possible, that option is not utilized. The simplicity of this system is a result of time constraints and scale of scope; had more time and computing power been allocated, layers of complexity could have been introduced for better approximations of distances, staggered introductions of cars into the system, and non-grid patterned traffic maps.

As is, the simulator works in a way that mimics traffic rules and etiquette and uses seconds to mark the passing events. Since this system is a discrete event simulation, using seconds to mark the passage of time allows for comparisons to be made in an accurate manner and allows easy access to the state of the system at any point that an action is performed. Cars enter the system at the same time at the very beginning of the simulation, and after the specialized first move, which differs from all following actions for reasons explained later, the event lists are iterated through. The car within the event then makes a decision regarding its movement then returns to the next list if it has not reached its final destination. Following each iteration of car events, an iteration of node events is executed to update intersection states. Time in the system is held in the individual cars as well as in the nodes, allowing for easy manipulation of the state of the vehicle or the intersection lights. This is because after a while, each intersection is on its own schedule. The pseudocode shown in 3.1 is the basic process that the simulation follows by individually going through the car events and returning the events to the queue until each car reaches its destination and is removed from the queue. Note that after the queue of cars is looped through, there is a secondary cleaning loop which resets the state of the car so that it can go again in the next rotation.

```
Input: Queue $CE \Longleftarrow Cars$
Input: Queue $NE \Longleftarrow Nodes$
Result: Fitness of intersection configuration
while $CE$ not empty do
    for $ce \in CE$ do
        if $c \in ce$ canGo then
            if $c$ at_Destination then
                remove from system;
            else
                if can_it_move then
                    try to move;
                else
                    increment time values;
                end
            end
        end
        add $ce$ to $CE$;
    end
    for $ce \in CE$ do
        set canGo true;
    end
    for $ne \in NE$ do
        check_and_transition lights;
    end
end
```

Figure 3.1: Algorithm describing simulation process

Starting at the beginning, there are three files that are not reliant on the ECJ engine, unlike the configuration file. These three files are the `node.txt` file, `edge.txt` file, and the `car.txt` file, which are comma-separated values to hold details about other aspects of the simulator. The `node.txt` file contains information about the different nodes in the system. The nodes are the heart of the simulator and at any given point, every “car” object in the system will be in one. Nodes represent both intersections and roads connecting these intersections. The `node.txt` file first supplies the overall dimensions of the grid, as well as the length of the roads in the grid, in feet. The following lines of information provide the primary node data including whether the node is an intersection, in boolean format; the node ID, in form of `IorRxRowxColumn`, where the first value indicates whether it is a road or intersection while the other values represent a coordinate; speed limit, in miles per hour; and an indicator as to intersection vs road, including type of road. These final two merit further detail to describe what logic is used in the decision of their final forms. The speed limit is an important aspect to measure time within the roads, however all intersections have a speed limit of 0 miles per hour. This is because a car’s location in the intersection is determined solely by other cars in the intersection and since real world intersections are often inconsistent in speed. The final aspect of the nodes worth noting is the labeling of roads and intersections. In order to easily determine lanes and the appropriate queues that cars enter when changing nodes, using the labels “Lateral Roads” and “Vertical Roads” made it significantly simpler than checking parts of the node IDs each time.

The second file used was the `edge.txt`, which simply represents the connections that are drawn between the different nodes. The file lists each connection once; however, when the information is brought into the system, the connections are doubled so that there is a link from node a to node b and vice versa. These are primarily used when the cars decide directions. After accessing these two files, the simulator can build a grid in which cars will be added. Nodes are outfitted with directed queues to imitate cars moving in lanes. These directed queues act as lanes; for example, North directional queues have cars moving from South to North. While intersections have all four of these, the roads have only the two associated with their orientation, where lateral roads have the East and West queues and vertical roads have North and South queues. Following the construction of the map, cars are added to the system. The `car.txt` file contains first, the node ID of the car's starting position, as well as the node ID of the car's destination. It is important to note that cars cannot start and end at the same node as that would lead to an infinite fitness value since the time of transit would be divided by a distance of zero. The start and destination nodes are followed by the green-house gas emissions numbers, which are available on EPA's website. This number, while available, is not utilized in this study; it is directly correlated to time within the system. The next input data source is the cars' IDs, which are used primarily for debugging and quality control checks. The final information is the cars' fuel consumption, in miles per gallon. All of the variables that describe characteristics of the cars are used solely for calculation of the system's fitness after the simulation.

The final file that is used during the initiation process is the intersection configuration file. This is read in from a comma-separated file format and is such that each line is a new intersection. The intersections' configurations are in order and after the values are read in, the simulator iterates through the nodes and assigns each intersection the set-up determined by the ECJ engine. All values of these chains are in binary, and the first digit indicates whether the intersection has a light or stop signs at the corners. If the first value is a 0, then the intersection has stop signs and the following 7 bits are of no consequence to the activity and behavior of the node.

However if the first value is a 1, then there is a light at the intersection and the next three bits indicate the length of the North- South direction green light time. If the decimal value of this number is 0, then that light direction is automatically assigned the green light duration time of 15 seconds. If it is not 0, then the binary number is then multiplied by 15 seconds and is added to the initial 15 seconds value that is associated with a 0 value. The following three bits are also associated with the green light time, however, they refer to the East-West directional light. The same rules as mentioned before apply to this value. The final bit is used to indicate which direction has a green first, specifically, the North-South direction, indicated as true by a 1 and false otherwise.

Once all of these outside values have been entered into the system, the car objects generate directions from their start nodes to their destination nodes. This directions generation is provided by a breadth-first search algorithm which aims to create the route with the shortest path, regardless of the speed limit or any other factors besides distance. Once this is done, the system is ready to begin and the cars are all individually loaded into separate car events and the nodes into the node events. The first iteration of all of these events is more of a preliminary phase where cars are actually placed in the appropriate queues and the lights can begin rotating. All cars immediately jump from their start nodes to their second nodes so there will be no problems in the future with directional queue decisions. Additionally the internal car time variables are incremented as they normally would be. The decision making behind updating which directional queue that a car belongs in requires the knowledge of the previous node. While this could have been randomly assigned for the first iteration, some minor difficulties were averted by jump starting the system. Once this happens, the car events are executed as they would normally be. The first rotation for node events acts as it will for the rest of the simulation, incrementing the time variable as normal.

The car events execute in a simple way outlined by the pseudo code below. It begins by checking if the car is in the system, and if it is, the simulator checks immediately if it is at its destination. If the car proves to be at its destination, it is removed from the system and the details of the car's trip are recorded into the record data structures so they can be recalled later for calculation of the fitness value. If the car is not at its destination, then it decides how to proceed, whether decisions should be made as if it is on a road, at an intersection that has a light, or an intersection that has a stop sign. The simulator uses the time after the car leaves the system to do some cleaning of the queues to ensure no null values are displacing the lists, then replaces the car event back in the simulator to repeat these steps. If the car is flagged as not being in the system, it will be removed from the list of car events thus reducing the size of the list until it is empty, marking the termination of the simulation.

```
Input: Car Event $CE$
Result: Simulator with additional Car Event
if Car is in system then
    if car at_destination then
        Exit system;
    end
    else if Car is_on_road then
        Do road operations;
    end
    if Car is_at_light_intersection then
        Do light operations;
    end
    Do sign updates;
    Re-queue car event to simulator;
end
Do nothing, do not re-queue;
```

Figure 3.2: Algorithm describes car event logic

Light intersections and stop sign intersections work in different ways, one requiring information from the node, the other requiring information solely from the other queues at the intersection and the cars at the front of these “lanes”. The intersection this paper will review is the light intersection and is demonstrated in 3.3. The first principle is that lights have three states: green, red, or transition. These first two are rather self-explanatory, where cars can go on green and must stop on red. The transition period follows only after a green light and provides the opportunity for cars which wish to turn left, but did not have the opportunity during its green light because of traffic, to make that turn. When a car arrives at the intersection, the first step it takes is to check whether or not it is allowed to move. This is because in a situation where two cars at a green light, `car a` and `car b` where `car a` is at the front of the queue and `car b` is behind, if `car a` event is selected first and makes its movement into the next road. When `car b` is selected to make its decision and the car object decides to make a move because it is at the front of the queue and the light is green, there would be a conflict because two actions would be performed at the same time which in reality would cause a collision. To prevent this conflict, each car has a boolean `can go` variable. After confirming that the selected car is available for movement, the car checks the light signal and its logic branches. If the appropriate directional lights are green, one of three possibilities will occur. If the car is not at the front of the queue it will make no movement. If it is at the front of the queue, it will proceed straight or right if either is its direction, or it will first check the oncoming lane and proceed left only if that queue is empty. If the light is in a transition period, only if a car is at the front of the queue will it have the option of moving. If that lead car has a left turn to make, that left is given priority. If the lead car has a right turn, it will make it on the condition that the other lane across from it is empty. After movement decisions are made, it is important to note that all of the other cars that are in the queue are immediately flagged as not allowed to go.

```
Data: Car $C$
begin
    if Car can_go then
        if Light is_green then
            if Car not_at_front_of_queue then
                Do nothing;
            else
                if Car going_straight_or_right then
                    Update car info;
                else
                    if Opposite queue is empty then
                        Update car info;
                    else
                        Do nothing;
                    end
                end
            end
        end
        else if Light is_transition then
            if Car moving_left then
                Update car info;
            else
                if Opposite lane Is_empty then
                    Update car info;
                else
                    Do nothing;
                end
            end
        end
        else
            Do Nothing;
        end
    end
end
```

Figure 3.3: Algorithm describes light intersection logic

Sign intersections are slightly more complicated logically since in reality, various factors will often lead to behaviors that differ from that strictly dictated by law. The simulation breaks down the intersection in a way that closely mimics the way an intersection of stop signs would work. The premise of the stop sign is that a car must come to a complete stop, and if it is not there first, must yield to any oncoming or conflicting traffic. This operation begins in the same way as the light intersection, with checks to see if the car is legally able to move or not and to see if the car is in the front of the queue. If both are true, then after setting the rest of the cars in the queue to not being able to move, the car checks to see how long it has been at the front of the queue and compares that time to the lead cars in different lanes at the intersection. If the time is the longest, or if that time exceeds four seconds, then that car is allowed to move on with priority over the others. The reason the four-second rule is applied is that if the car has been there for four seconds, all other cars around the intersection have been rotated through so no matter what, this car must be next to move. Primarily this rule is used to save time in calculating the times of the other cars around the intersection. If several cars are there and several have the same time at front stamps, then a random roll is done to decide whether or not the car in question is to go first. If it can move first, then it is automatically allowed to move, and all other cars at the front of their lanes must wait unless their move does not conflict with the car-in-question's movement. If that is the case, rather than updating the other non-conflicting cars' "can go?" flag to false, it is ignored so that it can go during its turn while the others are flagged. This process can get rather complicated when more lanes have cars but the general premise is that longest there goes first.

Although it has been mentioned several times, how cars are able to update and what happens when the car is determined to do nothing has not yet been thoroughly

```
Data: Car $C$
begin
    if Car Can_go then
        Get List: thereLongest;
        if c there_longest then
            if thereLongest ¿ 1 then
                if Roll to go == true then
                    Update car info;
                    check for any non conflicting;
                    set rest to no_go;
                else
                    Do nothing;
                end
            end
        else
            Do nothing;
        end
    else
        Do nothing;
    end
end
```

Figure 3.4: Car sign intersection logic

explained. These two operations are relatively simple, however, involving primarily incrementation of different timers, updating location information, and cleaning any information from the current car event operations. Starting with the `doNothing` operation, the primary goal is to increment all timers. This includes updating the time it has been at the front if it is already there, the time in its current node/queue, updating its total time in the system, and finally, if it is waiting to move through an intersection, then incrementing the delay timer as well. The `updateCar` method is slightly more intricate primarily because when a car's info is updated, it is moving from one node to the next node in its directions list. This process calls methods to first, set the previous node to the node it is currently in, then set the current node to the node that is considered next at the time, and finally to update its next node, which pulls from the direction list to actually increment through the directions assigned at its instantiation. In addition to these important updates, information such as time in the current queue/node and time at the front of the current queue/node are cleared for the upcoming node. Finally information that is not related to being in any particular node or queue is also updated and incremented.

Node events are far more simple than the car events. The only actions that must be done in a node is either simply the incrementation of an internal node timer, which exists to keep track of when light patterns will change, or the actual change of light states. Both of these are handled within the event and the only necessary explanation is that when the green time of a direction reaches the time dictated by the configuration file, the state is changed to the transition state. Once in this state, it will wait one second in the simulation time, or a time that is specified by the source code pre-compilation, before switching. Finally it will then switch to the red light state and repeat the process. Unlike the car events, these events are never removed from the system and each node is updated every internal tick of the simulator.

The final aspect of this simulator that bears discussion is the fitness function. The fitness function is one that is tuned to find the fitness according to multiple objectives, chiefly reducing the time a car spends in the system. To weigh all of these values properly, the simulator takes the distance that they traveled and divides the time variables by it. When a car exits the system, the car's delay time, total time, and gas consumption is recorded into a list. The list is then iterated through and each car gets each of those numbers, divided by that car's total distance travelled, then subtracted from the existing fitness. The reason that the system subtracts these values from the fitness is that the optimizer strives to maximize the fitness value up to and including infinity and negative infinity. Due to this behavior characteristic, since all recorded values are technically values that are unwanted, it is easier to flip the sign to "maximize" this function than to minimize these values. This value is in the form of a float because ECJ requires a float while evaluating genomes, as a fitness.

3.1.2 Java-based Evolutionary Computation Research System (ECJ)

The Java-based Evolutionary Computation Research System, or ECJ system, is a java written, run-time determined system focussed on evolutionary algorithms. Using parameter files, this system allows problems to be solved that the users design, through a number of different algorithms, though for this study, only a genetic algorithm was used based around the simple generational evolutionary algorithm. The basis of this algorithm is from the `SimpleEvolutionState` which defines the the actual process that is used while solving the problem. ECJ requires users to define a parameters file, an evaluate method for the problem, and a problem. In this situation, the problem was the simulator so after defining the other aspects of the ECJ engine, an ECJ plug was developed that allowed interaction between the the evaluate method and the remainder of the problem.[5]

As shown in 3.5, the configuration for this process is relatively simple and did not take much work other than creating the appropriate parameters file. When the main function is called within Evolve, ECJ requires that a parameters file be provided to supply the necessary information that is needed to call the correct classes and provide the necessary information as a global variable file. The config file below is the one that is primarily used throughout the study, besides a few variations explained in the experimental section. The first two parameters are mundane and influence how quickly the program can run by increasing the thread count. For this system, it was opted to use only a single thread for each to lower system requirements and since time of the operation was not as imperative. Following this, the seed number is identified. This number is very important because it allows trials to be reproduced time after time because this number controls all random number generation, including genome generation, mutation, and crossover. Following this, the configuration file labels this as a SimpleEvolutionState problem which, as mentioned before, means it is a simple generational problem. Other class parameters, primarily the selection and breeding aspects, determine it to work as a genetic algorithm when the simple class is chosen. Following the class parameter configurations are the genetic algorithm configurations, including what type of genome is used, generation numbers, and population sizes. The final most important aspect of this parameters file is the output file, which records the output and statistics of the run. This is saved to a `.stat` file, and contains, for each generation, the individual with the highest fitness, and the genome of that individual.

The final aspect of the ECJ that should be discussed is the evaluate method. This method is what the ECJ engine references as the problem and is responsible for

```
breedthreads = 1
evalthreads  = 1
seed.0 = 1

state      = ec.simple.SimpleEvolutionState
pop        = ec.Population
init       = ec.simple.SimpleInitializer
finish     = ec.simple.SimpleFinisher
breed      = ec.simple.SimpleBreeder
eval       = ec.simple.SimpleEvaluator
stat       = ec.simple.SimpleStatistics
exch       = ec.simple.SimpleExchanger

generations          = 20
quit-on-complete      = true
checkpoint           = false
checkpoint-prefix       = ec
checkpoint-modulo       = 1

stat.file          = $out.stat

pop.subpops          = 1
pop.subpop.0         = ec.Subpopulation
pop.subpop.0.size       = 10
pop.subpop.0.duplicate-retries  = 0
pop.subpop.0.species      = ec.vector.BitVectorSpecies
pop.subpop.0.species.fitness    = ec.simple.SimpleFitness
pop.subpop.0.species.ind    = ec.vector.BitVectorIndividual
pop.subpop.0.species.genome-size = 800

pop.subpop.0.species.crossover-type = two
pop.subpop.0.species.crossover-prob = .5
pop.subpop.0.species.mutation-prob  = 0.02

pop.subpop.0.species.pipe       = ec.vector.breed.
   VectorMutationPipeline
pop.subpop.0.species.pipe.source.0    = ec.vector.breed.
   VectorCrossoverPipeline
pop.subpop.0.species.pipe.source.0.source.0   = ec.select.
   TournamentSelection
pop.subpop.0.species.pipe.source.0.source.1   = ec.select.
   TournamentSelection

select.tournament.size = 2

eval.problem  = ec.app.trafficSim.TrafficSim
```

Figure 3.5: `TrafficSim.params`: The baseline parameters file used for this study

ensuring the genomes are valid, and sending those genomes into the simulator to be evaluated. Genomes of individuals are bit vectors and are randomly created by the ECJ engine, using that random seed mentioned before. The size is determined by the number of intersections within the map, so a map with 25 intersections would be 200 bits long. This method splits the genome into a form that can be used, as explained earlier, by the simulator in the CSV format needed. After it is converted, it is written to the timings file and sent to the simulator which creates, runs, and evaluates the current configuration. The individual is then assigned its fitness as a float, and then returned to the ECJ system.

As a reminder, this study is geared towards discovering whether or not genetic algorithms can be used to optimize traffic configurations in a larger traffic grid than some of the previous studies. To do this, the ECJ system was utilized and configured in a way that would use a genetic algorithm to generate traffic signal configurations for the grid, then use the simulator to evaluate these configurations, and finally return to the ECJ system to complete the optimization process. This study then used a series of trials to confirm whether this worked or not by comparing the initial randomly generated fitness to the final fitness of the program. If the fitness value was greater than it was at the start, then it successfully optimized it. After a few preliminary trials, it appeared that the program optimized the systems every time, however the matter of degree came into question. To discover this, several performance tests were set up and ran, followed by analysis and an in depth review of how this could possibly be so. The parameters this study focus on manipulating are crossover probability, mutation probability, and finally the modifiers or weights of the variables of the fitness function. These are all self-explanatory besides the modifiers. The modifiers variable is used to give weights to the three different fitness inputs, with one parameter giving weight to each one, and one giving an equal weight to all three. Each of these parameters were exhaustively studied and for each trial, a total of 30 different random seeds were used. These seeds were randomly generated before performance testing and the same seeds were used throughout experimentation.

To test these parameters, the evaluate method and parameter file had to be changed for each configuration. To do this a program was written that generated each of the necessary files using loops to substitute the necessary values in, such as mutation rate, parameter file name, and other variables that changed per trial. The java file took the text from the actual files that would not be changed in strings, and appended them together in a way to create a new trial. These experiments each have an individual value and more importantly an output file where the results were recorded. This was deemed the easiest method to approach performance testing compared to one of the built in systems that allowed ECJ to run multiple jobs with multiple configurations in a set called jobs. These generation programs also produced index files which kept track of parameters used for each run in the form of comma separated format. These index files are important for the later analysis of the results. Additionally, to run all of these different configurations, a bash script was generated to simply run each individual trial. This script simply was an exhaustive list of all the trials generated and was, once again generated in a java program beforehand. The final aspect that was used in the preparation and experimentation phase of this process was the program which stripped all of the output files, index files, and other important details, compiling them into a single statistical analysis file. This file was in comma separated format and would later be used in analysis of the results.

3.2 Threats to Validity

There are several parts of this project which could influence the validity of this study including the accuracy of several aspects, the random generation, and other logical choices made when designing this project. The most important idea was to make an accurate simulator however, an idealized system is not a very accurate simulation of the real world. Some of the aspects that call validity into question are how cars are introduced to the system, how the cars exit the system, how they drive on the roads, and finally the basic layout of the system. The cars all enter into the system at the same time as mentioned earlier. This is not realistic in the least and affects movement a bit since the queues are flooded and cars all have the same starting time when they arrive initially in their queues. A better approach, which is slightly more complex, would involve cars being introduced sporadically throughout the duration of the simulation. This would have the trade off of taking longer to run and if too large of a system were used, it may take significantly longer, though speed was not the goal of this project. The next possible problem with this project involves how the cars exit the system. Currently, a car is pulled from the system as soon as it reaches its destination node, without regard to its more specific destination within the node, and hence, without regard to such possible time-consuming tasks as parking. This treatment of exiting a system is deemed imperative, however, to remain within the scope of the project due to time constraints. The next important aspect which could be improved on in future projects is the manner in which cars move down a road. The current system has the cars moving both regularly, and reacting immediately to changes in the environment. This is also not realistic, especially in modern traffic systems, where there is so much news of texting and other distractions influencing how cars drive. Additionally, drivers personalities can greatly influence the way events on the road occur. For example, more aggressive drivers are likely to take the initiative at a stop sign intersection while student drivers are likely to be hesitant in their behaviors on the road. All of these factors contribute to less realism because the cars objects acting extremely mechanical and logically.

Yet another problem that merits attention is one of the fundamental behaviors of the intersections. The simulator has the ability to check, at a stop sign intersection with multiple cars, each of the other car's intentions and use that knowledge to allow other cars to move, in addition to ensuring other cars have not been there longer than the current car. Light intersections have a similar knowledge and this can be slightly less realistic than needed. While knowledge is conveyed in the real world, there are always chances that cars will forget to signal or inform the others around it of its intentions. The final important element to discuss about problems with is the idealized traffic grid. There are extremely few towns or cities which have perfect grids, where roads are identical and intersections have 4 roads each. This threatens validity by showing that even if there were a trend of genetic algorithms being able to optimize a system such as the one proposed, more complex intersections and city plans could be too complex for the optimizer to solve.

Another threat to validity stems from the performance testing of the system and how different aspects were chosen. The biggest example is how cars were chosen and generated to fill the system. For the trials in the experiments, cars were randomly generated and had completely random starting points and locations. While this could be applicable to real life, as on any given day a different number of cars can be found on the road from different locations with different destinations, this is not always the case. Cities are designed in a way that holds majority of public housing and starting points are in some areas, and destinations at other points. That is to say that city planning often has a method behind designing where major traffic veins will be. By introducing randomness to starting points and destinations, there is a higher chance that there will be a wider, even coverage across the map for both starting and destination locations. In a smaller system, it would be more ideal to choose these starting and stopping destinations, saying a higher percentage of cars would start in `area A` than `area B`, as well as destination spots, however with the larger trials, this is too time consuming.

The final point worth mentioning is the choice of direction choosing algorithms. For the trials run in experiments, the simulator used a breadth-first search, shortest path algorithm. This algorithm paid no attention to speed limits or any other factors besides how many nodes that the car would pass through since nodes are associated with distance. The reason this algorithm was feasible to use is that all roads in the trials have the same speed limit thus are evenly weighted, while in real world circumstances, this would not be the case. In a city plan where speed limits were more relative and varied from road to road, a different algorithm for determining car pathing would be more applicable such as Djikstra's algorithm, which is an algorithm or finding the shortest path through a graph with weighted edges, in this circumstance extending to the speed limits of roads.

Chapter 4

Implementation

As the previous chapter covered, several different parameters were experimented with, both in preliminary trials and during a more thorough performance testing phase. These included primarily modifier weight, mutation rate, and crossover rate, all thoroughly trialed with thirty different randomly generated seeds. As performance test results were acquired and data analyzed, the prominent trend seemed to be that this optimization process succeeded in reducing the negative characteristics used to measure the efficiency of the traffic control network. To gain this insight, the starting and resulting fitnesses were both recorded and compared to each other to gain some sense of the increase in fitness from either; while some, in fact, had a smaller increase in fitness, only 10% while some had 30% increase in fitness. Additional preliminary trials were run on larger populations and larger generation numbers and while there appeared to be an increase in performance, due to the time sensitive nature of the project, these parameters were not overly tested due to their lengthy execution time.

In the 2000+ trials that were run, the percent change in fitness was 35.83% while the minimum was actually 0% and a mean of 16.1%. This is regardless of the different changes however it shows that this system is, in fact, applicable to the problem proposed. The case where the largest occurred was shown when the fuel consumption had the heaviest weight. An interesting observation, as exemplified in the graph below is that when the heavier consumption rate, the lower the mutation rate, regardless of the crossover rate. In fact this trend, as noticed in the other graphs below, is consistent throughout all performance testing. Values in these graphs closer to 0, other than the percentage change values, indicate a sense of "better" optimization. The only true change in scaling arose when more weight was given to the fuel consumption which was a matter of flooding the units, however even then the trend is still obvious.

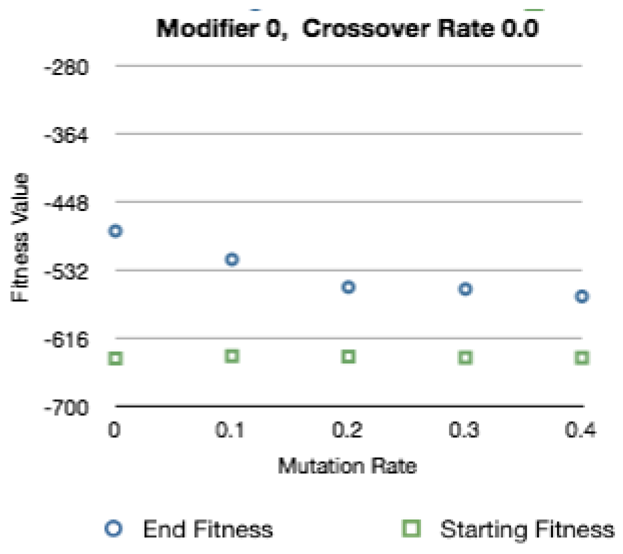

Figure 4.1: Equally weighted variables, 0.0 crossover rate fitness comparisons

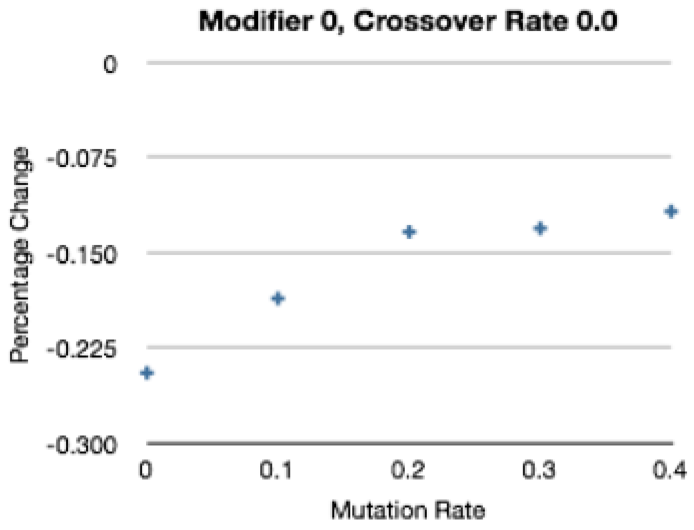

Figure 4.2: Percent change between starting fitnesses and final fitnesses

It should be noted that values closer to 0 indicate a “better” fitness level and closer to optimization, while percentage change values further from 0 indicate that a greater change happened between original fitness to final fitness. Additionally, all of these graphs use average values of the twenty different seeds used for each trial. This was to ensure an accurate coverage and number of repetitions were done.

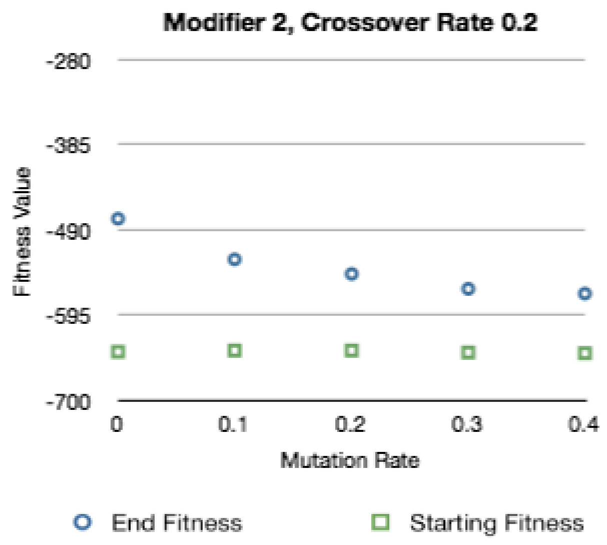

Figure 4.3: Time delay heavily weighted (.6 vs .3), 0.2 crossover rate fitness comparisons

The precise values of the trials are as follows. Generation number was held constant at 25 generations per trial, and population size was held at 20 individuals per generation. Crossover probability ranged from 0.0 to 0.6, incrementing by 0.2 each iteration. Mutation rate ranged from 0.0 to 0.5 incrementing by 0.1 each iteration. Additionally there were four modifiers used to adjust the weight of the different criteria of the fitness function. The weights are either all even with a multiplier value of 0.33, or one is weighted more heavily as 0.66 while the others remain 0.33. Modifier 0

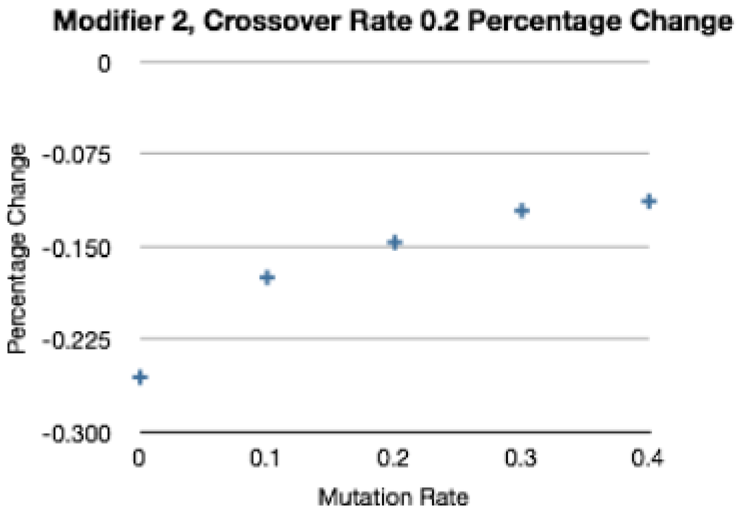

Figure 4.4: Percent change between starting fitnesses and final fitnesses

represents an even weighting, modifier 1 weighs time transit more heavily, modifier 2 weighs time delay more heavily, and finally modifier 3 weighs gas usage more heavily. These do not seem to disrupt trend but rather appear to influence the scaling as shown in there graphs.

These results support the hypothesis that using genetic algorithms can a good metric and standard for optimizing these kind of idealized city plan systems. This entails that genetic algorithms, or this approach to using genetic algorithms, may be appropriate to select, firstly, whether or not an intersection has a stop sign or a traffic

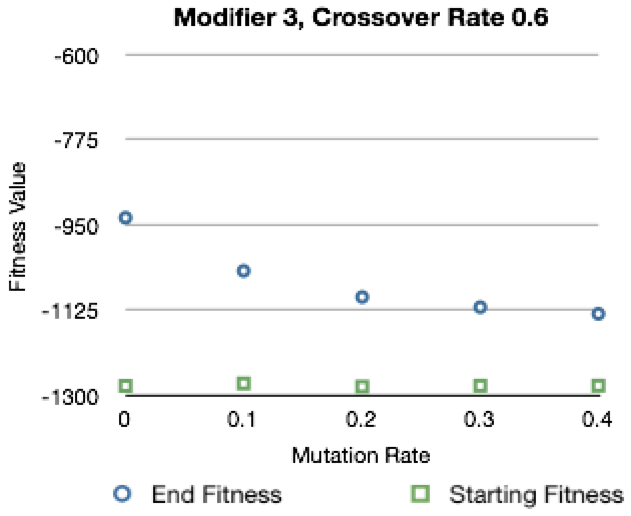

Figure 4.5: Fuel consumption heavily weighted (.6 vs .3), 0.6 crossover rate

light at each node, and secondly, the timings at each of these intersections. This means that even though there are a myriad of variables and possible configurations, the evolutionary approach of genetic algorithms allows for the program to approach a local or global ideal state. This is not to say that the chosen pipelines and manners in which selection and crossover were performed, could not have been improved. For this study, the simplest of genetic algorithms was implemented using, as seen in the parameters file earlier, tournament selection, two sources for breeding, and a tournament size of two, meaning that from each generation, four random individuals were selected, and

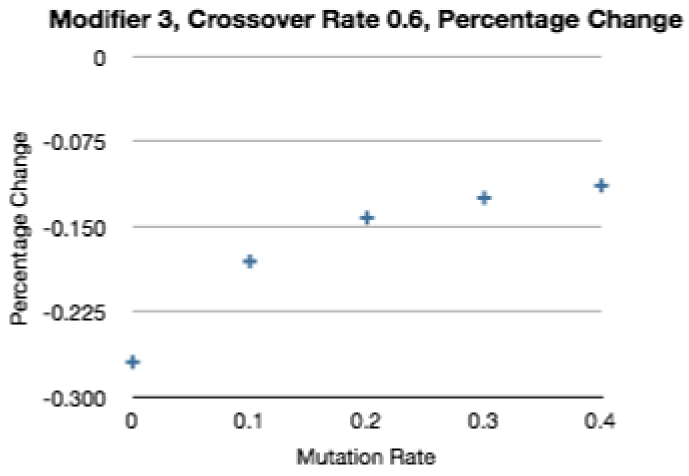

Figure 4.6: Percent change between starting fitnesses and final fitnesses

the two best of these four were used as parents for the next generation. There are several alternatives to many of these pipelines that were used and perhaps in future studies, these may prove to perform better for this type of problem.

Chapter 5

Discussion and Future Work

This section covers and discusses the possible field of future works, and broadly overviews and reviews some of the issues, discoveries, and information delivered by the other aspects of this paper.

5.1 Summary of Results

As mentioned before, the results of this project were relatively promising. The system was able to optimize the traffic grid up to 35% more than the starting fitness in some cases. While these solutions could be more accurate with the use of a more accurate simulator, this is not a bad starting point and should encourage further research in the field. It was also observed that several of the variables tested for performance, had little effect on the outcomes of the optimization process. An exception was the mutation rate, where lower values resulted in greater optimization. Finally, the only major variation in values stemmed from weighing the different variables for the fitness function differently. Even with these variations, overall trends remained consistent.

5.2 Future Work

The most easily achievable improvement to this project would be a better representation of the map and traffic system. This would include a more realistic map, while still constrained to the grid system, involving more accurate speed limits for these roads, car destination and starting point hot spots, a better direction choosing system, and a better method of having cars enter and exit the system. The speed limits and car starting and destination choices would be very simple, yet time-intensive jobs. It would entail limits within the city being set for urban, housing, and commercial sections, better approximating the way cities exist in real life. Concurrently, it would be beneficial for a better direction setting algorithm to be used in place of breadth-first search. Djikstra's algorithm is a possible replacement, because it calculates weight, or the speed limit of a road in this situation, as a factor while deciding on the best path. Finally, allowing cars to enter and exit the system at different times so that there wasn't a flood of them entering the system at once would be an improvement. This could be achieved by keeping a global time within the system and incorporating an entrance time variable for each car. Exiting the system in a more elegant manner would be slightly more involved and require additional variables and checking to be made for how far in a road a car was and needed to be to be considered at its destination. Doing this in a generic way would be slightly more complex.

Besides the above simple steps for increasing the accuracy of this project's system, there are other areas which can be improved in future work. These are primarily limited to having more accurate simulation models and using these models on more accurate real city plans. As mentioned, the primary drawback of this study was the amount of time that could be diverted to the development of the simulator. If the simulation were more accurate, then indications of how well the genetic algorithm is able to perform would also be much more accurate. Enhancements towards realism would include the use of irregular sized intersections (more or less than 4 lanes), as well as factors pertaining to the characteristics and behaviors of the cars. Additionally, incorporating aspects such as rush hour, weather, driver personalities, road construction, and other driving irregularities would render the simulator more realistic. The simulator also, as discussed earlier, is in charge of evaluating the fitness of the configuration. It would be interesting to see if a new fitness could be developed which was able to more accurately portray how well the traffic configuration was able to influence and improve the conditions within the system.

A final area of future work for this project might be the comparison of this project's methods with several of the afore-mentioned algorithms in the related works section. Although this method was able to successfully optimize systems, the degree to which they were improved could perhaps be increased through the use of other algorithms, or partial genetic algorithms which use some heuristic to avoid local minima and maxima. Comparing the results of this algorithm, and the results of other algorithms used on the same system would provide a better idea on how well this heuristic was able to optimize the system.

5.3 Conclusion

As a review, this study had the goal of showing that genetic algorithms, when coupled with a simulator to evaluate fitness, are able to adequately optimize a traffic grid to minimize several resultants of the system. These include time delay, total time transit, and gas usage of the cars within the system. To do this, a simulator was built which would take a perfect square as a grid pattern, incorporate speed limits and other aspects, introduce a number of randomly or personally configured cars, and implement an intersection configuration file to run as a discrete event simulator. The simulations eventually resulted in a fitness numbers which represented the different important values itemized earlier, that were then used to indicate how well the current light configuration performed. This simulator was then paired with the ECJ evolutionary algorithm suite to implement a genetic algorithm which generated and eventually proposed a more optimal traffic configuration.

To test the performance of these components, an exhaustive suite of trials was run impacting the mutation rate, crossover rate, and modifier weights, related to the fitness function. These were run through a bash script and the results compiled and analyzed. The findings indicate that genetic algorithms and this projects approach may be a valid and powerful tool to optimizing a large traffic grid through manipulation of the intersection traffic controls only. Additionally none of the trialed values had a major impact on the results of the trials apart from the mutation rate. In this case, it was found that having a lower mutation rate of 0 was the most powerful. This trend was produced across all other configurations of the genetic algorithm

This study encourages research to move further into the presented problem of optimizing traffic control configurations to minimize congestion and delays. Implementing different genetic algorithms is suggested. Perhaps using multi-objective genetic algorithm would yield better results. Finally, making a comparative study between this implementation and other heuristic algorithms would allow the findings of this study to be compared with other approaches and thus result in more meaningful conclusions.

Appendix A

Project Code

A.1 Simulator

A.1.1 Core

```
package ec.app.trafficSim.sim.core;


import java.util.LinkedList;
import java.util.Queue;
/**
 * Abstract Simulator, important to note any events need queues
    defined here
 * Insert methods add events back into simulator
 * @author Adam Wechter
 */
public class AbstractSimulator {
  Queue<CarEvent> carQueue = new LinkedList<CarEvent>();
  Queue<NodeEvent> nodeQueue = new LinkedList<NodeEvent>();
/**
 * Adds car events into simulator (cast)
 * @param e
 *     Abstract event cast into car event
 */
    void carInsert(AbstractEvent e) {
        carQueue.add((CarEvent)e);
    }
/**
 * Adds node event into simulator (cast)
 * @param e
 *    Abstract event cast into Node event
 */
    void nodeInsert(AbstractEvent e) {
      nodeQueue.add((NodeEvent) e);
    }
}
```

```
package ec.app.trafficSim.sim.core;
import java.util.Queue;
/**
 * Abstract Simulator class represents a simulawtor.  Includes a do
    rotation for
 * All of the events in some event queue
 * @author
 *     Adam Wechter
 */
public class Simulator extends AbstractSimulator {
  /**
   * Rotates through queue and executes each event
   * @param queue
   *     Queue of events
   */
    void doRotation(Queue queue) {
      Event e;

      int queueSize = queue.size();
      for(int i = 0; i < queueSize; i++) {
        e = (Event)queue.remove();
        e.execute(this);
      }


    }
}
```

```
package ec.app.trafficSim.sim.core;

import java.util.ArrayList;
import java.util.LinkedList;
import java.util.Queue;

import ec.app.trafficSim.sim.io.ECJPlug;
import ec.app.trafficSim.sim.util.*;

/**
 * Traffic simulator that does car and queue events
 * @author
 *    Adam Wechter
 */
public class TrafficSimulator extends Simulator {
  /**
   * Initiate puts the nodes and cars into car events and node
      events and places them in the simulator queue
   * It also executes all events
   * @param nodes
   *    List of nodes for node events
   * @param cars
   *    List of cars for car events
   */
  public void initiate(ArrayList<Node> nodes, ArrayList<CarGen> cars
     ) {

    for(CarGen car : cars) {
      carQueue.add(new CarEvent(car));
    }
    for(Node node : nodes) {
      nodeQueue.add(new NodeEvent(node));
    }
    doRotation(carQueue);

    while(!carQueue.isEmpty()) {
      if(carQueue.size() < 100)
        for(CarEvent c : carQueue)
          if(c.getCar().getCurrentQueue().getQueue().peek() != null
              && carQueue.contains(c.getCar().getCurrentQueue().
              getQueue().peek())) //if carqueue doesnt have the first
               car of a directional queue
            RuntimeOperations.exitSystem(c.getCar().getCurrentQueue
               ().getQueue().poll());

      doRotation(carQueue);
      doRotation(nodeQueue);
      for(CarEvent cE : carQueue) {
        cE.getCar().setCanGo(true);
      }



    }
  }


}
```

```
package ec.app.trafficSim.sim.core;


/**
 * Abstract event, inheritable execute method
 * @author Adam Wechter
 */
public abstract class AbstractEvent {
  /**
   * abstract event execute, defined in child
   * @param simulator
   *    simulator to which the event is re added to
   */
  abstract void execute(AbstractSimulator simulator);


}
```

```
package ec.app.trafficSim.sim.core;
/**
 * Buffer between abstract event and car/node events.
 * @author
 *    Adam Wechter
 */
public abstract class Event extends AbstractEvent {


}
```

```
package ec.app.trafficSim.sim.core;
import ec.app.trafficSim.sim.util.*;
/**
 *  Node event, contains each node, used to transition light
 * @author
 *     Adam Wechter
 */
public class NodeEvent extends Event {
  Node node;
  NodeEvent(Node node) {
    this.node = node;
  }
  /**
   * Inherited execute is in charge of changing light rotation and
      adding time
   * Always return to simulator
   * @param simulator
   *     Simulator to be readded to
   * @return
   *
   */
  @Override
  void execute(AbstractSimulator simulator) {
    if(node.getIsLightIntersection()) {
      Node.Rotation currRotation = node.getCurrentState();
      node.setTimeInCurrentRotation(node.getTimeInCurrentRotation()
         + 1);
      if(currRotation == Node.Rotation.NORTHSOUTH) {
        if(node.getTimeInCurrentRotation() == node.getNorthSouthTime
           ()) {
         node.changeRotationState();
```

```
          node.setTimeInCurrentRotation(0);
        }
      }
      else if (currRotation == Node.Rotation.EASTWEST) {
        if(node.getTimeInCurrentRotation() == node.getEastWestTime()
           ) {
          node.changeRotationState();
          node.setTimeInCurrentRotation(0);
        }
      }
      else if(currRotation == Node.Rotation.TRANSITION) {
        if(node.getTimeInCurrentRotation() == node.getTransitionTime
           ()) {
          node.changeRotationState();
          node.setTimeInCurrentRotation(0);
        }
      }
    }







    simulator.nodeInsert(this);


  }
  public Node getNode() {
    return node;
  }
}
```

```
package ec.app.trafficSim.sim.core;
import ec.app.trafficSim.sim. util.*;
/**
 * Car event contains a car
 * Important method is execute, which allows car to move through
    system
 * @author
 *     Adam Wechter
 */
public class CarEvent extends Event {
  private CarGen car;
  /**
   * Creates a CarEvent
   * @param car
   *     Generic car of the event
   */
  CarEvent(CarGen car) {
    this.car = car;
  }
  /**
   * CarEvent Getter
   * @return
   *     Car in car event
  */
  public CarGen getCar() {
    return car;
  }
  /**
   * Overidden generic execute event.  Sends car to its logic
       decision making for updates in runtime
   * operations.  Doesnt return anything, but readds car event to
```

```
    simulator IF hasnt reached destination
 * @param simulator
 *    Abstract simulator that this car is returned to
 */
@Override
public void execute(AbstractSimulator simulator) {
  //has it reached destination
  if(car.getIsInSystem()) {
    if(car.getCurrentNode().equals(car.getDestinationNode()))
      RuntimeOperations.exitSystem(car);
  //means that its not at destination lets determine the type of
     intersection then make our decision of what happens
    else if(car.getCurrentNode().getIsLightIntersection() == false
        && car.getTimeAtFront() > 5) {
      RuntimeOperations.setRestOfQueueNoMove(car);
      RuntimeOperations.updateCarInfo(car);
    }
    else if(car.getCurrentNode().getIsLightIntersection() == true
       && car.getTimeInCurrent() >75 && car.getCurrentQueue().
       getQueue().peek().equals(this) && ( car.getQueueType() == "
       north" || car.getQueueType()== "south" ) && car.
       getCurrentNode().getCurrentState() == Node.Rotation.
       NORTHSOUTH) {
      RuntimeOperations.setRestOfQueueNoMove(car);
      RuntimeOperations.updateCarInfo(car);
    }
    else if(car.getCurrentNode().getIsLightIntersection() == true
       && car.getTimeInCurrent() >75 && car.getCurrentQueue().
       getQueue().peek().equals(this)&& ( car.getQueueType() == "
       east" || car.getQueueType()== "west" ) && car.
       getCurrentNode().getCurrentState() == Node.Rotation.
       EASTWEST) {
```

```
    RuntimeOperations.setRestOfQueueNoMove(car);
    RuntimeOperations.updateCarInfo(car);
  }

  else
    if(car.getCurrentNode().getType().equals("latRoad") || car.
       getCurrentNode().getType().equals("ladRoad")) {
      RuntimeOperations.roadOperations(car);
    }
    else {  //must be an intersection
    //which type of intersection?
      if(car.getCurrentNode().getIsLightIntersection() == true)
        RuntimeOperations.lightIntersectionOperation(car);
      else
        RuntimeOperations.signIntersectionOperation(car);

    }
  //Cleaning nodes of any null from queue
  //Important to maintain system integrity
  Node node = car.getCurrentNode();
  if(node.getType().equals("intersection")) {
    node.getNorthQueue().getQueue().remove(null);
    node.getSouthQueue().getQueue().remove(null);
    node.getEastQueue().getQueue().remove(null);
    node.getWestQueue().getQueue().remove(null);
  }
  else if(node.getType().equals("ladRoad")) {
    node.getNorthQueue().getQueue().remove(null);
    node.getSouthQueue().getQueue().remove(null);
  }
  else {
    node.getEastQueue().getQueue().remove(null);
```

```
      node.getWestQueue().getQueue().remove(null);
    }
    node = car.getNextNode();
    if(node != null) {
      if(node.getType().equals("intersection")) {
        node.getNorthQueue().getQueue().remove(null);
        node.getSouthQueue().getQueue().remove(null);
        node.getEastQueue().getQueue().remove(null);
        node.getWestQueue().getQueue().remove(null);
      }
      else if(node.getType().equals("ladRoad")) {
        node.getNorthQueue().getQueue().remove(null);
        node.getSouthQueue().getQueue().remove(null);
      }
      else {
        node.getEastQueue().getQueue().remove(null);
        node.getWestQueue().getQueue().remove(null);
      }
    }
    node = car.getPreviousNode();
    if(node != null) {
      if(node.getType().equals("intersection")) {
        node.getNorthQueue().getQueue().remove(null);
        node.getSouthQueue().getQueue().remove(null);
        node.getEastQueue().getQueue().remove(null);
        node.getWestQueue().getQueue().remove(null);
      }
      else if(node.getType().equals("ladRoad")) {
        node.getNorthQueue().getQueue().remove(null);
        node.getSouthQueue().getQueue().remove(null);
      }
      else {
            node.getEastQueue().getQueue().remove(null);
            node.getWestQueue().getQueue().remove(null);
          }
        }

        //Add car event back into simulator
        simulator.carInsert(this);
      }
    }



}
```

```
package ec.app.trafficSim.sim.core;

import java.util.ArrayList;
import java.util.LinkedList;
import java.util.Queue;

import ec.app.trafficSim.sim.io.ECJPlug;
import ec.app.trafficSim.sim.util.*;

/**
 * Logic for moving the car through system as well as updating cars
    in system and having cars exit the system
 * @author
 *    Adam Wechter
 */
public class RuntimeOperations {



  /**
   * checks for conflicts for stop signs
   * @param car1
   * @param car2
   * @return
   *    true if no conflict, false otherwise
   */
  public static boolean hasNoConflict(CarGen car1, CarGen car2) {
    if(car2 == null)
      return true;
    String car1CurrentQueueType = car1.getQueueType();
    String car2CurrentQueueType = car2.getQueueType();
String car1Movement = car1.getCarMovement();
String car2Movement = car2.getCarMovement();

if( (car1CurrentQueueType.equals("NORTH") &&
   car2CurrentQueueType.equals("SOUTH")) || (
   car1CurrentQueueType.equals("SOUTH") && car2CurrentQueueType.
   equals("NORTH")) || (car1CurrentQueueType.equals("EAST") &&
   car2CurrentQueueType.equals("WEST")) || (car1CurrentQueueType
   .equals("WEST") && car2CurrentQueueType.equals("EAST")) ) {
  if(car1Movement.equals("STRAIGHT") || car1Movement.equals("
     RIGHT")) {
      if(car2Movement.equals("STRAIGHT") || car2Movement.equals(
         "RIGHT"))
        return true;
      else
        return false;
  }
  else {
    if(car2Movement.equals("LEFT"))
      return true;
    else
      return false;
  }
}

else if( (car1CurrentQueueType.equals("NORTH") &&
   car2CurrentQueueType.equals("WEST")) || (car1CurrentQueueType
   .equals("WEST") && car2CurrentQueueType.equals("SOUTH")) || (
   car1CurrentQueueType.equals("SOUTH") && car2CurrentQueueType.
   equals("EAST")) || (car1CurrentQueueType.equals("EAST") &&
   car2CurrentQueueType.equals("NORTH")) ) {
  if(car1Movement.equals("STRAIGHT"))
```

```
      return false;
    else if(car1Movement.equals("LEFT")) {
      if(car2Movement.equals("RIGHT"))
        return true;
      else
        return false;
    }
    else {
      if(car2Movement.equals("STRAIGHT") || car2Movement.equals("
          RIGHT"))
        return true;
      else
        return false;
    }

  }

  else {
    if(car1Movement.equals("STRAIGHT")) {
      if(car2Movement.equals("RIGHT"))
        return true;
      else
        return false;
    }
    else if(car1Movement.equals("RIGHT")) {
      if(car2Movement.equals("LEFT"))
        return true;
      else
        return false;
    }
    else {
      if(car2Movement.equals("LEFT"))
```

```
          return true;
        else
          return false;
      }
    }
  }

  /**
   * Preps and removes car from a system as well as sends the car to
       have its information recorded
   * @param car
   */
  public static void exitSystem(CarGen car) {
    car.setIsInSystem(false);
    if(car.getCurrentQueue() != null)
      car.getCurrentQueue().getQueue().remove(car);
    if(!car.getIsFirstRotation() && car.getTotalTime() > 2)
      car.getCurrentQueue().getQueue().remove(car);
    recordInfo(car);
  }

  /**
   * Clear all times other than total time in system
   * Also moves into next node/queue
   * @param car
   *     car in question
   */
  public static void updateCarInfo(CarGen car) {
    car.setPreviousNode();

    car.setCurrentNode();
    car.setNextNode();
    car.setTimeAtFront(0.0);
    car.setTimeInCurrent(0);
    if(car.getCurrentNode().getType().equals("intersection")) {
      car.setDelayTime(car.getDelayTime() + 1);
    }

    car.updateCurrentQueue(false);
    car.setCarMovement();
    car.setNodeType();
    car.setCanGo(false);
    car.setPriority(-1);
    car.setTotalTime((car.getTotalTime()+1));

  }

  /**
   * will only be used the first time and just to move it into the
       next node
   * @param car
   *     car in question
   */
  public static void updateCarInfoFirst(CarGen car) {
    car.setIsFirstRotation();
    car.setPreviousNode();
    car.setNextNode();
    car.setCurrentNode();
    car.setNextNode();
    car.updateCurrentQueue(true);
    car.setTimeAtFront(0);
    car.setTimeInCurrent(0);
    car.setCarMovement();
    car.setTotalTime((car.getTotalTime()+1));

  }



  /**
   * if car is still on the road, we just want to increment timers
      so basically "do nothing"
   * @param car
   *    car is in some queue and we want it to just sit
   */
  public static void doNothing(CarGen car) {
    if(car.getCurrentQueue().getQueue().peek().equals(car))
      car.setTimeAtFront((car.getTimeAtFront() + 1));
    car.setTimeInCurrent((car.getTimeInCurrent() +1));
    car.setTotalTime((car.getTotalTime()+1));
    if(car.getCurrentNode().getType().equals("intersection")) {
      car.setDelayTime(car.getDelayTime() + 1);
    }
    car.setCanGo(false);
  }



  /**
   * Handles decison making and logic as to whether or not the car
      can update/move on or must just sit/donothing
   * @param car
   */
  public static void roadOperations(CarGen car) {
    double timeOnRoad = car.getTimeInCurrent();

    if(car.getIsFirstRotation() == true) {
      updateCarInfoFirst(car);
      return;
    }
    if(!car.getCanGo()) {
      doNothing(car);
      return;
    }
    if(!car.getCurrentQueue().getQueue().peek().equals(car)) {
      doNothing(car);
      return;
    }
    setRestOfQueueNoMove(car);

    if((timeOnRoad * car.getCurrentNode().getSpeedLimit()) >= car.
       getCurrentNode().getDistance()  && car.getCurrentQueue().
       getQueue().peek().equals(car)) {
      updateCarInfo(car);
      return;
    }
    else
      doNothing(car);

  }



  /**
   * Handles decision making and logic as to whether or not the car
       can update/move on or needs to do nothing and just sit
   * @param car
   */
  public static void signIntersectionOperation(CarGen car) {
    if(car.getIsFirstRotation() == true) {  //first rotation
      updateCarInfoFirst(car);
      return;
    }
    if(!car.getCanGo()) {
      doNothing(car);
      return;
    }
    if(!car.getCurrentQueue().getQueue().peek().equals(car)) {  //if
        not at the front cant go
      doNothing(car);
      return;
    }

    setRestOfQueueNoMove(car);  //at front of its respective queue,
       rest cannot move this second
    if(car.getTimeAtFront() > 4) {
      updateCarInfo(car);
      return;
    }
    ArrayList<CarGen> thereLongest = getThereLongest(car);

    if(!thereLongest.contains(car)) {
      doNothing(car);
      return;
    }
    else {
      for(int i = 0; i < thereLongest.size(); i++) {
        if( thereLongest.get(i) == null) {
          thereLongest.remove(i);
          i = 0;
        }
      }
```

```
if(thereLongest.size() == 1 ) { //if theres only one car that
    has priority and if its this one, it can go
  updateCarInfo(car);
  return;
}
else if (thereLongest.get(1)==null) {
  updateCarInfo(car);
  return;
}
else if(thereLongest.size() == 2 ) {
  //check for conflict
  CarGen car2 = thereLongest.get((thereLongest.get(0).equals(
      car) ? 1 : 0));
  if(hasNoConflict(car, car2)) {
    updateCarInfo(car);
    return;
  }

  int maybe = (int)(Math.random()*200);
  if(maybe %2 != 0) {  //cant go
    doNothing(car);
    return;
  }
  else {
    updateCarInfo(car);
    car2.setCanGo(false);
  }
}
else if(thereLongest.size() == 3 ) {
  int maybe = (int)(Math.random()*200);
  if(maybe %3 != 0) {  //cant go
    doNothing(car);
```

```
      return;
    }
    else {
      thereLongest.remove(car);
      CarGen car2 = thereLongest.get(0);
      if(hasNoConflict(car, car2)) {
        updateCarInfo(car);
        thereLongest.get(1).setCanGo(false);
        return;
      }
      else if(hasNoConflict(car, thereLongest.get(1))) {
        updateCarInfo(car);
        car2.setCanGo(false);
        return;
      }
      else {
        updateCarInfo(car);
        for(CarGen temp : thereLongest)
          temp.setCanGo(false);
        return;
      }
    }
  }
  else if(thereLongest.size() == 4) {  //all four arrived at
     same time
    int maybe = (int)(Math.random()*200);
    if(maybe %4 != 0) {  //cant go
      doNothing(car);
      return;
    }
    else {
      thereLongest.remove(car);
```

```
        if(hasNoConflict(car, thereLongest.get(0))) {
          updateCarInfo(car);
          thereLongest.get(1).setCanGo(false);
          thereLongest.get(2).setCanGo(false);
          return;
        }
        else if(hasNoConflict(car, thereLongest.get(1))) {
          updateCarInfo(car);
          thereLongest.get(0).setCanGo(false);
          thereLongest.get(2).setCanGo(false);
          return;
        }
        else if(hasNoConflict(car, thereLongest.get(2))) {
          updateCarInfo(car);
          thereLongest.get(0).setCanGo(false);
          thereLongest.get(1).setCanGo(false);
          return;
        }
        else {
          updateCarInfo(car);
          thereLongest.get(0).setCanGo(false);
          thereLongest.get(1).setCanGo(false);
          thereLongest.get(2).setCanGo(false);
          return;
        }
      }

    }
  }
}
```

```
/**
 * Handles decision making and logic as to whether or not the car
    can update/move on or needs to do nothing
 * @param car
 */
public static void lightIntersectionOperation(CarGen car) {
  if(car.getIsFirstRotation() == true)  { //first rotation
    updateCarInfoFirst(car);
    return;
  }
  if(!car.getCanGo()) {
    doNothing(car);
    return;
  }

  boolean isGreen = isGreen(car);

  if(isGreen) {
    if(!car.getCurrentQueue().getQueue().peek().equals(car))  //if
        not front
      doNothing(car);
    else {  //front of queue
      setRestOfQueueNoMove(car); //nothing else can move
      if(car.getCarMovement().equals("STRAIGHT") || car.
          getCarMovement().equals("RIGHT")) {
        updateCarInfo(car);
      }
      else
        if(getQueueOpposite(car).getQueue().isEmpty()) {
          updateCarInfo(car);
        }
        else
```

```
            doNothing(car);
        }
    }
    else { //is red or transition
        if(car.getCurrentNode().getCurrentState() == Node.Rotation.
            TRANSITION && car.getCarMovement().equals("LEFT") && car.
            getCurrentQueue().getQueue().peek().equals(car)) {
          updateCarInfo(car);
          setRestOfQueueNoMove(car);
        }
        else if( car.getCurrentNode().getCurrentState() != Node.
            Rotation.TRANSITION && car.getCarMovement().equals("Right")
             && getQueueLeft(car).getQueue().isEmpty() && car.
            getCurrentQueue().getQueue().peek().equals(car)) {
          updateCarInfo(car);
          setRestOfQueueNoMove(car);
        }
        else
          doNothing(car);
    }

}



/**
 * Simple method to determine if green or not green.  Note if not
     green return false, notonly if red, but also transition
 * @param car
 * @return
 *     if the car.currentQueue's light is green
 */
private static boolean isGreen(CarGen car) {
    if( (car.getQueueType().equals("NORTH") || car.getQueueType().
        equals("SOUTH")) )
      if(car.getCurrentNode().getCurrentState() == Node.Rotation.
            NORTHSOUTH)
          return true;
        else
          return false;
    else
        if(car.getCurrentNode().getCurrentState() == Node.Rotation.
            EASTWEST)
          return true;
        else
          return false;
}



/**
 * Gets opposite direction queue
 * @param car
 * @return
 *    queue opposite of car's queue
 */
private static DirectionalQueue getQueueOpposite(CarGen car) {
    if(car.getCurrentQueue().getDirection().equals("NORTH"))
          return car.getCurrentNode().getSouthQueue();
    else if(car.getCurrentQueue().getDirection().equals("SOUTH"))
        return car.getCurrentNode().getNorthQueue();
    else if(car.getCurrentQueue().getDirection().equals("EAST"))
        return car.getCurrentNode().getWestQueue();
    else
        return car.getCurrentNode().getEastQueue();
}
```

```
/**
 * Gets the queue to the left in the intersection
 * @param car
 * @return
 *    queue to left of car's queue
 */
private static DirectionalQueue getQueueLeft(CarGen car) {
  if(car.getCurrentQueue().getDirection().equals("NORTH"))
    return car.getCurrentNode().getEastQueue();
  else if(car.getCurrentQueue().getDirection().equals("SOUTH"))
    return car.getCurrentNode().getWestQueue();
  else if(car.getCurrentQueue().getDirection().equals("EAST"))
    return car.getCurrentNode().getSouthQueue();
  else
    return car.getCurrentNode().getNorthQueue();
}


/**
 * Gets the queue to the right in the intersection
 * @param car
 * @return
 *    queue to right of car's queue
 */
private static DirectionalQueue getQueueRight(CarGen car) {
  if(car.getCurrentQueue().getDirection().equals("NORTH"))
    return car.getCurrentNode().getWestQueue();
  else if(car.getCurrentQueue().getDirection().equals("SOUTH"))
    return car.getCurrentNode().getEastQueue();
  else if(car.getCurrentQueue().getDirection().equals("EAST"))
```

```
      return car.getCurrentNode().getNorthQueue();
    else
      return car.getCurrentNode().getSouthQueue();
  }



  /**
   * Method calculates of all of the cars at an intersection, which
       was there longest
   * Only used in stop sign intersections.
   * @param car
   * @return
   *     Car that was in the queue longest out of all of them
   */
  private static ArrayList<CarGen> getThereLongest(CarGen car) {
    ArrayList<CarGen> thereLongest= new ArrayList<CarGen>();
    thereLongest.add(car);



    boolean isLeftEmpty = getQueueLeft(car).getQueue().isEmpty();
    boolean isRightEmpty = getQueueRight(car).getQueue().isEmpty();
    boolean isOppositeEmpty = getQueueOpposite(car).getQueue().
        isEmpty();

    if(!isLeftEmpty && getQueueLeft(car).getQueue().peek()==null){

      isLeftEmpty = true;
    }
    if(!isRightEmpty && getQueueRight(car).getQueue().peek()==null)
      isRightEmpty = true;
    if(!isOppositeEmpty && getQueueOpposite(car).getQueue().peek()==
        null)
      isOppositeEmpty = true;
    double leftTimeAtFront = 0;
    double rightTimeAtFront = 0;
    double oppositeTimeAtFront = 0;
    double thisTimeAtFront = car.getTimeAtFront();
    double longestTimeAtFront = thisTimeAtFront;
    if(!isLeftEmpty && getQueueLeft(car).getQueue().peek().getCanGo
       ())
      leftTimeAtFront = getQueueLeft(car).getQueue().peek().
         getTimeAtFront();

    if(!isRightEmpty && getQueueRight(car).getQueue().peek().
       getCanGo())
      rightTimeAtFront = getQueueRight(car).getQueue().peek().
         getTimeAtFront();

    if(!isOppositeEmpty && getQueueOpposite(car).getQueue().peek().
       getCanGo())
      oppositeTimeAtFront = getQueueOpposite(car).getQueue().peek().
         getTimeAtFront();

    if(leftTimeAtFront > longestTimeAtFront) {
      thereLongest.clear();
      longestTimeAtFront = leftTimeAtFront;
      thereLongest.add(getQueueLeft(car).getQueue().peek());
    }
    else if(leftTimeAtFront == longestTimeAtFront)
      thereLongest.add(getQueueLeft(car).getQueue().peek());
    else;

    if(rightTimeAtFront > longestTimeAtFront) {
      thereLongest.clear();
```

```
    longestTimeAtFront = rightTimeAtFront;
    thereLongest.add(getQueueRight(car).getQueue().peek());
  }
  else if(rightTimeAtFront == longestTimeAtFront)
    thereLongest.add(getQueueRight(car).getQueue().peek());
  else;

  if(oppositeTimeAtFront > longestTimeAtFront) {
    thereLongest.clear();
    longestTimeAtFront = oppositeTimeAtFront;
    thereLongest.add(getQueueOpposite(car).getQueue().peek());
  }
  else if(oppositeTimeAtFront == longestTimeAtFront)
    thereLongest.add(getQueueOpposite(car).getQueue().peek());
  else;
  return thereLongest;
}



/**
 * If the car at the front has moved, that means that another
    queue still in the same "time frame" could
 * have its decision making moved later.  If this is the case then
     we need to indicate that all other
 * cars within this queue are not allowed to move.  This method
    does it by using a temporary queue to store
 * the existing queue.
 * @param car
 *     some car at the front of a queue that made some action and
    should forbid others from doing so
 */
public static void setRestOfQueueNoMove(CarGen car) {
    CarGen c = null;
    Queue<CarGen> q = car.getCurrentQueue().getQueue();
    //at this point we know it IS at the front so set all of them to
        cant move
    for(int i = 0; i < car.getCurrentQueue().getQueue().size(); i++)
        {
      c = car.getCurrentQueue().getQueue().poll();
      c.setCanGo(false);
      q.add(c);
    }
  }



  /**
   * record information of the car into the record string
   * @param car
  */
  private static void recordInfo(CarGen car) {
    ECJPlug.cars.add(car);
    ECJPlug.emissions.add(car.getGreenGas());
    ECJPlug.timeTransit.add(car.getTotalTime());
    ECJPlug.totalDelay.add(car.getDelayTime());
    ECJPlug.milesPerGallon.add(car.getTotalTime() * car.
       getAverageTimeConsumption());

  }
}
```

A.1.2 Input/Output

```
package ec.app.trafficSim.sim.io;




import java.io.File;
import java.io.FileReader;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

import ec.CSVReader;

//import ec.lib.au.com.bytecode.opencsv.CSVReader;
import java.io.Reader;

import ec.app.trafficSim.sim.util.Edge;
import ec.app.trafficSim.sim.util.TrafficMap;
import ec.app.trafficSim.sim.util.CarGen;
import ec.app.trafficSim.sim.util.Graph;
import ec.app.trafficSim.sim.util.Node;
import ec.app.trafficSim.sim.core.RuntimeOperations;
import ec.app.trafficSim.sim.core.TrafficSimulator;

@SuppressWarnings("unused")

/**
 * In charge of reading in information and acting as a proxy between
     the ECJ optimizer
 * And the simulator.  Also contains the fitness evaluator
 * @author
 *    Adam Wechter
 */
public class ECJPlug {

  //These are record keeping lists. used when recording information
     from exitting cars
  public static List<Double> timeTransit;
  public static List<Double> milesPerGallon;
  public static List<Double> totalDelay;
  public static List<Double> emissions;
  public static List<CarGen> cars;
    /**
     * Method is important to read in files and intiate both record
        keeping information and simulator
     * @param nodeFile
     *    Contains node information
     * @param edgeFile
     *    Contains connections of nodes
     * @param carsFile
     *    Contains all car information
     * @param modifier1
     *    Weight of modifier1(Timetransit) (0-1)
     * @param modifier2
     *    Weight of modifier2(timeDelay) (0-1)
     * @param modifier3
     *    Weight of modifier3(mpg) (0-1)
     * @param timingsFile
     *    Light configuration in binary
     * @return
     *    Float fitness for the ecj to use
     */
```

```
public static double evaluateFitness(String nodeFile, String
    edgeFile, String carsFile, double modifier1, double modifier2
    , double modifier3, String timingsFile) throws
    FileNotFoundException, IOException {
  int width;
  int height;
  int length;

  /*
   * setting up saving stuff
   */

  timeTransit = new ArrayList<Double>();
  milesPerGallon = new ArrayList<Double>();
  totalDelay = new ArrayList<Double>();
  emissions = new ArrayList<Double>();
  cars = new ArrayList<CarGen>();
  numRepeated = 0;
  lastNumInCars = 0;
  numOfLogs = 0;


  /** Nodes **/
  File test = new File(nodeFile);
  System.out.println("File:  " + test.getAbsolutePath());
  CSVReader reader = new CSVReader(new FileReader(nodeFile));
    String [] nextLine;
    nextLine = reader.readNext();
      height = Integer.parseInt(nextLine[0]);
    width = Integer.parseInt(nextLine[1]);
      length = Integer.parseInt(nextLine[2]);
    ArrayList<Boolean> nodeIsI = new ArrayList<Boolean>();
    ArrayList<String> nodeId = new ArrayList<String>();
    ArrayList<Integer> nodeSpeedLimit = new ArrayList<Integer
       >();
    ArrayList<String> nodeType = new ArrayList<String>();


    while ((nextLine = reader.readNext()) != null) {
    nodeIsI.add(Boolean.parseBoolean(nextLine[0]));
    nodeId.add(nextLine[1]);
    nodeSpeedLimit.add(Integer.parseInt(nextLine[2]));
    nodeType.add(nextLine[3]);
  }
    reader.close();


    /** EDGES **/
    reader = new CSVReader(new FileReader(edgeFile));
  ArrayList<String> node1Id = new ArrayList<String>();
  ArrayList<String> node2Id = new ArrayList<String>();
  while((nextLine = reader.readNext()) != null) {
    node1Id.add(nextLine[0]);
    node2Id.add(nextLine[1]);
    node1Id.add(nextLine[1]);
    node2Id.add(nextLine[0]);
  }
 //create the map
  TrafficMap trafficMap = new TrafficMap(height, width, length
     , nodeIsI, nodeId, nodeSpeedLimit, nodeType, node1Id,
     node2Id);
  ArrayList<Edge> edges = new ArrayList<Edge>();
  edges.addAll(trafficMap.getMap().getE());
```

```
reader.close();

/**CARS*/
reader = new CSVReader(new FileReader(carsFile));
ArrayList<String> startNodes = new ArrayList<String>();
ArrayList<String> destNodes = new ArrayList<String>();
ArrayList<Double> greenGas = new ArrayList<Double>();
ArrayList<String> carId = new ArrayList<String>();
ArrayList<Double> averageTimeConsumption = new ArrayList<
    Double>();
while((nextLine = reader.readNext()) != null) {
  startNodes.add(nextLine[0]);
  destNodes.add(nextLine[1]);
  greenGas.add(Double.parseDouble(nextLine[2]));
  carId.add(nextLine[3]);
  averageTimeConsumption.add(Double.parseDouble(nextLine[4])
      );
}
reader.close();
//car init
ArrayList<CarGen> autos = new ArrayList<CarGen>();
for(int i = 0; i<startNodes.size(); i++) {  //may be just a
    < instead of <=
  autos.add(new CarGen(startNodes.get(i), destNodes.get(i),
      greenGas.get(i), averageTimeConsumption.get(i), carId.
      get(i), trafficMap));
}


/**     Timings **/
ArrayList<Boolean> isLightIntersection = new ArrayList<
    Boolean>();
```

```
ArrayList<Integer> timeNorthSouth = new ArrayList<Integer>()
    ;
ArrayList<Integer> timeEastWest = new ArrayList<Integer>();
ArrayList<Boolean> isNorthSouthFirst = new ArrayList<Boolean
    >();

reader = new CSVReader(new FileReader(timingsFile));
System.out.println("TIMINGS!!");
while((nextLine = reader.readNext()) != null) {
  if(nextLine[0].equals("1"))
    isLightIntersection.add(true);
  else
    isLightIntersection.add(false);
  timeNorthSouth.add(intify(nextLine[1]));
  timeEastWest.add(intify(nextLine[2]));
  if(nextLine[3].equals("1"))
    isNorthSouthFirst.add(true);
  else
    isNorthSouthFirst.add(false);
}
reader.close();

Graph g = trafficMap.getMap();
ArrayList<Node> nodes = g.getV();
int i = 0;
for(int k = 0; k < nodes.size(); k++) {
  if(nodes.get(k).getType().equals("intersection")) {
    nodes.get(k).setIsLightIntersection(isLightIntersection.
        get(i));
    nodes.get(k).setNorthSouthTime((timeNorthSouth.get(i)
        *15) + 15);
```

```
            nodes.get(k).setEastWestTime((timeEastWest.get(i)*15) +
                15);
            nodes.get(k).setCurrentState(isNorthSouthFirst.get(i));
            i++;
          }
        }

        new TrafficSimulator().initiate(nodes, autos);



        float fitness = 0;
        fitness = calculateFitness(modifier1, modifier2, modifier3);

      return fitness;
    }
    /**
     *  Converts binary string (3 digits) into an int
     * @param str
     *     String of 3 binary digits
     * @return sum
     *     value of binary number
     */
    private static int intify(String str) {
      int sum = 0;
      char[] chars = str.toCharArray();
      int [] mults = {4,2,1};

      for(int i = 0; i<3; i++) {
        sum += mults[i] * Character.getNumericValue(chars[i]);
      }
      return sum;
    }
```

```
/**
 * Fitness Calculator
 * Takes record lists and calculates each car's fitness in the
    system.
 * this number is the sum of all of them and eventuallly is
    divided by number of cars to get
 * fitness of the average car in the system.
 * @return fitness
 *    Fitness value of simulation
 */
private static float calculateFitness(double modifier1, double
   modifier2, double modifier3) {

  float fitness = 0;

  for(int i = 0; i < ECJPlug.cars.size(); i++) {

    double tt = ECJPlug.timeTransit.get(i);
    double mpg = ECJPlug.milesPerGallon.get(i);  //already
       calculated
    double td = ECJPlug.totalDelay.get(i);





    CarGen c = ECJPlug.cars.get(i);
    double currDist = c.getTotalDistance();
    double trueTt = 0;
    double trueTd = 0;

    if(currDist < 1) {
      trueTd = td / 1;
```

```
          trueTt = tt / 1;
        }
        else {
          trueTt = tt/currDist;
          trueTd = td/currDist;
        }



        fitness += ((modifier1 * (trueTt)) * 10000) + ((modifier2 *
            (trueTd)) * 10000) + ((modifier3*(mpg)) * 10000);
      }
      fitness = fitness/ECJPlug.cars.size();

      return fitness;
    }


}
```

A.1.3 Utilities

```
package ec.app.trafficSim.sim.util;

import java.util.ArrayList;

/**
 * Map used to put everything together allows further customization
 *@author
 *    Adam Wechter
 */
public class TrafficMap {
  private Graph map;
  public TrafficMap(int height, int width, int length, ArrayList<
     Boolean> nodeIsI, ArrayList<String> nodeId, ArrayList<Integer>
     nodeSpeedLimit, ArrayList<String> nodeType, ArrayList<String>
     node1Id, ArrayList<String> node2Id) {
    map = new Graph(height, width, length, nodeIsI, nodeId,
       nodeSpeedLimit, nodeType, node1Id, node2Id);
  }

  public Graph getMap() {
    return map;
  }
}
```

```
package ec.app.trafficSim.sim.util;
import java.util.ArrayList;
/**
 * Graph composed of nodes and Edges
 *@author
 *    Adam Wechter
 */
public class Graph {
  private ArrayList<Node> V;  //vertices (nodes)
  private ArrayList<Edge> E;  //Edges (weights/distances)
  /**
   *  Both parameters refer to number of intersections in the grid
   * @param height
   * @param width
   * @param length  basically distance of roads
   */
  public Graph(int height, int width, int length, ArrayList<Boolean>
      nodeIsI, ArrayList<String> nodeId, ArrayList<Integer>
     nodeSpeedLimit, ArrayList<String> nodeType, ArrayList<String>
     node1Id, ArrayList<String> node2Id) {
    V = new ArrayList<Node>(((width + width-1) * (width+width-1)));
    E = new ArrayList<Edge>((((height+1)*(width)) + ((height+1) * (
       width)) * 2));




    int widthCount = 0;
    boolean isVertical = false;

    for(int i = 0; i<nodeIsI.size(); i++) {
      int  thisDistance;
      if(nodeType.get(i).equals("intersection"))
        thisDistance = 0;
      else
        thisDistance = length;

      //begin building
      Node n = new Node(nodeType.get(i), nodeId.get(i), thisDistance
          , nodeSpeedLimit.get(i), i);
      V.add(n);

      //logic for laderal vs lateral
      if(isVertical == false && widthCount == width) { //so end of
          intersections should go to vertical roads
        isVertical = true;
        widthCount = 0;
      }
      else if(isVertical == true && widthCount == width) {
        isVertical = false;
        widthCount = 0;
      }
      //ADDING NODES SHOULD BE DONE NOW
    }

    //Create and add edges to the chart
    for(int i = 0; i<node1Id.size(); i++) {
      Node node1 = getCorrespondingNode(node1Id.get(i));
      Node node2 = getCorrespondingNode(node2Id.get(i));;
      E.add(new Edge(node1, node2));
    }
  }
  /**
   * @return V
   *    ArrayList of all nodes
   */
  public ArrayList<Node> getV() {
    return V;
  }
  /**
   * @return E
   *    ArrayList of all edges
   */
  public ArrayList<Edge> getE() {
    return E;
  }
  /**
   * @param nodeId
   *    Gets specific node
   */
  public Node getCorrespondingNode(String nodeId) {
    for(Node n : V) {
      if(n.getId().equals(nodeId))
        return n;
    }
    System.out.println("NODE NOT FOUND:   " + nodeId);
    return null;
  }




}
```

```
package ec.app.trafficSim.sim.util;


import java.util.ArrayList;
/**
 * Node file contains directional quues too
 *@author
 *    Adam Wechter
 */
public class Node {
  //general
  private String type;
  private String id;
  private int intId;
  private DirectionalQueue north;
  private DirectionalQueue south;
  private DirectionalQueue east;
  private DirectionalQueue west;



  //for roads
  double distance = 0;  //equal to seconds
  double speedLimit = 0;

  //for intersections
  public static enum Rotation {
    NORTHSOUTH, EASTWEST, TRANSITION;
  }
  private double northSouthTime;
  private double eastWestTime;
  private double timeInCurrentLight;  //is used in nodeEvent (ECLIPS
      LIEZ)
```

```
private Rotation previousState;
private Rotation currentState;
private boolean isLightIntersection;

private double timeInCurrentRotation;
private final double TRANSITIONTIME = 2; //2 seconds per
   transition period



/**
 *Basic constructor, requires type of node, id, distance assoc.
 *and an int id determined by io class
 */
public Node(String type, String id, int distance, int speedLimit,
   int intId) {
  this.type = type;
  this.id = id;
  this.intId = intId;

  if(type.equals("intersection")) {
    north = new DirectionalQueue("NORTH");
    south = new DirectionalQueue("SOUTH");
    east = new DirectionalQueue("EAST");
    west = new DirectionalQueue("WEST");
    this.distance = 2;
  }
  else if(type.equals("latRoad")) {
    east = new DirectionalQueue("EAST");
    west = new DirectionalQueue("WEST");
    this.speedLimit = speedLimit;
    this.distance = distance;
  }
  else if(type.equals("ladRoad")) { //must be up and down roads (
     north / south)
    north = new DirectionalQueue("NORTH");
    south = new DirectionalQueue("SOUTH");
    this.speedLimit = speedLimit;
    this.distance = distance;
  }
}

public DirectionalQueue getNorthQueue() {
  return north;
}
public DirectionalQueue getSouthQueue() {
  return south;
}
public DirectionalQueue getEastQueue() {
  return east;
}
public DirectionalQueue getWestQueue() {
  return west;
}
public void setNorthQueue(DirectionalQueue north) {
  this.north = north;
}
public void setSouthQueue(DirectionalQueue south) {
  this.south = south;
}
public void setEastQueue(DirectionalQueue east) {
  this.east = east;
}
public void setWestQueue(DirectionalQueue west) {
  this.west = west;
```

```
}
public String getType() {
  return type;
}
public String getId() {
  return id;
}
public double getDistance() {
  return distance;
}
public double getSpeedLimit() {
  return speedLimit;
}
public Rotation getCurrentState() {
  return currentState;
}
public void setNorthSouthTime(double northSouthTime) {
  this.northSouthTime = northSouthTime;
}
public void setEastWestTime(double eastWestTime) {
  this.eastWestTime = eastWestTime;
}
public void setTimeInCurrentRotation(double timeInCurrentRotation)
    {
  this.timeInCurrentRotation = timeInCurrentRotation;
}
public double getNorthSouthTime() {
  return northSouthTime;
}
public double getEastWestTime() {
  return eastWestTime;
}
```

```
public double getTimeInCurrentRotation() {
  return timeInCurrentRotation;
}
public double getTransitionTime () {
  return TRANSITIONTIME;
}
public void setIsLightIntersection(boolean isLightIntersection) {
  this.isLightIntersection = isLightIntersection;
}
public int getIntId() {
  return intId;
}
/**
 * This is for the initial light settings
 * @param isNorthSouthFirst
 */
public void setCurrentState(boolean isNorthSouthFirst) {
  if(isNorthSouthFirst == true) {
    currentState = Rotation.NORTHSOUTH;
  }
  else {
    currentState = Rotation.EASTWEST;
  }
}
public boolean getIsLightIntersection() {
  return isLightIntersection;
}
/**
 * Takes into account the previous rotation and sets the new
    rotation.  Also stores the current
 * rotation state at the time this method is called and stores it
    inprevious after next state decision is made
 */
public void changeRotationState() {
  if(currentState == Rotation.NORTHSOUTH || currentState ==
     Rotation.EASTWEST) {
    previousState = currentState;
    currentState = Rotation.TRANSITION;
  }
  else { //current state = trasition
    if(previousState == Rotation.NORTHSOUTH) {
      previousState = currentState;
      currentState = Rotation.EASTWEST;
    }
    else {  //previousState was EAST WEST
      previousState = currentState;
      currentState = Rotation.NORTHSOUTH;
    }
  }
}

/**
 * Gets any nodes adgacent to this node
 * @param map
 *    we need some reference
 * @return
 *    array list of any connecting nodes to this node
 */
public ArrayList<Node> getConnections(Graph map) {
  ArrayList<Edge> edges = map.getE();
  ArrayList<Node> children = new ArrayList<Node>();
  for(Edge e : edges) {
    if(e.getNode1().getId().equals(id)) {
      children.add(e.getNode2());
      }
    }
    return children;
  }
}
```

```
package ec.app.trafficSim.sim.util;



import java.util.LinkedList;
import java.util.Queue;

/**
 * Directional queues are queues with directions associated with
    them.
 * Cars enter and leave toward the direction indicated (car going
    north will
 * be in a north queue)
 * @author wechtera
 *
 */
public class DirectionalQueue{

  Queue<CarGen> queue;
  String direction;  //cars enter from opposite and leave toward
     direction
  public DirectionalQueue(String direction) {
    queue = new LinkedList<CarGen>();
    this.direction = direction;
  }
  public DirectionalQueue(String direction, Queue<CarGen> queue) {
    this.queue = queue;
    this.direction = direction;
  }
  /**
   * @return
   * String of which direction either north, south, east or west
   */
  public Queue<CarGen> getQueue() {
    return queue;
  }
  /**
   * @return
   *     Direction the queue exits
   */
  public String getDirection() {
    return direction;
  }


}
```

```
package ec.app.trafficSim.sim.util;
/**
 * Edge file of edges connecting nodes
 *@author
 *    Adam Wechter
 */
public class Edge {
  int distance;
  Node node1;
  Node node2;

  public Edge(Node node1, Node node2) {
    this.node1 = node1;
    this.node2 = node2;
    distance = 1;
  }

  public int getDistance() {
    return distance;
  }
  public Node getNode1() {
    return node1;
  }
  public Node getNode2() {
    return node2;
  }
}
```

```java
package ec.app.trafficSim.sim.util;
/**
 * Made for if I want to include cars with complex aspects such as
    driving styles or decisions, thus can inherit
 * @author Adam Wechter
 *
 */
public abstract class CarObj {


}
```

```
package ec.app.trafficSim.sim.util;
import java.util.LinkedList;
import java.util.ArrayList;
import java.util.Queue;
import java.util.Scanner;

/**
 * Car object, not called car just because I thought I may have an
    inheritable object of cars
 * @author Adam Wechter
 */

public class CarGen extends CarObj {

  private Node startNode;
  private Node currentNode;
  private Node destinationNode;
  private Node nextNode;
  private Node previousNode;
  private String carMovement;
  //for stop signs
  private int priority;
  private boolean canGo;

  private double totalTime;

  private double timeInCurrent;
  private double timeAtFront;  //stop signs
  private double delayTime;
  private double greenGas;
  private double averageTimeConsumption;
  private double totalDistance;
  private String carId;

  private boolean isInSystem;

  private boolean isFirstRotation;

  private String nodeType;
  private DirectionalQueue currentQueue;

  private Queue<Node> directions;  //directions



  public CarGen(String startNode, String destinationNode, double
     greenGas, double averageTimeConsumption, String carId,
     TrafficMap map) {
    this.startNode = getNodeStartNode(startNode, map);
    this.destinationNode = getNodeDestinationNode(destinationNode,
       map);
    this.carId = carId;

    currentNode = this.startNode;

    totalTime = 0;
    timeInCurrent = 0;
    timeAtFront = 0;
    delayTime = 0;
    this.averageTimeConsumption = averageTimeConsumption;

    isFirstRotation = true;
    isInSystem = true;
    this.greenGas = greenGas;
```

```
    priority = -1;
    canGo = true;
    nodeType = currentNode.getType();
    currentQueue = null;   //initially nothing but then we'll do it


    directions = getDirections(map.getMap(), this.startNode.getIntId
        (), this.destinationNode.getIntId());

    setTotalDistance();
    nextNode = directions.peek();
    previousNode = currentNode;
  }
  /**
   * Goal:  Find what directionalqueue this car is in
   * Procedure:  get all queue directions, search, if find car,
      return that direction, else search others
   * @return
   *     Queue which the car is in (lane)
   */
  private DirectionalQueue getQueue() {
    //get all the queues
    DirectionalQueue north = currentNode.getNorthQueue();
    DirectionalQueue south = currentNode.getSouthQueue();
    DirectionalQueue east =  currentNode.getEastQueue();
    DirectionalQueue west =  currentNode.getWestQueue();


    if(!currentNode.getType().equals("latRoad")) {
      Queue<CarGen> northQueue = north.getQueue();
      for(CarObj c : northQueue) {
```

```
      if(c.equals(this))
        return north;
    }
    Queue<CarGen> southQueue = south.getQueue();
    for(CarObj c : southQueue) {
      if(c.equals(this))
        return south;
    }
  }
  Queue<CarGen> eastQueue = east.getQueue();
  for(CarObj c : eastQueue) {
    if(c.equals(this))
      return east;
  }
  Queue<CarGen> westQueue = west.getQueue();
  for(CarObj c : westQueue) {
    if(c.equals(this))
      return west;
  }

  //IT WILL NEVER GET TO THIS POINT
  return null;
}
/**
 * @return startNode
 *     Cars start node
 */
public Node getStartNode() {
  return startNode;
}

/**
```

```
 * @param startNode
 *     sets startNode as node
 */
public void setStartNode(Node startNode) {
  this.startNode = startNode;
}
/**
 * @return currentNode
 *     Cars current node
 */
public Node getCurrentNode() {
  return currentNode;
}

/**
 *  Sets current node based on directions
 */
public void setCurrentNode() {
  currentNode = directions.peek();
}
  /**
 * @return destinationNode
 *     Car's destination node
 */
public Node getDestinationNode() {
  return destinationNode;
}

/**
 * @param destinationNode
 *     Sets destination Node as Node
 */
```

```
public void setDestinationNode(Node destinationNode) {
  this.destinationNode = destinationNode;
}

/**
 * @return nextNode
 *     cars Next Node
 */
public Node getNextNode() {
  return nextNode;
}

/**
 * This will automatically increment the queue too!!!!
 * VERY IMPORTANT
 * Remember this increments queue
 */
public void setNextNode() {
  directions.remove();
  nextNode = directions.peek();
}

/**
 * @return totalTime
 *     total time in system
 */
public double getTotalTime() {
  return totalTime;
}

/**
 * @param totalTime
 *    double how long in system
 */
public void setTotalTime(double totalTime) {
  this.totalTime = totalTime;
}



/**
 * @return greenGas
 *    Green house gas emissions
 */
public double getGreenGas() {
  return greenGas;
}

/**
 * @return carId
 *    Id of car
 */
public String getCarId() {
  return carId;
}



/**
 * @return nodeType
 *    type of currentNode
 */
public String getNodeType() {
  return nodeType;
}
```

```
/**
 *  Updates what type of node
 */
public void setNodeType() {
  nodeType = currentNode.getType();
}

/**
 *    previous node automaticlally set as current node
 */
public void setPreviousNode() {
  previousNode = currentNode;
}

/**
 * @return previousNode
 *    the previous node
 */
public Node getPreviousNode() {
  return previousNode;
}

/**
 * @return timeInCurrent
 *    how long car has been in current node
 */
public double getTimeInCurrent() {
  return timeInCurrent;
}

/**
 * @param timeInCurrent
 *     double set how long its been in current queue
 */
public void setTimeInCurrent(double timeInCurrent) {
  this.timeInCurrent = timeInCurrent;
}

/**
 * @return timeAtFront
 *     time at the front of the queue
 */
public double getTimeAtFront() {
  return timeAtFront;
}

/**
 * @param timeAtFont
 *     Set how long its been at front
 */
public void setTimeAtFront(double timeAtFront) {
  this.timeAtFront = timeAtFront;
}

/**
 * @return
 *     String north, south, east, or west
 */
public String getQueueType() {
  return currentQueue.getDirection();
}

/**
 * @return currentQueue
 *    current directional queue
 */
public DirectionalQueue getCurrentQueue() {
  return currentQueue;
}

/**
 * @return directions
 *    the directions queue of this car
 */
public Queue<Node> getDirections() {
  return directions;
}

/**
 * @return carMovement
 *    which direction the car is moving next
 */
public String getCarMovement() {
  return carMovement;
}

/**
 * @return isFirstRotation
 *    boolean true if first rotation in system
 */
public boolean getIsFirstRotation() {
  return isFirstRotation;
}

/**
 * @return  isInSystem
 *    boolean true if in system
 */
public boolean getIsInSystem() {
  return isInSystem;
}

/**
 * @param isInSystem
 *    Set false once leaves
 */
public void setIsInSystem(boolean isInSystem) {
  this.isInSystem = isInSystem;
}



/**
 * only used once that matters (used every time but more
    explanation in @SEE RuntimeOperations.updateCarInfo
 * automatically changes firstRotation to False, because
    initialization changes it to true and only time it should be
    true
 */
public void setIsFirstRotation() {
  isFirstRotation = false;
}

/**
 * @param priority
 *    Intersections
 */
public void setPriority(int priority) {
  this.priority = priority;
```

```
}

/**
 * @return priority
 *     Priority at stop signs
 */
public int getPriority() {
  return priority;
}

/**
 * @param canGo
 *     Boolean can car go
 */
public void setCanGo(boolean canGo) {
  this.canGo = canGo;
}

/**
 * @return canGo
 *     boolean, true if can move that rotation
 */
public boolean getCanGo() {
  return canGo;
}

/**
 * @return delayTime
 *     double how long at intersections
 */
public double getDelayTime() {
  return delayTime;
```

```
}

/**
 * @param delayTime
 *     Set delay time
 */
public void setDelayTime(double delayTime) {
  this.delayTime = delayTime;
}

/**
 * @return averageTimeConsumption
 *     double gas consumed per tick
 */
public double getAverageTimeConsumption() {
  return averageTimeConsumption;
}

/**
 * @return totalDistance
 *     double total distance covered
 */
public double getTotalDistance() {
  return totalDistance;
}

/**
 * used for initial declartion of car.  Since it is straight from
    raw input data, we get a string
 * to reference the starting node.  This method provides the mean
    to transform that node into an
 * actual node
 * @param nodeId
 *     String representation of start node
 * @param map
 *     We need some reference
 * @return
 *     The Node with the String Id given
 */
public Node getNodeStartNode(String nodeId, TrafficMap map) {
  ArrayList<Node> nodes = map.getMap().getV();
  for(Node c : nodes) {
    if(c.getId().equals(nodeId))
      return c;
  }
  return null; //if no node found returns empty should throw
     error
}

/**
 * See getNodeStartNode Description
 * @param nodeId
 * @param map
 * @return
 *     Node associated with String Id given in the map
 */
public Node getNodeDestinationNode(String nodeId, TrafficMap map)
   {
  ArrayList<Node> nodes = map.getMap().getV();
  for(Node c : nodes)
    if(c.getId().equals(nodeId))
      return c;
  return null;
  //TODO: throw if returns null cause its an error
}
/**
 * This makes our directions using simple BFS to find the Shortest
     path
 * @param map
 * @param startNode
 * @param endNode
 * @return
 *     Returns a queue of the directions for this car from Start
    node to Finish
 */
private Queue<Node> getDirections(Graph map, int startNode, int
   endNode) {
  Queue<Node> directs = new LinkedList<Node>();
  Queue<Integer> q = new LinkedList<Integer>();
  boolean [] visited = new boolean[(map.getV().size())];
  String [] path = new String[map.getV().size()];

  q.add(startNode);
  path[startNode] = startNode + " ";
  while(q.peek() != null) {
    if(doBFS(q.poll(), endNode, visited, q, path, map))
      break;
  }

  String s = path[endNode];
  Scanner scan = new Scanner(s);
  while(scan.hasNextInt())
    directs.add(getNode(scan.nextInt(), map));
  return directs;
}
```

```
/**
 * Supporting method for BFS search
 * @param start
 * @param end
 * @param visited
 * @param q
 * @param path
 * @param map
 * @return
 *    Boolean true if the destination node was found so we know
    which index to use to find the string
 */
private boolean doBFS(int start, int end, boolean [] visited,
   Queue<Integer> q, String [] path, Graph map) {
  if(visited[start]);
  else if(start == end)
    return true;
  else {
    visited[start] = true;
    ArrayList <Node> connected = getNode(start, map).
       getConnections(map);
    for(Node n : connected) {
      int nextVert = n.getIntId();
      path[nextVert] = path[start] + nextVert + " ";
      q.add(nextVert);
    }
  }
  return false;
}

/**
 * Simple toString of the directions of each car
 * Used for debugging
 * @return
 *    Directions given as node id's from start to end
 */
public String directionsToString() {
  String str = "";
  StringBuilder sb = new StringBuilder(str);
  Queue<Node> temp = new LinkedList<Node>();
  temp.addAll(directions);

  while(!temp.isEmpty()) {
    sb.append(temp.remove().getId() + ",\n");
  }
  return sb.toString();
}
/**
 * used to update next turn type really needed only in
    intersections
 */
public void setCarMovement () {
  int currentRow = getRow(currentNode.getId());
  int currentColumn = getColumn(currentNode.getId());
  if(nextNode == null)
    return;
  int nextRow = getRow(nextNode.getId());
  int nextColumn = getColumn(nextNode.getId());
  String currentQueueType = currentQueue.getDirection();

  if(currentQueueType.equals("NORTH")) {
    if(nextRow != currentRow)
      carMovement = "STRAIGHT";
    else if(nextColumn > currentColumn)
```

```
        carMovement = "RIGHT";
      else
        carMovement = "LEFT";
    }
    else if(currentQueueType.equals("SOUTH")) {
      if(nextRow != currentRow)
        carMovement = "STRAIGHT";
      else if(nextColumn > currentColumn)
        carMovement = "LEFT";
      else
        carMovement = "RIGHT";
    }
    else if(currentQueueType.equals("EAST")) {
      if(nextColumn != currentColumn)
        carMovement = "STRAIGHT";
      else if(nextRow > currentRow)
        carMovement = "RIGHT";
      else
        carMovement = "LEFT";
    }
    else {
      if(nextColumn != currentColumn)
        carMovement = "STRAIGHT";
      else if(nextRow > currentRow)
        carMovement = "RIGHT";
      else
        carMovement = "LEFT";
    }
  }

  /**
   * Disects a node Id String for the row number
```

```
 * @param id
 *     Some NOde Id
 * @return
 *     The row that this node is in
 */
private static int getRow(String id) {
  int thisRow = 0;
  char x = 'x';
  int i = 2;
  int multValue = 1;
  while(id.charAt(i) != x) { //get index of row value
    i++;
  }
  i--;
  while(i>1) {
    thisRow += Character.getNumericValue(id.charAt(i)) * multValue
       ;
    i--;
    multValue*=10;
  }
  return thisRow;
}
/**
 * Disects a Node Id string for the column
 * @param id
 *     String of Node Id
 * @return
 *     The column the node is in
 */
private static int getColumn(String id) {
  int thisColumn = 0;
  char x = 'x';
  int i = 2;
  int multValue = 1;
  while(id.charAt(i) != x) { //get index of row value
    i++;
  }
  //i now holds end of row (on the x)
  int j = id.length() -1;
  while(j>i) {
    thisColumn += Character.getNumericValue(id.charAt(j)) *
        multValue;
    j--;
    multValue*=10;
  }
  return thisColumn;
}
/**
 * Used for changing from the number format to the actual nodes
    for directions
 * @param intId
 *    Int id of node
 * @param map
 *    Map for reference
 * @return
 *    The node associated with the Int Id
 */
private static Node getNode(int intId, Graph map) {
  Node node = null;
  ArrayList <Node> nodes = map.getV();

  for(Node n : nodes) {
    if(n.getIntId() == intId)
      node = n;
  }

  return node;
}
/**
 * See where we are and what the next node is
 *
 * using the idea that we know what direction we are aiming
     towards determine which direction(north south east west) will
    get us there
 */
public void updateCurrentQueue(boolean isFirst) {
  setCurrentType();
  int previousRow = getRow(previousNode.getId());
  int previousColumn = getColumn(previousNode.getId());
  int currentRow = getRow(currentNode.getId());
  int currentColumn = getColumn(currentNode.getId());
  if(nextNode == null) {
    return;
  }
  int nextRow = getRow(nextNode.getId());
  int nextColumn = getColumn(nextNode.getId());
  //if currently in a queue we should remove it(if its the first
      we dont need to worry
  if( !isFirst) {
    currentQueue.getQueue().remove(this);  //We should not get
        here unless it is at front but to ensure we only remove
        this, we may want to relook
  }
  if(nodeType.equals("latRoad")) { //east west queues
    if(currentColumn < nextColumn) //moving eastward
      currentNode.getEastQueue().getQueue().add(this);
    else  //moving from westward
      currentNode.getWestQueue().getQueue().add(this);
  }
  else if(nodeType.equals("ladRoad")) {
    if(currentRow < nextRow)  //moving southward
      currentNode.getSouthQueue().getQueue().add(this);
    else  //moving northward
      currentNode.getNorthQueue().getQueue().add(this);
  }
  else {  //at an intersection
    if(previousRow == currentRow) {  //definately east west queue
      if(currentColumn < previousColumn)
        currentNode.getWestQueue().getQueue().add(this);
      else {
        currentNode.getEastQueue().getQueue().add(this);
      }
    }
    else { // definatel north or south
      if(currentRow < previousRow)
        currentNode.getNorthQueue().getQueue().add(this);
      else
        currentNode.getNorthQueue().getQueue().add(this);


    }
  }
  currentQueue = getQueue();
}


/**
```

```
   * Sets total distance car must travel based on directions
   */
  private void setTotalDistance() {
    double totalDistance = 0;
    Queue<Node> directs = new LinkedList<Node>();
    directs.addAll(directions);
    while(directs.peek() != null)
    {
      Node n = directs.poll();
      totalDistance += n.getDistance();
    }
    this.totalDistance = totalDistance;
  }
  public void setCurrentType() {
    nodeType = currentNode.getType();
  }



}
```

A.1.4 Support Files

```
Rx7x10,Rx18x17,100,C0,15
Rx15x14,Ix12x12,100,C1,15
Rx15x0,Rx6x13,100,C2,15
Ix16x6,Ix4x12,100,C3,15
Rx3x10,Rx11x2,100,C4,15
Rx11x0,Rx15x14,100,C5,15
Rx7x4,Ix18x8,100,C6,15
Ix0x2,Rx13x4,100,C7,15
Rx14x17,Rx11x18,100,C8,15
Rx18x15,Ix14x16,100,C9,15
Ix12x6,Ix14x16,100,C10,15
Ix14x8,Rx8x15,100,C11,15
Ix8x6,Rx15x4,100,C12,15
Rx1x6,Rx4x5,100,C13,15
Ix4x8,Ix2x4,100,C14,15
Ix4x6,Rx7x16,100,C15,15
Rx3x16,Ix8x2,100,C16,15
Rx17x0,Rx11x2,100,C17,15
Rx3x16,Rx13x8,100,C18,15
Rx11x8,Rx11x16,100,C19,15
Ix4x2,Ix16x10,100,C20,15
Ix0x6,Rx17x0,100,C21,15
Rx1x0,Rx7x12,100,C22,15
Rx3x12,Rx11x4,100,C23,15
Ix8x6,Rx10x1,100,C24,15
Rx17x4,Rx17x6,100,C25,15
Rx8x11,Rx7x8,100,C26,15
Rx9x16,Ix18x2,100,C27,15
Rx9x6,Rx11x6,100,C28,15
Ix0x0,Rx5x2,100,C29,15
```

```
Rx7x8,Rx0x1,100,C30,15
Ix2x14,Rx5x10,100,C31,15
Rx9x2,Rx5x14,100,C32,15
Ix10x16,Rx5x6,100,C33,15
Rx6x9,Rx10x11,100,C34,15
Rx12x17,Ix10x10,100,C35,15
Rx11x18,Ix16x16,100,C36,15
Rx1x14,Rx4x3,100,C37,15
Rx11x10,Rx3x8,100,C38,15
Rx6x15,Rx11x6,100,C39,15
Rx10x11,Rx17x12,100,C40,15
Rx9x16,Rx16x11,100,C41,15
Ix10x4,Rx7x4,100,C42,15
Rx3x18,Rx5x0,100,C43,15
Ix4x16,Ix16x2,100,C44,15
Rx0x15,Rx15x8,100,C45,15
Rx1x14,Rx18x1,100,C46,15
Ix8x12,Ix12x18,100,C47,15
Ix12x6,Rx3x18,100,C48,15
Rx1x8,Ix0x6,100,C49,15
Rx17x4,Rx9x8,100,C50,15
Rx13x18,Rx0x7,100,C51,15
Rx9x0,Rx7x6,100,C52,15
Rx3x4,Rx3x0,100,C53,15
Ix18x18,Rx11x18,100,C54,15
Rx3x12,Ix0x10,100,C55,15
Rx6x17,Rx9x4,100,C56,15
Rx0x7,Rx10x15,100,C57,15
Ix6x14,Rx17x6,100,C58,15
Rx9x8,Rx16x13,100,C59,15
Ix10x10,Ix8x14,100,C60,15
Ix6x10,Ix0x14,100,C61,15
```

```
Rx16x17,Rx7x18,100,C62,15

Rx17x16,Ix10x18,100,C63,15

Ix4x4,Rx7x6,100,C64,15

Rx3x12,Rx6x15,100,C65,15

Rx7x8,Rx9x6,100,C66,15

Rx13x8,Ix14x10,100,C67,15

Rx15x4,Rx5x14,100,C68,15

Rx15x4,Rx3x6,100,C69,15

Ix18x18,Ix4x4,100,C70,15

Rx5x18,Rx6x13,100,C71,15

Rx5x12,Rx9x2,100,C72,15

Rx15x4,Rx10x9,100,C73,15

Rx2x17,Ix4x14,100,C74,15

Rx16x7,Ix16x10,100,C75,15

Rx1x10,Rx18x7,100,C76,15

Rx0x11,Rx3x18,100,C77,15

Rx6x1,Ix6x16,100,C78,15

Ix16x18,Ix10x12,100,C79,15

Rx17x8,Rx17x14,100,C80,15

Ix6x18,Rx4x15,100,C81,15

Rx15x6,Rx7x0,100,C82,15

Rx7x16,Rx5x18,100,C83,15

Rx7x14,Rx9x6,100,C84,15

Rx1x18,Rx16x9,100,C85,15

Rx10x17,Rx7x14,100,C86,15

Rx13x4,Ix2x18,100,C87,15

Ix4x6,Rx2x11,100,C88,15

Rx18x7,Ix8x12,100,C89,15

Rx15x10,Rx9x12,100,C90,15

Rx9x4,Ix6x14,100,C91,15

Rx0x5,Rx13x0,100,C92,15

Rx5x10,Rx3x10,100,C93,15
```

```
Rx10x9,Rx5x18,100,C94,15
Rx3x10,Rx6x1,100,C95,15
Ix12x6,Ix14x6,100,C96,15
Rx8x13,Rx9x12,100,C97,15
Rx14x5,Ix0x8,100,C98,15
Rx11x4,Rx1x4,100,C99,15
Ix4x2,Rx16x7,100,C100,15
Rx15x6,Rx7x16,100,C101,15
Rx5x10,Ix12x12,100,C102,15
Ix14x16,Rx17x0,100,C103,15
Ix14x4,Rx15x18,100,C104,15
Rx14x3,Rx2x13,100,C105,15
Ix8x6,Rx17x0,100,C106,15
Rx4x7,Rx8x3,100,C107,15
Rx9x4,Rx15x18,100,C108,15
Rx1x0,Rx17x0,100,C109,15
Ix18x4,Ix18x6,100,C110,15
Rx8x1,Rx7x12,100,C111,15
Rx14x7,Rx15x6,100,C112,15
Ix2x8,Rx13x16,100,C113,15
Rx2x3,Rx9x2,100,C114,15
Ix0x4,Rx9x18,100,C115,15
Rx7x10,Rx7x0,100,C116,15
Rx17x0,Rx4x13,100,C117,15
Rx12x5,Ix4x8,100,C118,15
Rx13x14,Rx8x7,100,C119,15
Ix8x2,Rx9x2,100,C120,15
Rx5x10,Rx15x18,100,C121,15
Rx14x3,Rx2x13,100,C122,15
Rx7x12,Rx8x5,100,C123,15
Rx11x14,Ix18x4,100,C124,15
Ix6x18,Rx17x18,100,C125,15
```

```
Rx11x12,Rx10x9,100,C126,15
Rx7x14,Rx1x0,100,C127,15
Rx7x8,Rx10x11,100,C128,15
Rx18x9,Rx3x18,100,C129,15
Ix18x16,Rx8x5,100,C130,15
Rx9x8,Rx13x16,100,C131,15
Rx11x4,Rx4x3,100,C132,15
Ix16x18,Rx17x14,100,C133,15
Rx9x10,Rx17x16,100,C134,15
Ix18x12,Ix10x4,100,C135,15
Rx17x18,Ix6x2,100,C136,15
Rx11x4,Ix12x6,100,C137,15
Rx13x8,Rx16x5,100,C138,15
Ix16x2,Rx11x6,100,C139,15
Rx3x12,Rx15x6,100,C140,15
Ix4x2,Rx2x13,100,C141,15
Rx8x1,Ix8x6,100,C142,15
Rx15x14,Rx7x18,100,C143,15
Rx17x18,Rx3x10,100,C144,15
Rx16x13,Ix0x0,100,C145,15
Rx6x3,Rx13x14,100,C146,15
Ix16x18,Ix14x16,100,C147,15
Rx6x13,Ix4x10,100,C148,15
Ix4x14,Rx13x6,100,C149,15
Ix2x0,Rx15x10,100,C150,15
Rx4x3,Rx7x18,100,C151,15
Ix16x0,Rx3x6,100,C152,15
Rx9x10,Ix14x8,100,C153,15
Ix8x6,Ix16x16,100,C154,15
Rx1x14,Rx13x4,100,C155,15
Ix8x8,Rx17x8,100,C156,15
Ix10x6,Rx3x8,100,C157,15
```

```
Ix0x2 ,Ix4x18 ,100 ,C158 ,15

Rx10x15 ,Rx13x14 ,100 ,C159 ,15

Rx17x8 ,Rx9x2 ,100 ,C160 ,15

Ix18x2 ,Ix0x18 ,100 ,C161 ,15

Ix10x6 ,Ix2x12 ,100 ,C162 ,15

Rx1x6 ,Rx3x4 ,100 ,C163 ,15

Ix4x6 ,Ix10x16 ,100 ,C164 ,15

Rx17x4 ,Ix12x10 ,100 ,C165 ,15

Rx18x17 ,Rx1x2 ,100 ,C166 ,15

Ix2x18 ,Rx7x16 ,100 ,C167 ,15

Rx13x0 ,Rx13x12 ,100 ,C168 ,15

Rx8x13 ,Rx5x4 ,100 ,C169 ,15

Rx15x8 ,Ix8x14 ,100 ,C170 ,15

Rx7x4 ,Rx11x12 ,100 ,C171 ,15

Rx7x14 ,Rx13x8 ,100 ,C172 ,15

Rx15x10 ,Rx18x11 ,100 ,C173 ,15

Rx9x6 ,Rx6x9 ,100 ,C174 ,15

Rx10x3 ,Rx9x16 ,100 ,C175 ,15

Rx11x2 ,Ix12x12 ,100 ,C176 ,15

Rx10x11 ,Rx7x12 ,100 ,C177 ,15

Ix8x4 ,Rx17x16 ,100 ,C178 ,15

Ix2x0 ,Rx7x4 ,100 ,C179 ,15

Rx5x10 ,Rx11x10 ,100 ,C180 ,15

Rx14x1 ,Rx11x18 ,100 ,C181 ,15

Ix12x18 ,Rx14x13 ,100 ,C182 ,15

Rx9x2 ,Rx11x16 ,100 ,C183 ,15

Rx16x1 ,Rx17x12 ,100 ,C184 ,15

Rx6x7 ,Ix6x16 ,100 ,C185 ,15

Rx15x12 ,Rx2x17 ,100 ,C186 ,15

Rx4x5 ,Ix12x2 ,100 ,C187 ,15

Ix16x8 ,Rx10x17 ,100 ,C188 ,15

Rx10x13 ,Rx17x16 ,100 ,C189 ,15
```

```
Rx10x15 ,Ix8x4 ,100 ,C190 ,15
Rx4x7 ,Rx18x15 ,100 ,C191 ,15
Rx16x9 ,Rx0x11 ,100 ,C192 ,15
Rx7x12 ,Rx14x7 ,100 ,C193 ,15
Rx15x16 ,Rx15x0 ,100 ,C194 ,15
Rx1x14 ,Rx10x1 ,100 ,C195 ,15
Rx16x3 ,Ix2x2 ,100 ,C196 ,15
Rx7x16 ,Ix0x16 ,100 ,C197 ,15
Rx14x5 ,Ix12x0 ,100 ,C198 ,15
Rx3x12 ,Ix4x18 ,100 ,C199 ,15
Rx4x11 ,Ix12x10 ,100 ,C200 ,15
Rx14x1 ,Rx3x8 ,100 ,C201 ,15
Rx15x12 ,Rx8x9 ,100 ,C202 ,15
Rx18x17 ,Rx17x16 ,100 ,C203 ,15
Rx0x11 ,Ix16x12 ,100 ,C204 ,15
Rx6x5 ,Rx13x10 ,100 ,C205 ,15
Rx17x16 ,Rx3x4 ,100 ,C206 ,15
Ix12x8 ,Ix6x8 ,100 ,C207 ,15
Ix2x14 ,Ix14x6 ,100 ,C208 ,15
Rx7x14 ,Rx11x0 ,100 ,C209 ,15
Rx17x18 ,Rx5x4 ,100 ,C210 ,15
Rx11x12 ,Ix16x8 ,100 ,C211 ,15
Rx5x10 ,Rx13x4 ,100 ,C212 ,15
Ix8x4 ,Rx17x16 ,100 ,C213 ,15
Rx17x4 ,Rx15x18 ,100 ,C214 ,15
Ix16x0 ,Ix8x10 ,100 ,C215 ,15
Rx15x12 ,Rx4x13 ,100 ,C216 ,15
Rx13x18 ,Rx4x3 ,100 ,C217 ,15
Rx1x2 ,Rx9x12 ,100 ,C218 ,15
Rx13x18 ,Rx7x16 ,100 ,C219 ,15
Rx8x15 ,Rx9x0 ,100 ,C220 ,15
Rx13x12 ,Ix0x18 ,100 ,C221 ,15
```

```
Rx18x15,Rx10x5,100,C222,15
Rx9x12,Rx12x17,100,C223,15
Ix2x6,Ix18x0,100,C224,15
Rx1x12,Rx13x14,100,C225,15
Rx13x16,Ix16x4,100,C226,15
Rx13x16,Rx7x0,100,C227,15
Ix14x10,Ix6x4,100,C228,15
Rx3x10,Ix2x16,100,C229,15
Rx14x5,Rx11x8,100,C230,15
Ix4x12,Rx1x2,100,C231,15
Rx3x14,Ix12x12,100,C232,15
Rx17x4,Rx7x0,100,C233,15
Rx12x13,Rx13x4,100,C234,15
Rx5x12,Rx13x4,100,C235,15
Rx6x11,Ix8x16,100,C236,15
Ix14x18,Rx3x16,100,C237,15
Ix18x0,Rx3x4,100,C238,15
Rx13x8,Rx6x1,100,C239,15
Rx9x16,Ix8x12,100,C240,15
Ix14x16,Rx9x6,100,C241,15
Rx3x16,Rx13x18,100,C242,15
Rx11x18,Ix12x8,100,C243,15
Rx10x13,Rx9x10,100,C244,15
Rx1x14,Rx1x2,100,C245,15
Rx15x4,Rx3x14,100,C246,15
Rx11x6,Rx13x6,100,C247,15
Rx10x9,Ix14x10,100,C248,15
Rx3x8,Rx2x11,100,C249,15
Rx5x0,Rx7x0,100,C250,15
Rx13x8,Rx1x16,100,C251,15
Rx2x3,Rx15x16,100,C252,15
Rx8x7,Ix4x8,100,C253,15
```

```
Rx13x0,Rx9x0,100,C254,15
Rx15x10,Rx8x15,100,C255,15
Rx7x14,Rx5x6,100,C256,15
Rx16x1,Ix10x10,100,C257,15
Rx14x1,Ix4x2,100,C258,15
Rx1x18,Rx7x4,100,C259,15
Rx1x6,Rx9x2,100,C260,15
Rx10x11,Rx1x14,100,C261,15
Rx18x3,Ix18x12,100,C262,15
Rx3x0,Ix8x2,100,C263,15
Rx5x18,Rx7x16,100,C264,15
Rx13x6,Ix4x2,100,C265,15
Rx17x0,Rx13x12,100,C266,15
Rx16x9,Rx2x9,100,C267,15
Rx15x0,Rx10x11,100,C268,15
Rx3x10,Ix10x6,100,C269,15
Rx10x7,Rx1x18,100,C270,15
Ix14x0,Rx11x6,100,C271,15
Ix18x6,Rx7x8,100,C272,15
Rx18x7,Rx1x4,100,C273,15
Rx13x16,Rx9x16,100,C274,15
Rx1x2,Rx0x15,100,C275,15
Rx9x10,Rx18x7,100,C276,15
Rx17x18,Rx13x6,100,C277,15
Rx15x10,Rx7x8,100,C278,15
Rx15x8,Rx15x18,100,C279,15
Rx17x10,Rx9x16,100,C280,15
Rx9x4,Rx13x18,100,C281,15
Ix4x0,Rx8x15,100,C282,15
Rx11x6,Rx17x10,100,C283,15
Rx16x7,Rx10x15,100,C284,15
Ix2x10,Ix12x16,100,C285,15
```

```
Ix6x8,Rx18x15,100,C286,15
Ix16x4,Ix0x14,100,C287,15
Ix12x16,Rx15x18,100,C288,15
Rx5x12,Ix12x14,100,C289,15
Rx13x4,Ix0x16,100,C290,15
Ix2x6,Rx10x15,100,C291,15
Rx11x14,Rx13x14,100,C292,15
Rx15x14,Ix6x8,100,C293,15
Rx15x8,Ix10x4,100,C294,15
Rx1x4,Ix10x12,100,C295,15
Rx17x0,Rx5x4,100,C296,15
Rx17x2,Ix2x4,100,C297,15
Rx4x15,Ix2x12,100,C298,15
Rx17x10,Rx1x4,100,C299,15
Rx15x12,Rx0x1,100,C300,15
Rx15x14,Rx17x18,100,C301,15
Ix6x12,Rx17x18,100,C302,15
Rx13x18,Rx3x14,100,C303,15
Ix14x18,Ix18x0,100,C304,15
Rx1x14,Ix6x18,100,C305,15
Ix14x6,Ix8x10,100,C306,15
Rx5x0,Ix2x2,100,C307,15
Rx13x8,Rx1x14,100,C308,15
Rx11x6,Ix4x4,100,C309,15
Rx17x0,Rx10x13,100,C310,15
Ix4x2,Rx7x0,100,C311,15
Rx14x3,Rx6x13,100,C312,15
Rx2x3,Rx13x4,100,C313,15
Rx10x17,Rx13x18,100,C314,15
Ix8x18,Rx17x0,100,C315,15
Rx5x16,Rx11x14,100,C316,15
Rx1x4,Rx15x6,100,C317,15
```

```
Rx13x2,Ix2x0,100,C318,15
Rx2x1,Ix4x10,100,C319,15
Rx17x2,Ix6x0,100,C320,15
Rx11x14,Rx3x2,100,C321,15
Rx11x12,Rx2x3,100,C322,15
Rx12x17,Rx12x9,100,C323,15
Rx1x2,Ix0x16,100,C324,15
Rx13x0,Rx9x4,100,C325,15
Rx7x6,Rx5x12,100,C326,15
Ix10x14,Rx2x13,100,C327,15
Rx17x4,Rx15x6,100,C328,15
Ix0x18,Rx13x6,100,C329,15
Rx14x3,Rx1x4,100,C330,15
Rx7x6,Rx15x18,100,C331,15
Rx17x4,Ix0x16,100,C332,15
Rx5x6,Rx15x14,100,C333,15
Ix0x0,Rx13x12,100,C334,15
Rx13x0,Rx5x4,100,C335,15
Ix0x12,Rx8x7,100,C336,15
Rx1x2,Rx11x10,100,C337,15
Ix8x8,Rx17x6,100,C338,15
Rx1x18,Rx10x15,100,C339,15
Rx13x18,Ix8x10,100,C340,15
Rx11x2,Ix12x16,100,C341,15
Rx14x5,Rx1x2,100,C342,15
Rx16x17,Rx15x18,100,C343,15
Ix10x12,Rx10x3,100,C344,15
Rx3x4,Ix14x18,100,C345,15
Ix2x12,Rx3x2,100,C346,15
Ix14x8,Rx1x8,100,C347,15
Ix10x12,Rx7x4,100,C348,15
Rx1x16,Rx17x8,100,C349,15
```

```
Rx9x8 , Ix0x10 ,100 , C350 ,15
Rx2x3 , Rx7x4 ,100 , C351 ,15
Rx13x12 , Rx15x16 ,100 , C352 ,15
Rx0x17 , Rx9x8 ,100 , C353 ,15
Rx10x11 , Rx13x0 ,100 , C354 ,15
Rx18x15 , Rx17x6 ,100 , C355 ,15
Rx12x11 , Rx17x10 ,100 , C356 ,15
Rx11x8 , Rx2x5 ,100 , C357 ,15
Rx4x15 , Rx11x6 ,100 , C358 ,15
Rx16x5 , Rx15x2 ,100 , C359 ,15
```

```
Ix0x0,Rx0x1
Rx0x1,Ix0x2
Ix0x2,Rx0x3
Rx0x3,Ix0x4
Ix0x4,Rx0x5
Rx0x5,Ix0x6
Ix0x6,Rx0x7
Rx0x7,Ix0x8
Ix0x8,Rx0x9
Rx0x9,Ix0x10
Ix0x10,Rx0x11
Rx0x11,Ix0x12
Ix0x12,Rx0x13
Rx0x13,Ix0x14
Ix0x14,Rx0x15
Rx0x15,Ix0x16
Ix0x16,Rx0x17
Rx0x17,Ix0x18
Rx1x0,Ix0x0
Rx1x0,Ix2x0
Rx1x2,Ix0x2
Rx1x2,Ix2x2
Rx1x4,Ix0x4
Rx1x4,Ix2x4
Rx1x6,Ix0x6
Rx1x6,Ix2x6
Rx1x8,Ix0x8
Rx1x8,Ix2x8
Rx1x10,Ix0x10
Rx1x10,Ix2x10
Rx1x12,Ix0x12
```

```
Rx1x12,Ix2x12
Rx1x14,Ix0x14
Rx1x14,Ix2x14
Rx1x16,Ix0x16
Rx1x16,Ix2x16
Rx1x18,Ix0x18
Rx1x18,Ix2x18
Ix2x0,Rx2x1
Rx2x1,Ix2x2
Ix2x2,Rx2x3
Rx2x3,Ix2x4
Ix2x4,Rx2x5
Rx2x5,Ix2x6
Ix2x6,Rx2x7
Rx2x7,Ix2x8
Ix2x8,Rx2x9
Rx2x9,Ix2x10
Ix2x10,Rx2x11
Rx2x11,Ix2x12
Ix2x12,Rx2x13
Rx2x13,Ix2x14
Ix2x14,Rx2x15
Rx2x15,Ix2x16
Ix2x16,Rx2x17
Rx2x17,Ix2x18
Rx3x0,Ix2x0
Rx3x0,Ix4x0
Rx3x2,Ix2x2
Rx3x2,Ix4x2
Rx3x4,Ix2x4
Rx3x4,Ix4x4
Rx3x6,Ix2x6
```

```
Rx3x6,Ix4x6

Rx3x8,Ix2x8

Rx3x8,Ix4x8

Rx3x10,Ix2x10

Rx3x10,Ix4x10

Rx3x12,Ix2x12

Rx3x12,Ix4x12

Rx3x14,Ix2x14

Rx3x14,Ix4x14

Rx3x16,Ix2x16

Rx3x16,Ix4x16

Rx3x18,Ix2x18

Rx3x18,Ix4x18

Ix4x0,Rx4x1

Rx4x1,Ix4x2

Ix4x2,Rx4x3

Rx4x3,Ix4x4

Ix4x4,Rx4x5

Rx4x5,Ix4x6

Ix4x6,Rx4x7

Rx4x7,Ix4x8

Ix4x8,Rx4x9

Rx4x9,Ix4x10

Ix4x10,Rx4x11

Rx4x11,Ix4x12

Ix4x12,Rx4x13

Rx4x13,Ix4x14

Ix4x14,Rx4x15

Rx4x15,Ix4x16

Ix4x16,Rx4x17

Rx4x17,Ix4x18

Rx5x0,Ix4x0
```

```
Rx5x0,Ix6x0
Rx5x2,Ix4x2
Rx5x2,Ix6x2
Rx5x4,Ix4x4
Rx5x4,Ix6x4
Rx5x6,Ix4x6
Rx5x6,Ix6x6
Rx5x8,Ix4x8
Rx5x8,Ix6x8
Rx5x10,Ix4x10
Rx5x10,Ix6x10
Rx5x12,Ix4x12
Rx5x12,Ix6x12
Rx5x14,Ix4x14
Rx5x14,Ix6x14
Rx5x16,Ix4x16
Rx5x16,Ix6x16
Rx5x18,Ix4x18
Rx5x18,Ix6x18
Ix6x0,Rx6x1
Rx6x1,Ix6x2
Ix6x2,Rx6x3
Rx6x3,Ix6x4
Ix6x4,Rx6x5
Rx6x5,Ix6x6
Ix6x6,Rx6x7
Rx6x7,Ix6x8
Ix6x8,Rx6x9
Rx6x9,Ix6x10
Ix6x10,Rx6x11
Rx6x11,Ix6x12
Ix6x12,Rx6x13
```

```
Rx6x13 ,Ix6x14
Ix6x14 ,Rx6x15
Rx6x15 ,Ix6x16
Ix6x16 ,Rx6x17
Rx6x17 ,Ix6x18
Rx7x0 ,Ix6x0
Rx7x0 ,Ix8x0
Rx7x2 ,Ix6x2
Rx7x2 ,Ix8x2
Rx7x4 ,Ix6x4
Rx7x4 ,Ix8x4
Rx7x6 ,Ix6x6
Rx7x6 ,Ix8x6
Rx7x8 ,Ix6x8
Rx7x8 ,Ix8x8
Rx7x10 ,Ix6x10
Rx7x10 ,Ix8x10
Rx7x12 ,Ix6x12
Rx7x12 ,Ix8x12
Rx7x14 ,Ix6x14
Rx7x14 ,Ix8x14
Rx7x16 ,Ix6x16
Rx7x16 ,Ix8x16
Rx7x18 ,Ix6x18
Rx7x18 ,Ix8x18
Ix8x0 ,Rx8x1
Rx8x1 ,Ix8x2
Ix8x2 ,Rx8x3
Rx8x3 ,Ix8x4
Ix8x4 ,Rx8x5
Rx8x5 ,Ix8x6
Ix8x6 ,Rx8x7
```

```
Rx8x7,Ix8x8

Ix8x8,Rx8x9

Rx8x9,Ix8x10

Ix8x10,Rx8x11

Rx8x11,Ix8x12

Ix8x12,Rx8x13

Rx8x13,Ix8x14

Ix8x14,Rx8x15

Rx8x15,Ix8x16

Ix8x16,Rx8x17

Rx8x17,Ix8x18

Rx9x0,Ix8x0

Rx9x0,Ix10x0

Rx9x2,Ix8x2

Rx9x2,Ix10x2

Rx9x4,Ix8x4

Rx9x4,Ix10x4

Rx9x6,Ix8x6

Rx9x6,Ix10x6

Rx9x8,Ix8x8

Rx9x8,Ix10x8

Rx9x10,Ix8x10

Rx9x10,Ix10x10

Rx9x12,Ix8x12

Rx9x12,Ix10x12

Rx9x14,Ix8x14

Rx9x14,Ix10x14

Rx9x16,Ix8x16

Rx9x16,Ix10x16

Rx9x18,Ix8x18

Rx9x18,Ix10x18

Ix10x0,Rx10x1
```

```
Rx10x1,Ix10x2
Ix10x2,Rx10x3
Rx10x3,Ix10x4
Ix10x4,Rx10x5
Rx10x5,Ix10x6
Ix10x6,Rx10x7
Rx10x7,Ix10x8
Ix10x8,Rx10x9
Rx10x9,Ix10x10
Ix10x10,Rx10x11
Rx10x11,Ix10x12
Ix10x12,Rx10x13
Rx10x13,Ix10x14
Ix10x14,Rx10x15
Rx10x15,Ix10x16
Ix10x16,Rx10x17
Rx10x17,Ix10x18
Rx11x0,Ix10x0
Rx11x0,Ix12x0
Rx11x2,Ix10x2
Rx11x2,Ix12x2
Rx11x4,Ix10x4
Rx11x4,Ix12x4
Rx11x6,Ix10x6
Rx11x6,Ix12x6
Rx11x8,Ix10x8
Rx11x8,Ix12x8
Rx11x10,Ix10x10
Rx11x10,Ix12x10
Rx11x12,Ix10x12
Rx11x12,Ix12x12
Rx11x14,Ix10x14
```

```
Rx11x14,Ix12x14
Rx11x16,Ix10x16
Rx11x16,Ix12x16
Rx11x18,Ix10x18
Rx11x18,Ix12x18
Ix12x0,Rx12x1
Rx12x1,Ix12x2
Ix12x2,Rx12x3
Rx12x3,Ix12x4
Ix12x4,Rx12x5
Rx12x5,Ix12x6
Ix12x6,Rx12x7
Rx12x7,Ix12x8
Ix12x8,Rx12x9
Rx12x9,Ix12x10
Ix12x10,Rx12x11
Rx12x11,Ix12x12
Ix12x12,Rx12x13
Rx12x13,Ix12x14
Ix12x14,Rx12x15
Rx12x15,Ix12x16
Ix12x16,Rx12x17
Rx12x17,Ix12x18
Rx13x0,Ix12x0
Rx13x0,Ix14x0
Rx13x2,Ix12x2
Rx13x2,Ix14x2
Rx13x4,Ix12x4
Rx13x4,Ix14x4
Rx13x6,Ix12x6
Rx13x6,Ix14x6
Rx13x8,Ix12x8
```

```
Rx13x8,Ix14x8
Rx13x10,Ix12x10
Rx13x10,Ix14x10
Rx13x12,Ix12x12
Rx13x12,Ix14x12
Rx13x14,Ix12x14
Rx13x14,Ix14x14
Rx13x16,Ix12x16
Rx13x16,Ix14x16
Rx13x18,Ix12x18
Rx13x18,Ix14x18
Ix14x0,Rx14x1
Rx14x1,Ix14x2
Ix14x2,Rx14x3
Rx14x3,Ix14x4
Ix14x4,Rx14x5
Rx14x5,Ix14x6
Ix14x6,Rx14x7
Rx14x7,Ix14x8
Ix14x8,Rx14x9
Rx14x9,Ix14x10
Ix14x10,Rx14x11
Rx14x11,Ix14x12
Ix14x12,Rx14x13
Rx14x13,Ix14x14
Ix14x14,Rx14x15
Rx14x15,Ix14x16
Ix14x16,Rx14x17
Rx14x17,Ix14x18
Rx15x0,Ix14x0
Rx15x0,Ix16x0
Rx15x2,Ix14x2
```

```
Rx15x2,Ix16x2
Rx15x4,Ix14x4
Rx15x4,Ix16x4
Rx15x6,Ix14x6
Rx15x6,Ix16x6
Rx15x8,Ix14x8
Rx15x8,Ix16x8
Rx15x10,Ix14x10
Rx15x10,Ix16x10
Rx15x12,Ix14x12
Rx15x12,Ix16x12
Rx15x14,Ix14x14
Rx15x14,Ix16x14
Rx15x16,Ix14x16
Rx15x16,Ix16x16
Rx15x18,Ix14x18
Rx15x18,Ix16x18
Ix16x0,Rx16x1
Rx16x1,Ix16x2
Ix16x2,Rx16x3
Rx16x3,Ix16x4
Ix16x4,Rx16x5
Rx16x5,Ix16x6
Ix16x6,Rx16x7
Rx16x7,Ix16x8
Ix16x8,Rx16x9
Rx16x9,Ix16x10
Ix16x10,Rx16x11
Rx16x11,Ix16x12
Ix16x12,Rx16x13
Rx16x13,Ix16x14
Ix16x14,Rx16x15
```

```
Rx16x15,Ix16x16
Ix16x16,Rx16x17
Rx16x17,Ix16x18
Rx17x0,Ix16x0
Rx17x0,Ix18x0
Rx17x2,Ix16x2
Rx17x2,Ix18x2
Rx17x4,Ix16x4
Rx17x4,Ix18x4
Rx17x6,Ix16x6
Rx17x6,Ix18x6
Rx17x8,Ix16x8
Rx17x8,Ix18x8
Rx17x10,Ix16x10
Rx17x10,Ix18x10
Rx17x12,Ix16x12
Rx17x12,Ix18x12
Rx17x14,Ix16x14
Rx17x14,Ix18x14
Rx17x16,Ix16x16
Rx17x16,Ix18x16
Rx17x18,Ix16x18
Rx17x18,Ix18x18
Ix18x0,Rx18x1
Rx18x1,Ix18x2
Ix18x2,Rx18x3
Rx18x3,Ix18x4
Ix18x4,Rx18x5
Rx18x5,Ix18x6
Ix18x6,Rx18x7
Rx18x7,Ix18x8
Ix18x8,Rx18x9
```

```
Rx18x9,Ix18x10

Ix18x10,Rx18x11

Rx18x11,Ix18x12

Ix18x12,Rx18x13

Rx18x13,Ix18x14

Ix18x14,Rx18x15

Rx18x15,Ix18x16

Ix18x16,Rx18x17

Rx18x17,Ix18x18
```

```
10,10,400
true,Ix0x0,0,intersection
false,Rx0x1,35,latRoad
true,Ix0x2,0,intersection
false,Rx0x3,35,latRoad
true,Ix0x4,0,intersection
false,Rx0x5,35,latRoad
true,Ix0x6,0,intersection
false,Rx0x7,35,latRoad
true,Ix0x8,0,intersection
false,Rx0x9,35,latRoad
true,Ix0x10,0,intersection
false,Rx0x11,35,latRoad
true,Ix0x12,0,intersection
false,Rx0x13,35,latRoad
true,Ix0x14,0,intersection
false,Rx0x15,35,latRoad
true,Ix0x16,0,intersection
false,Rx0x17,35,latRoad
true,Ix0x18,0,intersection
false,Rx1x0,35,ladRoad
false,Rx1x2,35,ladRoad
false,Rx1x4,35,ladRoad
false,Rx1x6,35,ladRoad
false,Rx1x8,35,ladRoad
false,Rx1x10,35,ladRoad
false,Rx1x12,35,ladRoad
false,Rx1x14,35,ladRoad
false,Rx1x16,35,ladRoad
false,Rx1x18,35,ladRoad
true,Ix2x0,0,intersection
```

```
false,Rx2x1,35,latRoad
true,Ix2x2,0,intersection
false,Rx2x3,35,latRoad
true,Ix2x4,0,intersection
false,Rx2x5,35,latRoad
true,Ix2x6,0,intersection
false,Rx2x7,35,latRoad
true,Ix2x8,0,intersection
false,Rx2x9,35,latRoad
true,Ix2x10,0,intersection
false,Rx2x11,35,latRoad
true,Ix2x12,0,intersection
false,Rx2x13,35,latRoad
true,Ix2x14,0,intersection
false,Rx2x15,35,latRoad
true,Ix2x16,0,intersection
false,Rx2x17,35,latRoad
true,Ix2x18,0,intersection
false,Rx3x0,35,ladRoad
false,Rx3x2,35,ladRoad
false,Rx3x4,35,ladRoad
false,Rx3x6,35,ladRoad
false,Rx3x8,35,ladRoad
false,Rx3x10,35,ladRoad
false,Rx3x12,35,ladRoad
false,Rx3x14,35,ladRoad
false,Rx3x16,35,ladRoad
false,Rx3x18,35,ladRoad
true,Ix4x0,0,intersection
false,Rx4x1,35,latRoad
true,Ix4x2,0,intersection
false,Rx4x3,35,latRoad
```

```
true,Ix4x4,0,intersection
false,Rx4x5,35,latRoad
true,Ix4x6,0,intersection
false,Rx4x7,35,latRoad
true,Ix4x8,0,intersection
false,Rx4x9,35,latRoad
true,Ix4x10,0,intersection
false,Rx4x11,35,latRoad
true,Ix4x12,0,intersection
false,Rx4x13,35,latRoad
true,Ix4x14,0,intersection
false,Rx4x15,35,latRoad
true,Ix4x16,0,intersection
false,Rx4x17,35,latRoad
true,Ix4x18,0,intersection
false,Rx5x0,35,ladRoad
false,Rx5x2,35,ladRoad
false,Rx5x4,35,ladRoad
false,Rx5x6,35,ladRoad
false,Rx5x8,35,ladRoad
false,Rx5x10,35,ladRoad
false,Rx5x12,35,ladRoad
false,Rx5x14,35,ladRoad
false,Rx5x16,35,ladRoad
false,Rx5x18,35,ladRoad
true,Ix6x0,0,intersection
false,Rx6x1,35,latRoad
true,Ix6x2,0,intersection
false,Rx6x3,35,latRoad
true,Ix6x4,0,intersection
false,Rx6x5,35,latRoad
true,Ix6x6,0,intersection
```

```
false ,Rx6x7 ,35 , latRoad
true , Ix6x8 ,0 , intersection
false ,Rx6x9 ,35 , latRoad
true , Ix6x10 ,0 , intersection
false ,Rx6x11 ,35 , latRoad
true , Ix6x12 ,0 , intersection
false ,Rx6x13 ,35 , latRoad
true , Ix6x14 ,0 , intersection
false ,Rx6x15 ,35 , latRoad
true , Ix6x16 ,0 , intersection
false ,Rx6x17 ,35 , latRoad
true , Ix6x18 ,0 , intersection
false ,Rx7x0 ,35 , ladRoad
false ,Rx7x2 ,35 , ladRoad
false ,Rx7x4 ,35 , ladRoad
false ,Rx7x6 ,35 , ladRoad
false ,Rx7x8 ,35 , ladRoad
false ,Rx7x10 ,35 , ladRoad
false ,Rx7x12 ,35 , ladRoad
false ,Rx7x14 ,35 , ladRoad
false ,Rx7x16 ,35 , ladRoad
false ,Rx7x18 ,35 , ladRoad
true , Ix8x0 ,0 , intersection
false ,Rx8x1 ,35 , latRoad
true , Ix8x2 ,0 , intersection
false ,Rx8x3 ,35 , latRoad
true , Ix8x4 ,0 , intersection
false ,Rx8x5 ,35 , latRoad
true , Ix8x6 ,0 , intersection
false ,Rx8x7 ,35 , latRoad
true , Ix8x8 ,0 , intersection
false ,Rx8x9 ,35 , latRoad
```

```
true,Ix8x10,0,intersection
false,Rx8x11,35,latRoad
true,Ix8x12,0,intersection
false,Rx8x13,35,latRoad
true,Ix8x14,0,intersection
false,Rx8x15,35,latRoad
true,Ix8x16,0,intersection
false,Rx8x17,35,latRoad
true,Ix8x18,0,intersection
false,Rx9x0,35,ladRoad
false,Rx9x2,35,ladRoad
false,Rx9x4,35,ladRoad
false,Rx9x6,35,ladRoad
false,Rx9x8,35,ladRoad
false,Rx9x10,35,ladRoad
false,Rx9x12,35,ladRoad
false,Rx9x14,35,ladRoad
false,Rx9x16,35,ladRoad
false,Rx9x18,35,ladRoad
true,Ix10x0,0,intersection
false,Rx10x1,35,latRoad
true,Ix10x2,0,intersection
false,Rx10x3,35,latRoad
true,Ix10x4,0,intersection
false,Rx10x5,35,latRoad
true,Ix10x6,0,intersection
false,Rx10x7,35,latRoad
true,Ix10x8,0,intersection
false,Rx10x9,35,latRoad
true,Ix10x10,0,intersection
false,Rx10x11,35,latRoad
true,Ix10x12,0,intersection
```

```
false ,Rx10x13 ,35,latRoad
true ,Ix10x14 ,0,intersection
false ,Rx10x15 ,35,latRoad
true ,Ix10x16 ,0,intersection
false ,Rx10x17 ,35,latRoad
true ,Ix10x18 ,0,intersection
false ,Rx11x0 ,35,ladRoad
false ,Rx11x2 ,35,ladRoad
false ,Rx11x4 ,35,ladRoad
false ,Rx11x6 ,35,ladRoad
false ,Rx11x8 ,35,ladRoad
false ,Rx11x10 ,35,ladRoad
false ,Rx11x12 ,35,ladRoad
false ,Rx11x14 ,35,ladRoad
false ,Rx11x16 ,35,ladRoad
false ,Rx11x18 ,35,ladRoad
true ,Ix12x0 ,0,intersection
false ,Rx12x1 ,35,latRoad
true ,Ix12x2 ,0,intersection
false ,Rx12x3 ,35,latRoad
true ,Ix12x4 ,0,intersection
false ,Rx12x5 ,35,latRoad
true ,Ix12x6 ,0,intersection
false ,Rx12x7 ,35,latRoad
true ,Ix12x8 ,0,intersection
false ,Rx12x9 ,35,latRoad
true ,Ix12x10 ,0,intersection
false ,Rx12x11 ,35,latRoad
true ,Ix12x12 ,0,intersection
false ,Rx12x13 ,35,latRoad
true ,Ix12x14 ,0,intersection
false ,Rx12x15 ,35,latRoad
```

```
true , Ix12x16 ,0 , intersection
false , Rx12x17 ,35 , latRoad
true , Ix12x18 ,0 , intersection
false , Rx13x0 ,35 , ladRoad
false , Rx13x2 ,35 , ladRoad
false , Rx13x4 ,35 , ladRoad
false , Rx13x6 ,35 , ladRoad
false , Rx13x8 ,35 , ladRoad
false , Rx13x10 ,35 , ladRoad
false , Rx13x12 ,35 , ladRoad
false , Rx13x14 ,35 , ladRoad
false , Rx13x16 ,35 , ladRoad
false , Rx13x18 ,35 , ladRoad
true , Ix14x0 ,0 , intersection
false , Rx14x1 ,35 , latRoad
true , Ix14x2 ,0 , intersection
false , Rx14x3 ,35 , latRoad
true , Ix14x4 ,0 , intersection
false , Rx14x5 ,35 , latRoad
true , Ix14x6 ,0 , intersection
false , Rx14x7 ,35 , latRoad
true , Ix14x8 ,0 , intersection
false , Rx14x9 ,35 , latRoad
true , Ix14x10 ,0 , intersection
false , Rx14x11 ,35 , latRoad
true , Ix14x12 ,0 , intersection
false , Rx14x13 ,35 , latRoad
true , Ix14x14 ,0 , intersection
false , Rx14x15 ,35 , latRoad
true , Ix14x16 ,0 , intersection
false , Rx14x17 ,35 , latRoad
true , Ix14x18 ,0 , intersection
```

```
false,Rx15x0,35,ladRoad
false,Rx15x2,35,ladRoad
false,Rx15x4,35,ladRoad
false,Rx15x6,35,ladRoad
false,Rx15x8,35,ladRoad
false,Rx15x10,35,ladRoad
false,Rx15x12,35,ladRoad
false,Rx15x14,35,ladRoad
false,Rx15x16,35,ladRoad
false,Rx15x18,35,ladRoad
true,Ix16x0,0,intersection
false,Rx16x1,35,latRoad
true,Ix16x2,0,intersection
false,Rx16x3,35,latRoad
true,Ix16x4,0,intersection
false,Rx16x5,35,latRoad
true,Ix16x6,0,intersection
false,Rx16x7,35,latRoad
true,Ix16x8,0,intersection
false,Rx16x9,35,latRoad
true,Ix16x10,0,intersection
false,Rx16x11,35,latRoad
true,Ix16x12,0,intersection
false,Rx16x13,35,latRoad
true,Ix16x14,0,intersection
false,Rx16x15,35,latRoad
true,Ix16x16,0,intersection
false,Rx16x17,35,latRoad
true,Ix16x18,0,intersection
false,Rx17x0,35,ladRoad
false,Rx17x2,35,ladRoad
false,Rx17x4,35,ladRoad
```

```
false ,Rx17x6 ,35 , ladRoad
false ,Rx17x8 ,35 , ladRoad
false ,Rx17x10 ,35 , ladRoad
false ,Rx17x12 ,35 , ladRoad
false ,Rx17x14 ,35 , ladRoad
false ,Rx17x16 ,35 , ladRoad
false ,Rx17x18 ,35 , ladRoad
true , Ix18x0 ,0 , intersection
false ,Rx18x1 ,40 , latRoad
true , Ix18x2 ,0 , intersection
false ,Rx18x3 ,40 , latRoad
true , Ix18x4 ,0 , intersection
false ,Rx18x5 ,40 , latRoad
true , Ix18x6 ,0 , intersection
false ,Rx18x7 ,40 , latRoad
true , Ix18x8 ,0 , intersection
false ,Rx18x9 ,40 , latRoad
true , Ix18x10 ,0 , intersection
false ,Rx18x11 ,40 , latRoad
true , Ix18x12 ,0 , intersection
false ,Rx18x13 ,40 , latRoad
true , Ix18x14 ,0 , intersection
false ,Rx18x15 ,40 , latRoad
true , Ix18x16 ,0 , intersection
false ,Rx18x17 ,40 , latRoad
true , Ix18x18 ,0 , intersection
```

A.2 ECJ Optimizer

```
package ec.app.trafficSim;


import ec.*;
import ec.simple.*;
import ec.vector.*;


import java.util.ArrayList;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileWriter;
import java.io.BufferedWriter;
import java.io.IOException;


import ec.app.trafficSim.sim.io.ECJPlug;
/**
 * Method called by ECJ Evaluate method to solve for the optimal
    fitness
 *@author
 *    Adam Wechter
 */
public class TrafficSim extends Problem implements SimpleProblemForm
    {
  public void evaluate(final EvolutionState state, final Individual
     ind, final int subpopulation, final int threadnum) {


    File fnew = new File("/Users/wechtera/Desktop/FinalComp/ecj/ec/
       app/trafficSim/sim/files/Timings.txt");
    if(fnew.exists()) {
      System.out.println("DELETING EXISTING TIMINGS FILE");
      fnew.delete();
```

```
}
if(ind.evaluated)
  return;

if(!(ind instanceof BitVectorIndividual))
  state.output.fatal("Not a BitVector individual!!!",null);

double thisFitness = 0;
BitVectorIndividual ind2 = (BitVectorIndividual)ind;
ArrayList<int []> timings = new ArrayList<int []>();

System.out.println(ind2.genotypeToStringForHumans());
for(int i = 0; i<ind2.genome.length;) {
  int [] temp = new int[8];
  for(int j = 0; j < 8; j++) {

    int x;
    if(ind2.genome[i] == true)
      x = 1;
    else
      x = 0;

    temp[j] = x;
    i++;
  }

  timings.add(temp);
}

StringBuilder sb = new StringBuilder("");
for(int i = 0; i < timings.size(); i++) {
  sb.append(timings.get(i)[0]+","+timings.get(i)[1]);
```

```
    sb.append(timings.get(i)[2]);
    sb.append(timings.get(i)[3]+","+timings.get(i)[4]);
    sb.append(timings.get(i)[5]);
    sb.append(timings.get(i)[6]+","+timings.get(i)[7]+"\n");
  }
  try {
    fnew.createNewFile();
    FileWriter writer = new FileWriter(fnew,false);
    BufferedWriter buff = new BufferedWriter(writer);
    System.out.println(sb.toString());
    buff.write(sb.toString());
    buff.close();
    writer.close();
  } catch(IOException e) {
    System.out.println("Problem Writing to Timings");
    System.out.println("File Name: " + fnew.getAbsolutePath());
    e.printStackTrace();
  }

  try {
  thisFitness = ECJPlug.evaluateFitness("/Users/wechtera/Desktop/
     FinalComp/ecj/ec/app/trafficSim/sim/files/Node.txt", "/Users/
     wechtera/Desktop/FinalComp/ecj/ec/app/trafficSim/sim/files/
     Edge.txt", "/Users/wechtera/Desktop/FinalComp/ecj/ec/app/
     trafficSim/sim/files/Cars1.txt",.1,.1,.1, "/Users/wechtera/
     Desktop/FinalComp/ecj/ec/app/trafficSim/sim/files/Timings.txt
     ");
  } catch (FileNotFoundException e) {
    e.printStackTrace();
    state.output.fatal("Could Not Find File");
  } catch (IOException e) {
    e.printStackTrace();
```

```
      state.output.fatal("Problem with input");
    }


    ((SimpleFitness)ind2.fitness).setFitness(state, (float)
       thisFitness, false);  //setfit(evstate, fitness, is it ideal
       ?)
     ind2.evaluated = true;


  }
}
```

```
breedthreads = 1
evalthreads  = 1
seed.0 = 1

state      = ec.simple.SimpleEvolutionState
pop      = ec.Population
init     = ec.simple.SimpleInitializer
finish     = ec.simple.SimpleFinisher
breed      = ec.simple.SimpleBreeder
eval     = ec.simple.SimpleEvaluator
stat     = ec.simple.SimpleStatistics
exch     = ec.simple.SimpleExchanger

generations         = 20
quit-on-complete      = true
checkpoint          = false
checkpoint-prefix       = ec
checkpoint-modulo       = 1

stat.file         = $out.stat

pop.subpops         = 1
pop.subpop.0        = ec.Subpopulation
pop.subpop.0.size       = 10
pop.subpop.0.duplicate-retries  = 0
pop.subpop.0.species      = ec.vector.BitVectorSpecies
pop.subpop.0.species.fitness    = ec.simple.SimpleFitness
pop.subpop.0.species.ind    = ec.vector.BitVectorIndividual
pop.subpop.0.species.genome-size = 800

pop.subpop.0.species.crossover-type = two
pop.subpop.0.species.crossover-prob = .5
pop.subpop.0.species.mutation-prob  = 0.02

pop.subpop.0.species.pipe       = ec.vector.breed.
   VectorMutationPipeline
pop.subpop.0.species.pipe.source.0    = ec.vector.breed.
   VectorCrossoverPipeline
pop.subpop.0.species.pipe.source.0.source.0   = ec.select.
   TournamentSelection
pop.subpop.0.species.pipe.source.0.source.1   = ec.select.
   TournamentSelection

select.tournament.size = 2

eval.problem  = ec.app.trafficSim.TrafficSim
```

A.3 Support Programs

```
import java.io.File;
import java.io.BufferedWriter;
import java.io.FileReader;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.FileWriter;
/**
 * Code to generate TrafficSim.java files and Parameter files
 *@author
 *    Adam Wechter
 */
public class ParamsGen {
  public static void main(String [] args) {
    StringBuilder indexSB = new StringBuilder("");
    StringBuilder paramsSB = new StringBuilder("");

    String nodeTxtFile = "ecj/ec/app/trafficSim/sim/files/Nodes";
    String edgeTxtFile = "ecj/ec/app/trafficSim/sim/files/Edge";
    String carTxtFile = "ecj/ec/app/trafficSim/sim/files/Cars";
    String paramsName = "trafficSim";
    String javaName = "TrafficSim";
    String outputStat = "$out";
    final String javaTxt = "package ec.app.trafficSim;\nimport ec
       .*;\nimport ec.simple.*;\nimport ec.vector.*;\nimport java.
       util.ArrayList;\nimport java.io.File;\nimport java.io.
       FileNotFoundException;\nimport java.io.FileWriter;\nimport
       java.io.BufferedWriter;\nimport java.io.IOException;\nimport
       ec.app.trafficSim.sim.io.ECJPlug;\npublic class ";
```

```
final String javaTxt6 = " extends Problem implements
   SimpleProblemForm {\npublic void evaluate(final
   EvolutionState state, final Individual ind, final int
   subpopulation, final int threadnum) {\nSystem.out.println
   (\"\\n\\n----------NEW RUN------------\");\nFile fnew = new
   File(\"/Users/wechtera/Desktop/FinalComp/ecj/ec/app/
   trafficSim/sim/files/Timings.txt\");\nif(fnew.exists()) {\
   nSystem.out.println(\"DELETING EXISTING TIMINGS FILE\");\
   nfnew.delete();\n}\nif(ind.evaluated)\nreturn;\nif(!(ind
   instanceof BitVectorIndividual))\nstate.output.fatal(\"Not a
   BitVector individual!!!\",null);\ndouble thisFitness = 0;\
   nBitVectorIndividual ind2 = (BitVectorIndividual)ind;\
   nArrayList<int []> timings = new ArrayList<int []>();\nSystem
   .out.println(ind2.genotypeToStringForHumans());\nfor(int i =
   0; i<ind2.genome.length;) {\nint [] temp = new int[8];\nfor(
   int j = 0; j < 8; j++) {\nint x;\nif(ind2.genome[i] == true)\
   nx = 1;\nelse\nx = 0;\ntemp[j] = x;\ni++;\n}\ntimings.add(
   temp);\n}\nStringBuilder sb = new StringBuilder(\"\");\nfor(
   int i = 0; i < timings.size(); i++) {\nsb.append(timings.get(
   i)[0]+\",\"+timings.get(i)[1]);\nsb.append(timings.get(i)[2])
   ;\nsb.append(timings.get(i)[3]+\",\"+timings.get(i)[4]);\nsb.
   append(timings.get(i)[5]);\nsb.append(timings.get(i)
   [6]+\",\"+timings.get(i)[7]+\"\\n\");\n}\ntry {\nfnew.
   createNewFile();\nFileWriter writer = new FileWriter(fnew,
   false);\nBufferedWriter buff = new BufferedWriter(writer);\
   nSystem.out.println(sb.toString());\nbuff.write(sb.toString()
   );\nbuff.close();\nwriter.close();\n} catch(IOException e) {\
   nSystem.out.println(\"Problem Writing to Timings\");\nSystem.
   out.println(\"File Name: \" + fnew.getAbsolutePath());\ne.
   printStackTrace();\n}\ntry {\nthisFitness = ECJPlug.
   evaluateFitness(\""; //NodeFile
final String javaTxt2 = "\", \""; //edge   cars
final String javaTxt3 = "\", ";   //    modifier1
final String javaTxt4 = ", ";     //modifier2    modifier3
final String javaTxt5 = ", \"/Users/wechtera/Desktop/FinalComp/
   ecj/ec/app/trafficSim/sim/files/Timings.txt\");\n} catch (
   FileNotFoundException e) {\ne.printStackTrace();\nstate.
   output.fatal(\"Could Not Find File\");\n} catch (IOException
   e) {\ne.printStackTrace();\nstate.output.fatal(\"Problem with
    input\");\n}\n((SimpleFitness)ind2.fitness).setFitness(state
   , (float)thisFitness, false); //setfit(evstate, fitness, is
   it ideal?)\nind2.evaluated = true;\nSystem.out.println
   (\"----------- RUN END ------------\\n\\n\");\n}\n}";

final String paramsTxt = "breedthreads = 1\nevalthreads  = 1\
   nseed.0 = "; //Seed Number here
final String paramsTxt2 = "\nstate = ec.simple.
   SimpleEvolutionState\npop = ec.Population\ninit = ec.simple.
   SimpleInitializer\nfinish = ec.simple.SimpleFinisher\nbreed =
    ec.simple.SimpleBreeder\neval = ec.simple.SimpleEvaluator\
   nstat = ec.simple.SimpleStatistics\nexch = ec.simple.
   SimpleExchanger\ngenerations = "; //Generations
final String paramsTxt3 = "\nquit-on-complete = true\ncheckpoint
    = false\ncheckpoint-prefix = ec\ncheckpoint-modulo = 1\nstat
   .file "; //Stat File
final String paramsTxt4 = "\npop.subpops = 1\npop.subpop.0 = ec.
   Subpopulation\npop.subpop.0.size = "; //Pop Size
final String paramsTxt5 = "\npop.subpop.0.duplicate-retries = 0\
   npop.subpop.0.species = ec.vector.BitVectorSpecies\npop.
   subpop.0.species.fitness = ec.simple.SimpleFitness\npop.
   subpop.0.species.ind = ec.vector.BitVectorIndividual\npop.
   subpop.0.species.genome-size = "; //genomeSize
final String paramsTxt6 = "\npop.subpop.0.species.crossover-type
    = two\npop.subpop.0.species.crossover-prob = "; //
   CrossOverProb
final String paramsTxt7 = "\npop.subpop.0.species.mutation-prob
    = "; //Mutation Prob
final String paramsTxt8 = "\npop.subpop.0.species.pipe = ec.
   vector.breed.VectorMutationPipeline\npop.subpop.0.species.
   pipe.source.0 = ec.vector.breed.VectorCrossoverPipeline\npop.
   subpop.0.species.pipe.source.0.source.0 = ec.select.
   TournamentSelection\npop.subpop.0.species.pipe.source.0.
   source.1 = ec.select.TournamentSelection\nselect.tournament.
   size = 2\neval.problem = ec.app.trafficSim."; //Java File
   Name


int testNum = 0;
int [] seedNumList =
   {923,866,288,353,782,234,78,510,603,291,809,810,835,855,528,755,642,823,3


//Set up heading for index

indexSB.append("testNum,genNum,popSize,crossProb,mutProb,
   modifiers,seedNum\n");

OuterLoop:
for(int genNum = 20; genNum < 121; genNum+=20) {
  for(int popSize = 25; popSize < 61; popSize+= 20) {
      for(double crossProb = 0; crossProb <= .8; crossProb+=0.2)
          {
        for(double mutProb = 0; mutProb < .5; mutProb+=0.1) {
          for(int modifiers = 0; modifiers < 4; modifiers++) {
            for(int seedNum = 0; seedNum < 20; seedNum++) {
```

```
  String tempNodeTxtFile = "/Users/wechtera/Desktop/
     FinalComp/ecj/ec/app/trafficSim/sim/files/Node.
     txt";
  String tempEdgeTxtFile = "/Users/wechtera/Desktop/
     FinalComp/ecj/ec/app/trafficSim/sim/files/Edge.
     txt";
  String tempCarTxtFile = "/Users/wechtera/Desktop/
     FinalComp/ecj/ec/app/trafficSim/sim/files/Cars.
     txt";
  String tempParamsName;
  String tempJavaName;
  String tempOutPutStat;
  double modifier1;
  double modifier2;
  double modifier3;

//Do modifiers here
  /*
  * if 1, then favors modifier 1, 2 favors 2, 3
     favors 3 0 has no favorite
  */
  if(modifiers == 0) {
    modifier1 = .33;
    modifier2 = .33;
    modifier3 = .33;
  }
  else if (modifiers == 1) {
    modifier1 = .66;
    modifier2 = .33;
    modifier3 = .33;
  }
  else if (modifiers == 2) {
        modifier1 = .33;
        modifier2 = .66;
        modifier3 = .33;
      }
      else {
        modifier1 = .33;
        modifier2 = .33;
        modifier3 = .66;
      }

    //GENERATE File Names
      StringBuilder sb0 = new StringBuilder(paramsName);
      sb0.append(testNum + ".params");
      tempParamsName = sb0.toString();

      sb0 = new StringBuilder(javaName);
      sb0.append(testNum + ".java");
      tempJavaName = sb0.toString();

      sb0 = new StringBuilder(outputStat);
      sb0.append(testNum + ".stat");
      tempOutPutStat = sb0.toString();



      int totalNodeNumber = 800;





    //build java
      //javatxt + node + javatxt2 + edge + javatext2 +
          cars + javatxt3 + mod1 + javatxt4 + mod2 +
          javatext4 + mod3 + javatxt5
      sb0 = new StringBuilder(javaTxt);
      sb0.append(javaName + testNum);
      sb0.append(javaTxt6);
      sb0.append(tempNodeTxtFile);
      sb0.append(javaTxt2);
      sb0.append(tempEdgeTxtFile);
      sb0.append(javaTxt2);
      sb0.append(tempCarTxtFile);
      sb0.append(javaTxt3);
      sb0.append(modifier1);
      sb0.append(javaTxt4);
      sb0.append(modifier2);
      sb0.append(javaTxt4);
      sb0.append(modifier3);
      sb0.append(javaTxt5);
    //Writing the java file
      File newFile = new File(tempJavaName);
      if(newFile.exists()) {
        System.out.println("DELETING EXISTING JAVA FILE"
           );
        newFile.delete();
      }

      try {
        newFile.createNewFile();
        FileWriter writer = new FileWriter(newFile,false
           );
        BufferedWriter buff = new BufferedWriter(writer)
           ;
        buff.write(sb0.toString());
        buff.close();
        writer.close();
      } catch(IOException e) {
        System.out.println("Problem Writing to Timings")
           ;
        System.out.println("File Name: " + newFile.
           getAbsolutePath());
        e.printStackTrace();
      }

    //writing params file

      sb0 = new StringBuilder(paramsTxt);
      sb0.append(seedNumList[seedNum]);
      sb0.append(paramsTxt2);
      sb0.append(genNum);
      sb0.append(paramsTxt3);
      sb0.append(tempOutPutStat);
      sb0.append(paramsTxt4);
      sb0.append(popSize);
      sb0.append(paramsTxt5);
      sb0.append(totalNodeNumber);
      sb0.append(paramsTxt6);
      sb0.append(crossProb);
      sb0.append(paramsTxt7);
      sb0.append(mutProb);
      sb0.append(paramsTxt8);
      sb0.append(javaName+testNum);



      newFile = new File(tempParamsName);
      if(newFile.exists()) {
        System.out.println("DELETING EXISTING PARAMS
           FILE");
        newFile.delete();
      }
      try {
        newFile.createNewFile();
        FileWriter writer = new FileWriter(newFile,false
           );
        BufferedWriter buff = new BufferedWriter(writer)
           ;
        buff.write(sb0.toString());
        buff.close();
        writer.close();
      } catch(IOException e) {
        System.out.println("Problem Writing to Params");
        System.out.println("File Name: " + newFile.
           getAbsolutePath());
        e.printStackTrace();
      }

    //Writing Index
      indexSB.append(testNum+","+genNum+","+popSize+","+
         crossProb+","+mutProb+","+modifiers+","+
         seedNumList[seedNum]+"\n");
    //java file list
      paramsSB.append(tempParamsName + "\n");




      testNum++;
    }
```

```
          }
        }



    }
    break OuterLoop;  //end here because pop and gen num do not
        influence enough
  }
}

//Writing index file

File indexFile = new File("fileIndex6.txt");
if(indexFile.exists()) {
  System.out.println("DELETING EXISTING INDEX FILE");
  indexFile.delete();
}
try {
  indexFile.createNewFile();
  FileWriter writer = new FileWriter(indexFile,false);
  BufferedWriter buff = new BufferedWriter(writer);
  buff.write(indexSB.toString());
  buff.close();
  writer.close();
} catch(IOException e) {
  System.out.println("Problem Writing to index file");
  System.out.println("File Name: " + indexFile.getAbsolutePath()
     );
  e.printStackTrace();
}
indexFile = new File("paramsIndex7.txt");
if(indexFile.exists()) {
      System.out.println("DELETING EXISTING INDEX FILE");
      indexFile.delete();
    }
    try {
      indexFile.createNewFile();
      FileWriter writer = new FileWriter(indexFile,false);
      BufferedWriter buff = new BufferedWriter(writer);
      buff.write(paramsSB.toString());
      buff.close();
      writer.close();
    } catch(IOException e) {
      System.out.println("Problem Writing to index file");
      System.out.println("File Name: " + indexFile.getAbsolutePath()
         );
      e.printStackTrace();
    }
  }


}
```

Bibliography

[1] Minal K. Avzekar and Amruta Moon. Scats: The sydney co-ordinated adaptive traffic system-principles, methodology, algorithms. pages 30–32, 2014.

[2] Halim Ceylan and Michael G.H Bell. Traffic signal timing optimisation based on genetic algorithm approach, including drivers routing. *Transportation Research Part B: Methodological*, 38(4):329 – 342, 2004.

[3] Eric Dumbugh. Rethinking the economics of traffic congestion. 2012. `http://www.theatlanticcities.com/commute/2012/06/defense-congestion/2118/`. Accessed 7 May 2014.

[4] Michel Ferreira, Ricardo Fernandes, Hugo Conceição, Wantanee Viriyasitavat, and Ozan K. Tonguz. Self-organized traffic control. In *Proceedings of the Seventh ACM International Workshop on VehiculAr InterNETworking*, VANET '10, pages 85–90, New York, NY, USA, 2010. ACM.

[5] Sean Luke. *The ECJ Owners Manual.* George Mason University, 2013.

[6] Sean Luke. *Essentials of Metaheuristics.* Lulu, second edition, 2013. Available for free at http://cs.gmu.edu/∼sean/book/metaheuristics/.

[7] As'ad Salkham, Raymond Cunningham, Anurag Garg, and Vinny Cahill. A collaborative reinforcement learning approach to urban traffic control optimization. In *Proceedings of the 2008 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology - Volume 02*, WI-IAT '08, pages 560–566, Washington, DC, USA, 2008. IEEE Computer Society.

[8] Dobinson K.W. Sims, A.G. The sydney coordinated adaptive traffic (scat) system: Philosophy and benefits. pages 29–35, 1980.

[9] Siran Tao Yan Li, Lijie Yu and Kuanmin Chen. Multi-objective optimization of traffic signal timing for oversaturated intersection. In *Mathematical Problems in Engineering*, volume 2013, page 9, 1999.

[10] Ilsoo Yun and Byungkyu "Brian" Park. Application of stochastic optimization method for an urban corridor. In *Proceedings of the 38th Conference on Winter Simulation*, WSC '06, pages 1493–1499. Winter Simulation Conference, 2006.