The Truckin' Project
Experimenting with Genetic Algorithms

Debbie Papoulis

A Thesis

in

The Department

of

Computer Science

Presented in Partial Fulfillment of the Requirements
for the Degree of Master of Computer Science at
Concordia University
Montreal, Quebec, Canada

October 2001

Abstract

The Truckin' Project
Experimenting with Genetic Algorithms

Debbie Papoulis

Darwin's theory of evolution says that the more suitable an individual is to his environment, the more likely that individual is to reproduce. Conversely, the less suited an individual is to his environment the less likely he is to reproduce. Consequently, by the laws of heredity, the fitness of the subsequent generation, as a whole, should be greater than the last.

The Truckin' project attempts to use these theories and laws to prove that a program can also evolve. Using evolutionary programming and object oriented techniques, Truckin' simulates a world in which trucks buy and sell commodities competing for the best deals. The most successful of these trucks are allowed to 'reproduce' and compete in the next generation. Over a period of time the truck population should converge to an overall fitter population.

In this thesis the basics of genetic algorithms are explained and then the Truckin' project is described in detail. Finally the results and current status of the project are outlined.

Acknowledgements

I would like to thank my supervisor, Dr. Peter Grogono, for his guidance, support and infinite patience.

I would also like to thank the people at school, work and especially home whose support helped me accomplish this thesis.

Table of Contents

Table of Figures

Table of Tables

1. Introduction

Researchers working in the area of Genetic Algorithms try to use biological evolution as a model for the structure of the code of their programs. They try to mimic real world evolution in the hope that the outcome of the algorithm will parallel the outcome of nature. In the Truckin' simulation we attempted to mimic an economic world in which the successful players will survive long enough by trading to be chosen to reproduce. With this pattern in mind we expected that the results of the simulation would eventually converge to 'strong' solutions and eliminate 'weak' ones.

Truckin' is a simulation in which a square country of ten avenues and ten streets is modeled. On several of these intersections there exist dealers that can buy and sell commodities. Roaming around the country there are trucks whose mission is to buy commodities from one dealer at a low price and sell them to another at a higher price in order to make a profit. Each truck has a genetic combination that is made up of eight genes that determine the specific strategies used by the truck to make a profit. The goal of the simulation is to use the theory of Genetic Algorithms in order to display convergence to the optimal solution, i.e., the most profitable truck.

In Chapter 2 we will look first at the background of Genetic Algorithms. We will define terms such as *fitness function* and *selection mechanism*. Once we

have defined the terms, we will go through an example of a simple genetic algorithm to make it all clear.

In Chapter 3 we will explain the Truckin' simulation. We will outline all the players and the rules that they play by. We also explain the economic world that we have simulated and how genetic techniques should produce the optimal solution.

In Chapter 4 we will explain the Design of Truckin'. We will outline how the terms described in Chapter 2 fit into the Truckin' simulation and why we think that it should work.

In Chapter 5 we will describe how Truckin' was implemented. We look at the details of the implementation and how dynamic binding is used to produce the different make-up of each of the competing trucks.

In Chapter 6 we look at the results obtained by the simulation. We see how the different parameters of the simulation can affect the results produced and also the amount of convergence visible.

In Chapter 7 we will draw some conclusions of what went right and what went wrong in the simulation and finally in Chapter 8 we will speculate as to what

future work can be done with the simulation to improve the results and the amount of convergence visible.

2. Background

2.1 Definition

Genetic algorithms are an abstraction of biological evolution [1]. They employ the theory of natural selection to find good solutions for optimization problems.

The basic principle behind genetic algorithms is to do what nature does. In nature, individuals best suited to their environment survive long enough to reproduce, while the weaker ones die before being able to, a phenomenon known as *the survival of the fittest*. Nature has encoded each individual's characteristics into a chromosome that serves as the blueprint of that individual. A chromosome can be divided into genes, each encoding one trait of the individual. Competition for sparse resources, such as food, results in the stronger individuals overpowering the weaker ones thus surviving while the weaker die, a process knows as *natural selection*. As a direct result of natural selection, the genes of the stronger individual live on in their offspring, while those of the weaker ones do not, thus implicitly generating a second generation that is even better suited to their environment [2].

Genetic algorithms mimic this phenomenon by encoding each possible solution to an optimization problem into a string analogous to the chromosome.

The elements of the string represent a specific characteristic of the solution analogous to the gene. The genetic algorithm then repeatedly manipulates the most promising strings in its search until a sufficiently good solution has been found or some other termination condition is met. Operators such as selection, crossover and mutation are employed in the search for the optimal solution (see Section 2.3).

2.2 History

Genetic Algorithms were formally introduced in the 1970s by John Holland at the University of Michigan. They were subsequently studied by De Jong, Goldberg and others. Holland's original goal was not to design a solution to specific optimization problems, but instead to study the phenomenon of adaptation and its possible applications to computer systems [3].

Before 1980s most of the research done in genetic algorithms was theoretical. Hollstien and De Jong were the most active, with Hollstein's work providing an analysis of the effect of selection and De Jong's work focusing on the features of genetic algorithms that would lead to a robust search procedure. In the early 1980s experiments with large applications were developed across many disciplines. From this work came new and important findings about robustness and applicability of genetic algorithms [4].

2.3 Introduction to Simple Genetic Algorithms

The simplest of genetic algorithms work with a population of binary strings. Each string is an encoded version of a possible solution to the problem. Each solution is evaluated according to some fitness function. The fittest of the solutions are selected to contribute to the next generation of solutions. The selected solutions are recombined using crossover of the strings and then mutation is applied to produce the next generation. The next generation is then evaluated and cycles through the same process until the termination condition is met.

Therefore the seven basic components of a genetic algorithm are: an encoding mechanism, a fitness function, a selection mechanism, crossover, mutation, a generation cycle and a termination condition.

Table 1 maps each component of a simple genetic algorithm to a step in the basic structure.

Algorithm Component	Algorithm Step
Encoding Mechanism	1. Create an initial population
Fitness Function	2. Evaluate each individual in the population
Selection	3. Select the individuals best suited for the problem at hand.
Crossover	4. Generate a new population by recombining the original population.
Mutation	5. Apply mutation to the new population.
Generation Cycle and Termination Condition	6. Go to step 2 and repeat until the termination condition is met.

Table 1: Algorithm Components and Associated Steps

2.3.1 Encoding Mechanism

The solution to an optimization problem may consist of a discrete value, or it may assume continuous values. The encoding mechanism maps each solution to a unique string. The different variations of the string make up the search space and the goal of the genetic algorithm is to intelligently traverse the search space until it has found a satisfactory solution or ultimately the optimal solution.

2.3.2 Fitness Function

The fitness function provides a mechanism for evaluating each encoded solution to determine how well the chromosome solves the problem at hand. The value of the fitness function will determine if the chromosome did well enough to survive the current generation and produce offspring for the next.

2.3.3 Selection

Selection is the method used to implement nature's survival of the fittest mechanism. The solutions with the highest fitness values survive and produce offspring, while the weaker ones die. The selection strategy takes into consideration the proportion of solutions that survive onto the next generation and the amount of offspring each solution is allowed to produce.

2.3.4 Crossover

To recombine the genes of two successful solutions, crossover is used. In its simplest form, a single, randomly selected crossover point is chosen and the genes of the two parent chromosomes are switched at that point to create two new offspring. More complex forms of crossover may include multiple crossover points and/or a probability factor whether crossover should take place at all.

2.3.5 Mutation

To ensure proper searching of the search space, mutation is used. Mutation of a bit involves flipping it from a 0 to a 1 or vice versa. In so doing, the

genetic algorithm can regenerate lost bits of the search space if all strings in the population have converged to the wrong value.

2.3.6 Generation Cycle

A single generation cycle in a genetic algorithm is defined by one cycle of creation - evaluation - selection - manipulation phases. Each cycle examines a set of solutions, known as a population, and selects the fittest among them.

2.3.7 Termination Condition

The termination condition dictates when the algorithm should stop searching. Typical examples of termination conditions include a satisfactory solution has been located or a predefined number of generations have been searched.

2.3.8 Simple Genetic Algorithm Example

To illustrate a simple genetic algorithm the example of optimizing the function $f(x) = x^2$ over the interval [0-31] will be used [5].

The first step is to encode the possible values of x into a binary string. In this example, we will use a binary string of length 5 [00000-11111]. Next we generate an initial population of 4 individuals as shown in Table 2. The strength, or fitness function, of the chromosome is calculated as the percentage of the sum of all f(x). This initial population is then evaluated according to the fitness

function and the best solutions are chosen to survive and reproduce for the next generation.

Chromosome	x	f(x)	Strength
01101	13	169	14.4
11000	24	576	49.2
01000	8	64	5.5
10011	19	361	30.9

Table 2: Example Population and Fitness Values

Chromosome 11000 has a large fitness value, therefore it is allowed two copies in the mating pool for the next generation. Chromosomes 01101 and 10011 are each allowed one copy and since chromosome 01000 has a very small fitness value, it is not allowed any copies and therefore dies.

The next step is to construct the new population by applying crossover. There are two factors in applying crossover. The first being the crossover probability, whether crossover is to occur on a pair of strings and the second is the randomly selected crossover point. In this example both pairs of chromosomes will have crossover applied to them, the first with a crossover point of 5 and the second with a crossover point of 3. Table 3 displays the status of the population after crossover is applied, but before mutation.

Parent Chromosomes	Swapping	New Population
01101 11000	0110[1] 1100[0]	01100 11001
11000 10011	11[000] 10[011]	11011 10000

Table 3: Creation of New Population

("[ ]" Indicates crossover).

The next step is to apply mutation to the new population. Again, there are two factors in applying mutation, the first being whether mutation should be applied and the second being to which bit position should the mutation be applied. In this example we will apply mutation to the first chromosome of the new population in the first bit position. Table 4 displays the second generation and its fitness values.

Chromosome	x	f(x)	Strength
11100	28	784	32.7
11001	25	625	26.1
11011	27	729	30.5
10000	16	256	10.7

Table 4: Fitness of New Population

As we can see, the overall strength of the population has gone up from the first generation to the second. The algorithm continues like this until a predefined number of generations have been examined (say 50) or until the strength of the population reaches an acceptable level.

2.4 Why do Genetic Algorithms Work?

The *schema theory* and the *building block hypothesis* explain the essence of the workings of genetic algorithms.

A schema represents a subset of all possible binary strings of a certain length that have the same bit values in certain positions. For example, the schema 1**1 represents all strings of length four that have a one in the first and last bit positions. The members of this schema are: 1001, 1011, 1101 and 1111.

The positions in the schema that are not allocated a value, and therefore represented by a *, are called wild cards, and conversely, the positions in the schema that are allocated a value (0 or 1) are called fixed positions. The number of fixed positions in a schema is called the order of the schema (the order of 1**1 is 2).

A schema represents a subset of all strings that follow a certain pattern. We can therefore allocate a fitness value to a schema. The fitness value of a schema is the average fitness value of the members of the current population that are instances of that schema. Therefore, the fitness value of a schema can change from generation to generation as the make up of the population changes.

The Schema Theorem [6] describes the growth of a schema from one generation to the next. It implies that short, low order schemas whose fitness values are higher than the mean will increase in numbers of samples from generation to generation.

The defining length of a schema is the difference between its outermost fixed position and its innermost fixed position. Since crossover is disruptive to schemata (the crossover point may fall within the schema's defining length and cause the schema to be lost) the best schemata (those that will survive and increase in numbers within a population) are those with short defining lengths.

Schemata with high fitness values and short defining lengths are called *building blocks*. Given a schema with a high fitness value and a short defining length, the optimal solution to the problem will likely be the winner of that schema's individual competition. "The notion that strings with high fitness values can be located by sampling building blocks with high fitness values and combining the building blocks effectively is called the *building-block hypothesis*." [2]

2.5 Uses of Genetic Algorithms

When we attempt to apply genetic algorithms to a number of practical problems, we find that there are some inherent difficulties. These difficulties include:

1. A suitable representation of the solution is not always easy to find. The solution domain may not be easily mapped to binary strings or real-valued vectors therefore making the encoding mechanism difficult.
2. The various constraints of a problem need to be taken into account.
3. Expert knowledge needs to be incorporated into the representation to help the search, but without allowing for biased opinions of experts to interfere.
4. The fitness function needs to be developed, often with the help of experts in the domain. Once again, their possibly biased opinions need to be guarded against.
5. The parameters of the genetic algorithm need to be tuned and compared to the results given by experts or other algorithms.

Despite these limitations and obstacles, genetic algorithms have been successfully applied to a wide range of fields. Some examples are timetabling (of exams or classes), job scheduling (of maintenance jobs or others), optimization problems, engineering, natural science, economics and business. [3]

3. Truckin'

Truckin' is the simulation model that was used as a basis for experimentation. It models a country in which producers produce commodities, retailers sell them and consumers consume them. Trucks distribute the commodities for profit while consuming gas from gas stations. The remainder of this chapter will detail each aspect of the Truckin' project and how they all fit together.

3.1 The Components

Involved in the simulation are a number of entities that interact with each other to mimic the economic world that we live in. For simplicity's sake a number of economic facts of our world have been overlooked, but enough has been incorporated into the model so that the simulation is not trivial. The simulation is based on a country in which commodities are produced, distributed and consumed. The individual entities are outlined below.

3.1.1 The Country

The country is made up of a square grid of highways. Avenues run north - south and streets run east - west. In the current version of the simulation there are 10 avenues and 10 streets. At any given intersection there can either be a producer, a retailer, a consumer, a gas station or nothing at all. The trucks in the simulation move from intersection to intersection trying to make profitable deals.

3.1.2 Commodities

There are three types of commodities in the current version of the simulation: crates containing a fixed number of items, the individual items as commodities themselves and gas. A truck can buy or sell crates, buy or sell items or buy gas. A producer produces crates at a given rate and sells them to trucks. Producers do not buy anything. A retailer can buy the crates from the trucks and break them down into items and resell the items to the trucks - each crate contains the same number of items. A consumer buys items from trucks, but does not sell anything. And gas stations sell gas to the trucks but do not buy anything.

3.1.3 Gas Stations

Gas stations can sell gas to trucks in unlimited quantities (i.e. they do not run out of gas). They do not buy any other type of commodity. The purpose of the gas stations in the simulation is simply to make it more difficult for trucks to survive. A truck must have a strategy to avoid running out of gas, and when a truck runs out of gas it no longer competes in the simulation. Therefore the trucks with the better gas-getting strategies are the ones that will survive the longest to make the largest amount of profit and are therefore likely to be selected to reproduce for the next generation.

3.1.4 Producers

Producers produce crates of commodities at a fixed and predefined rate. This rate can be used to control the amount of commodities present in the country at any given time. By increasing the rate of production of crates, we can make it a little easier for the trucks to make a profit because there will be a more plentiful supply of commodities to trade. And by reducing the rate of production, we can make the simulation a little more challenging for the trucks because there will be more competition for the commodities. The producers sell the crates of commodities at a predefined constant price.

3.1.5 Retailers

Retailers buy crates of commodities from trucks and sell them as items back to the trucks, in other words only retailers have to ability to break up the crates into individual items that consumers consume. Each retailer has a limited amount of storage space for the commodities that they buy and once they have reached that maximum, they cannot buy any more crates until they sell some items to decrease their stock.

A retailer has the ability to change its buying price for crates and its selling price for items. If the retailer has not sold any crates or items for some predefined period of time, the retailer will raise its buying price and lower its selling price, thus making the retailer more attractive to trucks. If the retailer has

performed several transactions within a short period of time, the retailer will lower its buying price and raise its selling price since it can afford to do so.

3.1.6 Consumers

Consumers buy items from trucks to consume. They have a fixed buying price for items and can consume at an unlimited rate. The rationale for this is that since the producers can only produce at a certain rate there will never be a oversupply of commodities for consumption (i.e. we only need to limit the flow at one end).

3.1.7 Trucks

The goal of trucks is to traverse the country buying and selling commodities at prices that are profitable. All trucks initially start off with a full tank of gas, an initial amount of capital and no knowledge of the layout of the country. As they move around the country they can obtain knowledge about the whereabouts and prices of dealers therefore enabling them to intelligently attempt to perform deals that will be profitable.

All trucks have a capacity (the maximum amount of cargo they can carry around). They can buy crates from producers, the amount being limited by either their available capacity or their available capital, and sell these crates to retailers, attempting to make a profit. A good strategy for a truck will ensure that the deal decided upon is indeed profitable at the time that the truck decided to

pursue this deal. Within the execution time of the deal, several variables of the deal can change (such as the buying price of the retailer) which can make the deal no longer a profitable one. Trucks can also buy items from retailers and sell them to consumers, again attempting to make a profit. Moving from intersection to intersection has a cost of expending gas, and has the benefits of obtaining knowledge about the dealers at each intersection and facilitating the execution of a deal.

3.2 The Interaction

3.2.1 At Start Up

At the beginning of the simulation, the country is set up as follows:

1. There are 5 producers scattered at different intersections within the country. The producers have no crates to sell until the first cycle of time intervals are up when they have produced their first 'batch' of crates. By varying the length of the cycle of time intervals in which producers produce a batch of crates, we can control the rate of commodities within the country. Within every cycle of time intervals each producer produces one more batch of crates to sell.
2. There are 10 consumers scattered at difference intersections within the country. They have the ability to consume items immediately.

3. There are 25 retailers scattered at different intersections within the country. They have half of their stock space full with items that are ready to be sold.
4. There is a variable number of trucks that can either all have the same starting position or can each have randomly selected starting positions within the country (tests have been run on both scenarios). By varying the number of trucks in the country, we can control the amount of competition that the trucks are faced with. Each truck initially has a full tank of gas, an initial amount of capital and no knowledge of the setup of the country. The goal of the truck is to make a profit by traversing the country and obtaining information about dealers and their prices so that they can make intelligent decisions on what a profitable deal would be.

3.2.2 Simulation Cycles

The first generation of trucks competes with each other for a predefined amount of simulation time at which point they are evaluated according to the fitness function. The fitness function in the simulation is the amount of capital that they have at the end of one simulation cycle. The fittest of the trucks are then chosen to reproduce and contribute to the next generation of trucks to play.

The next generation of trucks is made up of: the winners of the last simulation, the offspring of the winners of the last simulation and a new sample

of trucks that are created randomly. The current version of the simulation selects the fittest 25% of the original population to be the winners of the last simulation. Each pair of these trucks is then recombined to make up four offspring, therefore contributing another 50% of the next population. The final 25% of the next population is randomly created for the gene pool simulating a version of mutation. These percentages can be varied to study the effects of the selection, crossover and mutation in the context of genetic algorithms. The next generation of trucks then competes for one simulation cycle and the program then cycles again through the selection and recombination process.

3.2.3 How Genetics Fits In

All dealers (producers, retailers, consumers and gas stations) are static entities. In other words they do not change positions or strategies from generation to generation. But the trucks are not static entities, they can and do change positions and strategies and this is where we have found some signs of evolution. Each truck employs eight different strategies. For example a truck can adopt one of three different strategies on when and how to look for gas (see Chapter 4 Design for more details on the trucks' strategies). By making trucks with different strategies compete with one another, and then selected the best of those to reproduce and create offspring, we can eventually find the best combination of strategies to make a truck that will be the most profitable.

4. Design

The framework of the Truckin' project consists of the classes depicted in Figure 1.

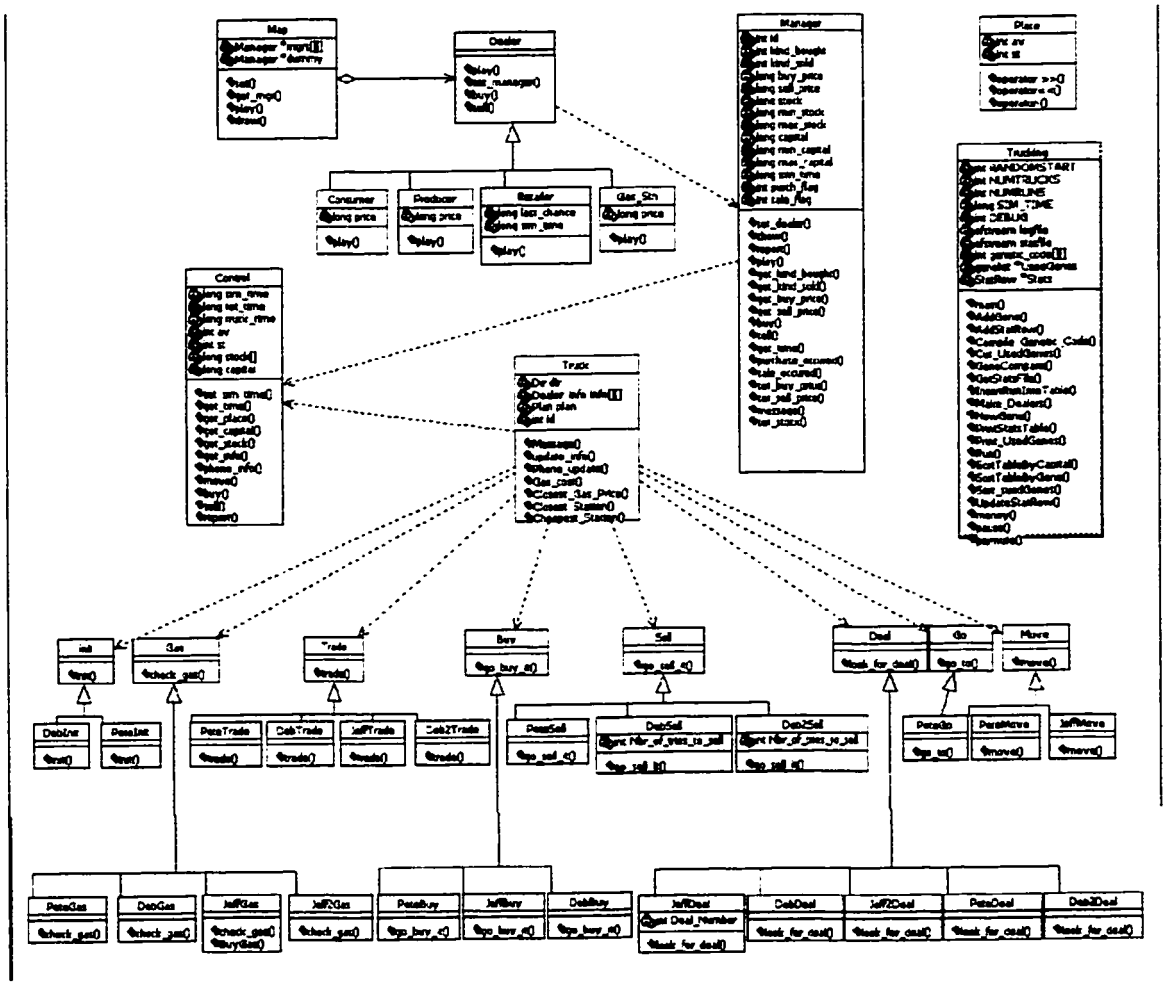

Figure 1: Truckin' Class Model

The Dealer class has four subclasses, one for each type of dealer in the simulation, namely Gas Station, Producer, Retailer and Consumer. To ensure that all dealers adhere to the rules of the simulation, no dealer is allowed to

access the data structures of the simulation directly. Each dealer has an associated Manager to which all requests are put forth. The manager can then in turn execute the request on behalf of the dealer.

The Place class identifies one location in the map of the country. The Map class consists of a map of the entire country and a pointer to each manager that lies within the country.

The Truck class is the base class for all trucks. Each truck is made up of eight different strategies. The different strategies are: Gas (when and how a truck should go about getting gas), Trade (how and when a truck should trade), Buy (how a truck buys), Sell (how a truck sells), Move (how a truck moves), Deal (what is a good deal to this truck and how it goes about finding one), Init (initialization of state parameters for the truck) and Go (how a truck goes from one place in the map to another - there is only one subclass of Go and therefore it does not add any diversity to the makeup of the different trucks). There are five subclasses for the Deal class, therefore any given truck can have one of five strategies on how to look for deals. There are four subclasses for the Gas and Trade classes, three subclasses for the Buy and Sell classes and two subclasses for the Move and Init classes. A single truck is created by selecting one subclass for each of these strategies.

Similar to the Manager - Dealer relationship, each truck has a Controller to which it puts forth all requests to be processed. The rationale for the Controller class is identical to that of the Manager class, which is to ensure that all trucks adhere to the rules of the simulation. At any given intersection of the map, the controller of the truck at that place interacts with the manager of the dealer at that location.

4.1 Encoding Mechanism

The encoding mechanism used in Truckin' is a discrete method but not binary. Each truck has a chromosome eight genes long. Each gene represents one of the eight strategies discussed in the previous section. The first gene specifies which Init class the truck has adopted and therefore has two possible values. The second and third genes specify which Gas and Trade classes the truck has adopted and therefore each have four possible values. The fourth and fifth genes specify which Buy and Sell classes the truck has adopted and therefore each have three possible values. The sixth gene specifies which Go class the truck has adopted and therefore has only one possible value - all trucks will have the same strategy for this method. The seventh gene specifies which Move class the truck has adopted and therefore has two possible values. The eighth, and last, gene specifies which Deal class the truck has adopted and therefore has five possible values.

By selecting one subclass for each of the possible strategies the simulation can examine up to 2*4*4*3*3*1*2*5 = 2880 different trucks. The goal of the simulation is to intelligently search the solution space and converge to a solution that approaches the optimal solution.

4.2 Fitness Function

The fitness function in the Truckin' simulation is directly related to the amount of capital a truck has at the end of a simulation cycle. At the end of a simulation cycle each truck will have a certain amount of gas left, a certain amount of cargo that it is carrying and a certain amount of capital that it has obtained. The trucks with the largest amounts of capital will be the ones selected to reproduce and compete in the next generation.

At the present time the amount of gas and cargo that the truck has at the end of a simulation cycle are not taken into consideration, but this may be an area for future work. Another possible implementation of the fitness function may be to give a monetary value to the gas and cargo and sum that with the capital to obtain a more accurate 'net worth' of the truck at the end of the simulation cycle.

4.3 Selection

Once each of the genes in a certain generation has been evaluated (i.e. their final amount of capital has been established), the selection process begins. The selection process in Truckin' selects the top one fourth of the trucks to

reproduce and compete in the next generation. Each pair of the surviving trucks create four children through crossover of their genes and the last fourth of the population is randomly selected from the gene pool (see Section 4.5 for more details). By varying the proportion of trucks that are selected to reproduce and compete in the next generation we can examine the effect of selection within genetic algorithms. By varying the proportion of trucks that are created from the survivors of the previous generation we can examine the effects of crossover within genetic algorithms. And by varying the proportion of trucks that are randomly created we can examine the effects of mutation within genetic algorithms.

With this method of selection the population size remains constant and the best of the trucks (i.e. the parents) compete in more than one generation cycle. This will enable the algorithm to show that the strongest genes are indeed persisting from generation to generation.

4.4 Crossover

The crossover mechanism in Truckin' randomly selects four genes from the first parent and then fills in the missing genes from the second parent. With this method there may be as little as one crossover point or as many as four crossover points. The rationale for using this method was to examine as much of the gene pool as possible in the least amount of generation cycles. The downfall to this method is that the convergence of good schemata is highly disrupted. As

outlined in Section 2.4, the higher the number of crossover points present, the higher the chance that a good schemata will be broken. Therefore, by randomly selecting the crossover points and the number of crossover points, we are very likely to break up a good schema if the algorithm does find one. Conversely, by randomly selecting the crossover points and the number of crossover point, we are likely to not spend too much time on any one schemata therefore raising the probability that we will in fact search the solution space in which the optimal schemata can be found and are more likely to find the optimal solution.

In the design selected crossover is implemented with a probability of one (i.e. it is always implemented). Possible variations to this design decision include a single crossover point that is (or is not) randomly selected and a probability factor taken into consideration.

4.5 Mutation

There is no mutation implemented in Truckin' at the present time that conforms to the classical definition of mutation. The classical definition of mutation says that for each truck in the current population alter one or more of its genes randomly with some small probability. This is used to avoid working with a finite, non-expanding gene pool, which may not contain the schemata of the gene that leads to the optimal solution.

In the Truckin' simulation no such mechanism was implemented, but the problem of 'missing' a set of gene combinations was addressed by adding one fourth of the new population randomly in each generation and by having random crossover points. Therefore as the simulation progresses, more and more of the gene pool will have been given a chance to compete.

4.6 Generation Cycle

The generation cycle in Truckin' is defined by creating a population, allowing the trucks in the population to compete for a certain amount of time and selecting the fittest of the trucks to contribute to the next generation. Each generation of trucks compete for some predefined amount of time. This amount of time is a variable within the Truckin' simulation and experiments with the length of the cycle have been performed. It has shown that the longer the simulation cycle is, the less the fluctuation that will appear in the results of the tests. In other words the evolution of the trucks is more apparent when they are allowed to compete for longer periods of time.

4.7 Termination Condition

The termination condition within Truckin' is defined by the number of generations that must compete before the simulation terminates. This number of generations is a variable that can be easily altered. We expect that extending this variable would lead to more convergence in the evolution of the gene combination of the trucks.

5. Implementation

Truckin' is implemented in C++ and takes advantage of features such as classes, inheritance and dynamic (run time) binding. In using these aspects of Object Oriented programming, Truckin' is capable of constructing trucks with different strategies to compete against each other and ultimately selecting the stronger trucks to continue to reproduce and compete in the following generations.

5.1 Program Structure

On start up, Truckin' checks if any command line parameters where supplied. If arguments are provided, Truckin' expects there to be exactly five. If no command line arguments are supplied, Truckin' uses defaults. The five command line arguments that can be provided are as follows: 'RANDOMSTART', 'NUMTRUCKS', 'NUMRUNS', 'SIM_TIME' and 'DEBUG'. 'RANDOMSTART' tells the program whether or not to start all the trucks at random positions in the country or if they should all start at the same intersection. Test runs have been run with both cases and we have observed that the start position of a truck affects the amount of profit that it makes. 'NUMTRUCKS' tells the program how many trucks to create in each generation. Test runs have proven that the higher the number of trucks in the simulation, i.e. the more competition the trucks have, the less likely it is for any one truck to do well. 'NUMRUNS' tells the program how many generations of trucks to create

and loop through the creation – evaluation – selection – manipulation phases. 'SIM_TIME' determines the length of one run of the program and 'DEBUG' determines whether or not to output the debugging statements to the log file.

There are also a number of other variables that have not been incorporated as command line arguments but have been left as constants within the program. These constants include: 'RANDOMGENE', 'SINGLEGENE', 'STARTAV' and 'STARTST'. By setting the 'RANDOMGENE' constant to zero a single set of genes that have been proven to work well together will be used to create all trucks. This feature is useful in determining if the evolutionary aspect of the program is not behaving as expected or if there is a more fundamental problem in the code that would occur even without the natural selection feature in place. 'SINGLEGENE' determines which of the three sets of proven pairs of strategies to run when 'RANDOMGENE' is set to zero. If 'RANDOMGENE' is set to one, then the value of 'SINGLEGENE' is ignored and the natural selection process takes place within the program. 'STARTAV' and 'STARTST' are the values used as the starting avenue and starting street for all trucks if the 'RANDOMSTART' command line parameter is set to false.

Once these command line parameters and constants have been established, the program proceeds in opening the log file and creating the map of the country. It then selects the first set of genes to take part in the simulation

(see Section 5.2. for more details) and begins with the first competing generation.

5.2 Encoding Mechanism

For the first generation of trucks, assuming the 'RANDOMGENE' constant is set to true, to make up the genetic code of one truck, the program randomly selects a number for each of the genes within the allowed values defined in the constants called 'POSGENES' (possible genes). Iteratively the program fills in the data structure called genetic_code that is a two dimensional array of the genetic codes of each of the trucks in the current generation.

```
if (RANDOMGENE)
 {
  v = rand()%(POSGENE[g]);
  genetic_code[t][g] = v;
 }
```

where t loops through the trucks and g loops through the genes.

For all subsequent generations, the program populates the genetic_code data structure according to the selection rules described in Section 5.4.

Once the program has completed the population of the genetic_code data structure, it is this information that is used to dynamically create trucks that use different strategies. The encoding mechanism in Truckin' uses base and derived classes to implement the creation of trucks. Each truck is initially created as an instance of the class Truck as shown in the following line of code.

```
trucks[t] = new Truck(t, ctrls[t], genetic_code[t]);
```

The base class of the Truck contains pointers to each of the strategy classes (Gas, Trade, Buy, Sell, Move, Deal, Init and Go). The constructor of the Truck class accepts as a parameter an array of eight numbers (the genetic code for one truck). This array is the gene of the truck to be created. Reading the array the constructor knows which derived class of each of the strategy classes to instantiate for each of the pointers in the base truck class.

```
Truck::Truck (int id_num, Control *controller, int genes[8])
{

 switch(genes[0])
  {
  case 0:
    init = new Peteinit(this);
    break;
  case 1:
    init = new Debinit(this);
    break;
  }

 switch(genes[1])
 {
 case 0:
   gas = new Petegas(this);
   break;
 case 1:
   gas = new Jeffgas(this);
   break;
 case 2:
   gas = new Debgas(this);
   break;
 case 3:
   gas = new Jeff2gas(this);
   break;
 }

... and similarly for the remaining genes.
```

5.3 Fitness Function

The fitness function in Truckin' is defined by the amount of capital a truck is able to accumulate during a single run of the program. The list of trucks that compete in a given run are managed by the 'UsedGenes' data structure:

```
struct genelist
{
  int genes[8];          // gene combination
  long money_acquired;   // capital after run
  struct genelist *next;
};
struct genelist *UsedGenes = NULL;
```

Once a generation has finished competing, the trucks are then sorted in decreasing order of money acquired and the top quarter of the population is allowed to reproduce trucks that compete in the next generation and to compete between themselves. In the following code UsedGenes is already sorted on capital acquired during the last generation cycle.

```
ptr = UsedGenes;
for(int i=1;i<div(counter,4).quot;i++)
  ptr = ptr->next;
prev = ptr;
ptr=ptr->next;
prev->next =NULL;
```

5.4 Selection

Once the first generation of trucks have completed their run, the selection of the next generation of trucks to compete is performed by taking the 25% of the trucks that performed the best in the last run, as described in the previous

section, creating the next 50% as children of these trucks and randomly selecting the last 25% to ensure that the gene pool continues to expand.

In the code below ptr points to a list of the best 25% of genes that participated in the last generation cycle therefore add these genes to the genetic_code data structure to compete in the next generation.

```
while (ptr != NULL)
 {
  for(int i=0;i<8;i++)
     genetic_code[counter][i] = ptr->genes[i];
  counter++;
  ptr = ptr->next;
 }
```

Once again ptr points to the best genes from the previous generation, and we therefore use these genes to create twice as many children as parents, i.e. for every pair of parents make 4 children and add them to the genetic_code data structure to compete in the next generation.

```
while ((ptr != NULL) && (ptr->next != NULL))
 {
  ptrmate = ptr->next;
  for (int x=0;x<4;x++)
     {
       ...
     }
  ptr = ptrmate->next;
 }
```

Finally, we fill the rest of the genetic_code data structure with random genes. This ensures that the program does not converging to a solution space that does not contain the optimal solution.

```
 for (int x=0;x<nbr_new;x++)
  {
   for(int g=0;g<8;g++)
      {
       if (RANDOMGENE)
        {
         v = rand()%(POSGENE[g]);
         genetic_code[counter+x][g] = v;
        }
...
}
```

5.5 Crossover

The crossover method used in Truckin' is to take a random selection of four genes from one parent and to fill in the rest from the other parent. This method is easily implemented and examines a large portion of the solution space but is highly disruptive to the schemata due to the many possible crossover points. In the worst case scenario there can be seven crossover points (each gene is taken alternately from each parent) and therefore the longest schemata is of length one. In the following code ptr points to the first parent and ptrmate points to the second parent.

First randomly select the first four genes from ptr->genes.

```
for (int y=0;y<4;y++)
        {
          v = rand()%8;
          genetic_code[counter][v] = ptr->genes[v];
          gene_filled[v] = 1;
        }
```

Then for every gene that is not already filled, fill it from ptrmate->genes.

```
for (int y=0;y<8;y++)
  {
    if (gene_filled[y] == 0)
        genetic_code[counter][y] = ptrmate->genes[y];
  }
```

5.6 Generation Cycle

For each generation cycle in Truckin' the simulation makes the dealers and managers to be involved in the cycle, makes the trucks and the controllers to compete in the cycle, allows the simulation to run for a predefined amount of time and then processes the results. In processing the results the simulation maintains the results of the cycle and then prepares the genetic_code data structure for the next generation cycle.

The dealers to be involved in the simulation are defined in a data file that is read in by Truckin'. According to the information found in the data file the simulation creates all the producers, retailers, consumers and gas stations at

particular locations on the map. One line of data from the data file is read in as follows:

```
data >> av >> st >> kind_bought >> kind_sold >> bp >> sp;
```

The same type of dynamic binding used to create trucks is used to create dealers as well. kind_bought and kind_sold define the type of dealer to be created.

```
...
else if (kind_sold == GAS)
   {
    dlr = new Gas_stn(sp);
    mgr = new Manager(d, NONE, kind_sold, -1, sp, DEF_MIN,
DEF_MAX, DEF_MAX, DEF_MIN, DEF_MAX);
   }
else if (kind_sold == CRATE)
   {
    dlr = new Producer(sp*UNITS_PER_CRATE);
    mgr = new Manager(d, NONE, kind_sold, -1, sp *
UNITS_PER_CRATE, 0, DEF_MAX, DEF_MAX, 0, DEF_MAX);
   }
...
```

Using the technique described in Section 5.2 the trucks and the controllers are created and then the simulation is allowed to run for 'total_time' amount of time.

```
while (time < total_time)
 {
  time += TIME_SLOT;
  map->play(time);
  for (tk = 0; tk < num_trucks; tk++)
     {
      ctrls[tk]->set_sim_time(time);
      trucks[tk]->play();
     }
 }
```

The map->play(time) method allows the managers to execute one simulation step. If the manager is associated to a producer then the producer would increase their stock by creating more crates. If the manager is associated to a retailer then the retailer would update their buying and selling prices. If the manager is associated to either a consumer or a gas station the play method has no effect. The trucks[tk]->play() method implements the playing strategy of the truck according to its genetic code.

Once the simulation time runs out Truckin' then collects the statistics for the run in the Stats data structure. The data structure is as follows:

```
struct StatRow
{
  int genes[8];            // gene combination
  long capital[MAX_RUNS];   // capital per run for this gene combination
  int instances[MAX_RUNS];  // the number of trucks with this gene
combination per run
  long TotCapital;         // capital truck obtained throughout all runs
  struct StatRow *next;
};
struct StatRow *Stats = NULL;
```

The amount of capital the truck collects per run and the number of times the truck is selected to compete in the simulation (instances) are tracked since they are indications of the fitness of the truck.

Once the statistics for the run are collected the genetic_code data structure is filled with the next generation of trucks and the simulation starts another generation cycle.

5.7 Termination Condition

The termination condition in Truckin' is simply a predefined number of generations. The NUMRUNS command line parameter defines the number of generations and the program terminates afterwards.

```
for (int nbr_runs=0;nbr_runs<NUMRUNS;nbr_runs++)
 {
      ...
 }
cout << "finished." << endl;
```

6. Results

In analyzing the results we have found that they are somewhat uneven. One set of runs may show signs of evolution, and then a subsequent run with the same parameters will show none. We find in the results that a certain gene combination can do very well in one run, and we therefore expect that it does equally as well or better in the next, but it doesn't. But what we do see quite clearly is that the different parameters in the simulation do in fact affect the results of the runs. The number of trucks in a generation, the number of generations, and the length of time simulated in one generation all affect the results produced by the simulation.

In analyzing the results, before we can look for signs of evolution, we must first make sure that there exists a gene combination that can make a profit. Once we have established that such a gene combination exists then we can look to see whether the simulations are able to find such a combination.

Therefore we investigate whether the trucks themselves make any money. If none of the trucks are making money, then we have to review the environment we created to ensure that there is a possibility for profit. By running the program with proven sets of gene combinations we can see that there are combinations that will make a profit. In the following example the simulation was run with ten trucks, for ten generations, each generation lasting 5000 time units. We can see

from Table 5 below that the gene combination 0 2 2 2 1 0 0 2 is a proven combination that can make a profit. Since we now know that there are combinations of genes that do make a profit, we now would like to see the simulation converge to these combinations, or possible others, when started from a random gene pool.

Gene Combination	Capital Acquired
02221002	$9283.50
02221002	$8402.65
02221002	$8393.45
02221002	$8189.25
02221002	$7767.85
02221002	$7515.55
02221002	$7244.75
02221002	$7160.95

Table 5: Capital Acquired by Proven Gene Combination

Now we will examine the affects of the different parameters to the results of the simulation. The parameters that may affect the results are:

1. The starting position of the trucks
2. The number of trucks in the simulation
3. The number of generations in the simulation
4. The length of time for one generation

We have observed that the starting position of the trucks does in fact affect the results of the simulation. Trucks that start off in the middle of the

country (intersection (5,5)) are more likely to succeed than trucks that start off at the North-West corner of the country (intersection (0,0)), the South-East corner of the country (intersection (9,9)) or at random starting points. The following figure illustrates this point. Each line in the graph represents a run of the simulation in which all of the trucks started at the specified intersection. For each line graphed, the results were produced by running the simulation with 50 trucks, for 50 generations, each generation lasting 10 000 time units. The graph outlines the results of the last generation of trucks, sorted in ascending order of capital acquired.

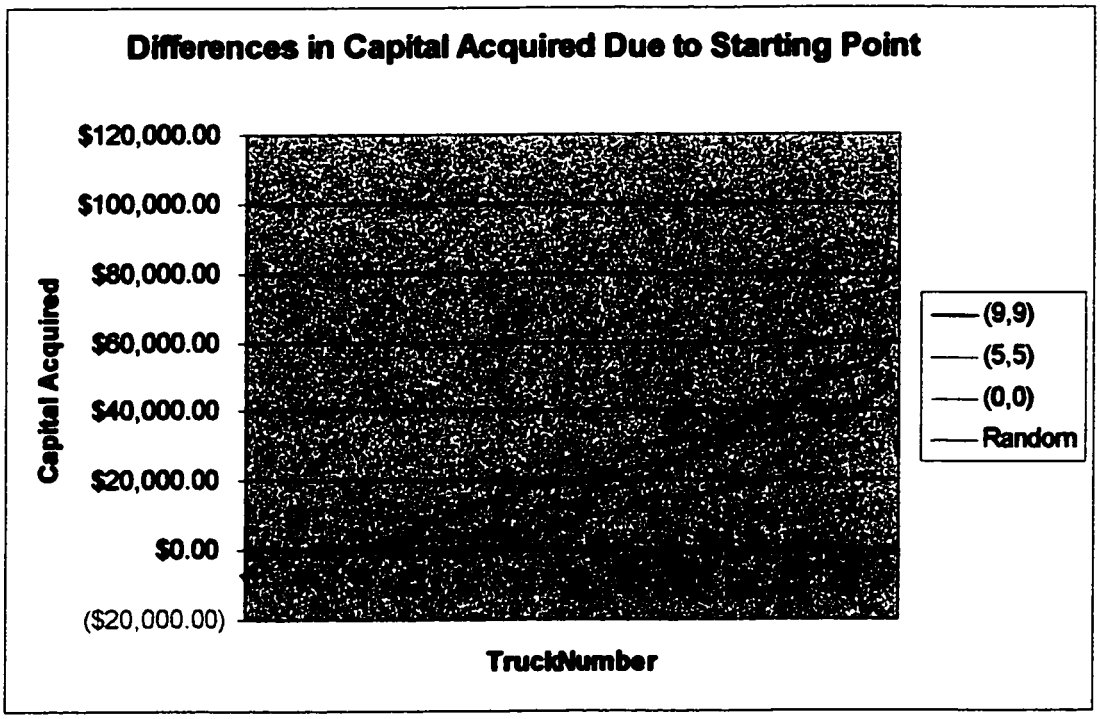

Figure 2: Differences in Capital Acquired Due to Starting Point

As we can see from Figure 2, the run of the simulation that started all trucks off in the middle of the country is the run in which more trucks made more money. For example, the best truck starting at (5,5) accumulated more that $100 000. At the beginning of the simulation, trucks explore the country by traveling East and South, collecting information about the dealers before attempting to make any deals. Consequently, a truck that starts at (0,0), the North-West corner, will explore the entire country before starting to deal, whereas a truck that starts at (9,9), the South-East corner, will not do any exploring. We believe that this is why trucks starting in the middle of the country are the most profitable, they don't spend too much, or too little, time exploring before attempting any deals. The trucks that start of at intersection (0,0) probably spend too much time collecting information (they roam the entire country) before attempting any deals, and the ones that start off at intersection (9,9) probably don't spend enough time (they immediately attempt to make deals with no knowledge of the country).

We have observed that the number of trucks in the simulation also affects the results greatly. The more trucks in the simulation, the more likely it is that the best truck will make a large amount of money, and that more trucks in general will make money. The reason that we suspect for this is that there is a greater sampling of the gene pool and therefore more of the 'winning' combinations are likely to have been explored. Conversely, if there are too many trucks in the simulation, then the competition for the limited amount of commodities (the

producers production of crates is unlimited in that they don't have to sell the existing crates before producing more, but yet the *rate* of production is fixed therefore there is a maximum amount of crates that they will produce) is felt by the trucks and they are less likely to make a huge amount of profit. The following figure illustrates this point. For each line graphed, the results were produced by running the simulation with the specified number of trucks, for 50 generations, each generation lasting 10 000 time units. The graph outlines the results of the last generation of trucks for each run of the simulation sorted in ascending order of capital acquired.

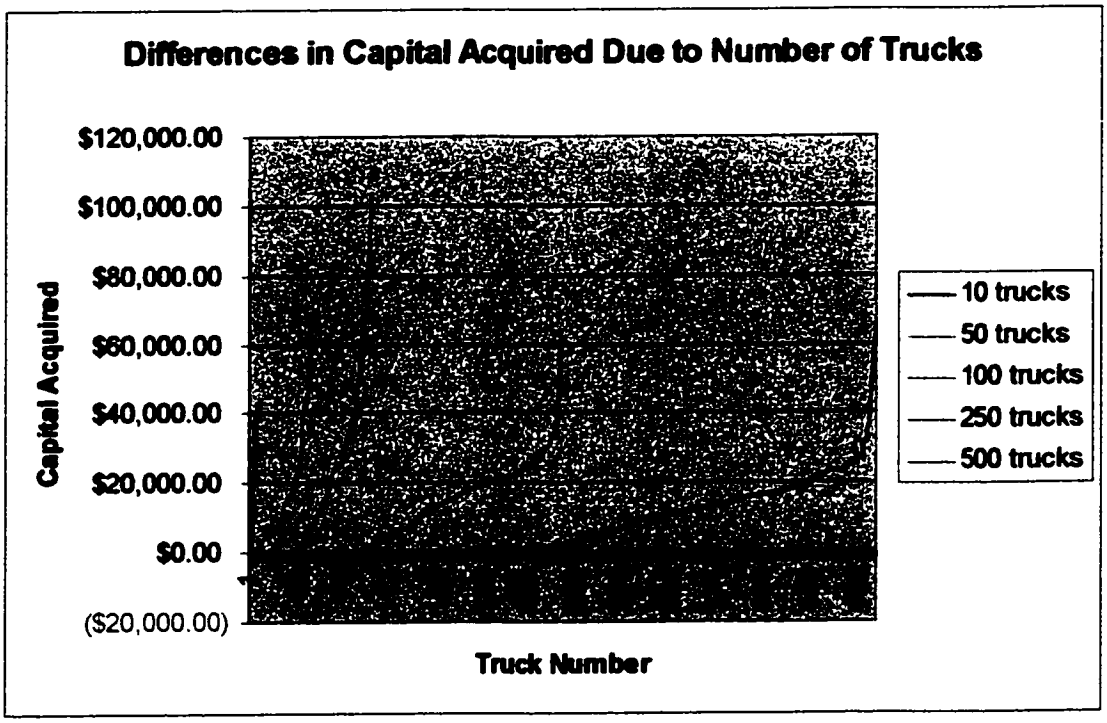

Figure 3: Differences in Capital Acquired Due to Number of Trucks

As we can see from Figure 3 above, the most successful run was the one with 100 trucks. In this run the most profitable truck in the last generation made more that $100 000. There where enough trucks in the simulation to sample a large part of the gene pool, but few enough to give each of them a chance to do well.

The number of generations in the simulation does not seem to affect the amount of money that the trucks can make, but does seem to affect the amount of convergence visible in the results. The more generations in the simulation the more of the trucks in each generation converge to a certain gene for some of the strategies. The results in Table 6 were obtained from running the simulation with 100 trucks, for the specified number of generations, each generation lasting 10 000 time units. The top 30 money making gene combinations of the last generation of the runs are extracted and shown in Table 6.

5 Generations	10 Generations	50 Generations	100 Generations
12111013	01201010	01321010	01000010
12110013	01201013	13211010	01300010
12110013	01200014	01321010	01000010
01220012	02112013	00321010	01302010
01220012	03310013	01321010	01302010
01220012	01312010	01321010	01302010
01220012	02112013	01321010	00300013
01220012	03100014	02121010	11000013
03022014	00111010	01321010	01302010
01220012	02022010	01321014	01302010
11111003	02020010	01321010	02302010
12122011	03010013	11201010	00302010
13001000	02310010	01321010	00302010
11111003	00020014	01321010	01302013
13112004	01022010	01321010	01302010
02321011	02020010	00322010	01302010
11102004	02002010	01321010	01302010
13100014	02022010	01321010	01302010
12101013	03100003	01321010	01302010
00111000	02201014	01321010	01000013
13102004	01020010	00321010	01302010
11201004	01201014	11201010	01300014
11222014	02201003	01321010	01302010
13110013	00200004	01321010	01322014
02110000	02300010	00321014	01300010
11122014	01111000	00321010	01302010
11211004	00000014	01321010	01302010
11201004	12220002	01321010	01302010
11202014	01101004	00321014	01302010

Table 6: Difference in Convergence Due to Number of Generations

We can see from Table 6 in the first column the gene combinations have much less of a pattern visible than in the last column. In the first column the gene combinations are rather random, but in the last column there is a high proportion of gene combinations of form 0130201X. Therefore the more generations the simulation is allowed to run, the more likely it is to converge to a pattern in the gene combinations.

The length of time for one generation also seems to affect the results of the simulation. The amount of capital acquired by the trucks is greatly increased if the trucks are allowed to compete for longer time periods in each generation, but the amount of convergence exhibited is not affected by the length of time for one generation. Figure 4 and Table 7 below illustrate this point.

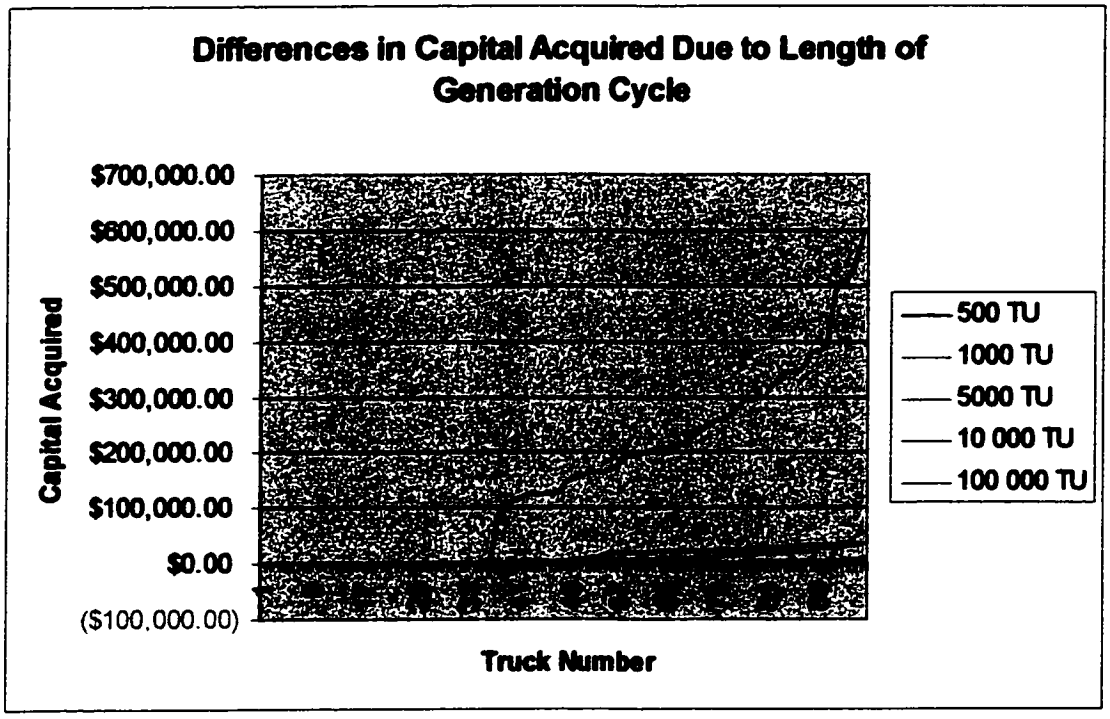

Figure 4: Differences in Capital Acquired Due to Length of Generation Cycle

500 TU	1000 TU	5000 TU	10 000 TU	100 000 TU
03300010	00301010	01210010	01201014	02021010
02312013	01311010	01110013	01321010	01020010
01311014	01300010	01312010	00211010	02021010
01311013	03100010	01312010	00320010	02021010
03100013	00202010	02012013	00320010	02321010
01301014	00300010	02011013	12322010	02021010
01300013	02200010	02001013	00321010	01221014
01310014	01301010	00100013	01320010	01221014
03111013	02100010	00012000	00321010	01221014
01102013	02200010	00311013	01320010	02321014
02012013	01112010	13301014	01212010	01320010
02002013	03202010	13301013	11112004	01021010
01302010	00101010	00000003	01320010	01321010
01111010	02102010	11200013	00320010	01021010
00012014	02202010	11101013	01301010	01021010
02312013	03300010	01000010	00320010	02021010
02312013	01100010	01012010	02201010	02021014
00001010	01202010	00012013	11301010	02321010
01301010	03300010	00312010	00320014	02321014
01302013	03200010	13311013	00320010	02021010
01301010	02100010	12110013	00320010	02321010
01301010	01000013	13001010	12222013	01021014
02102014	03200010	11211013	01320010	01021010
01311010	02202010	13011013	00320010	01021010
01301010	00102010	13001014	01320010	02022010
01111014	03220012	12311013	00321010	02021010
01300014	02300010	02212014	11320010	01022010
02302014	03102010	11200013	00320010	01021010
02310014	03321010	00311013	13210014	02022010

Table 7: Difference in Convergence Due to Length of Generation Cycle

The results in the Figure 4 and Table 7 above were obtained from running the simulation with 100 trucks, for 10 generations, each generation lasting the specified time units. All 100 trucks of each of the last generations are graphed in Figure 4 in ascending order of money acquired. The top 30 money making gene combinations of the last generation of the runs are extracted in Table 7. We can clearly see that the run with 100 000 time units for each generation made the most money (the best truck made $600 000), but it does not display any more definite signs of convergence then the other four runs.

Lastly we will show that the simulation does display some signs of convergence. The following table displays the results of the first, the 25th, the 50th and the 100th generations with the simulation having been run with 100 trucks for 100 generations each generation lasting 100 000 time units. The top 30 trucks of each generation are extracted. We can see that the first gene tends to converge to 0 by the last run, the second also to 0 with some sporadic 2s, the third to 3, the fourth to 2, the fifth to 0 or 1, the sixth to 0 (it is the only option for this gene), the seventh to 1 and the last one to 0 or 4. Conversely the first run shows no such patterns.

1st run	25th run	50th run	75th run	100th run
00200013	00321014	00021010	00321010	02320010
02322013	00321010	00321010	11201010	02320010
12111013	00321010	00321010	13012010	00320010
12310014	00321010	00321010	00321010	00320010
02220002	00321010	00321014	00321010	00321010
12321010	00321010	00321010	00321010	00321010
13101003	00321010	00321010	00321010	00320010
10100013	00321010	00321014	00320010	00021010
12222011	00321014	00321010	00321010	00321010
11012004	00321010	00321010	00320010	12300014
12301003	00321010	00321010	00321010	00022010
02221012	00321010	00021010	00321010	00320014
03221014	02221014	00321010	00320010	00320010
01101010	00321010	00321010	00321010	00320010
02322012	00321010	00321014	00321010	00320014
12321013	01321010	00321010	00320010	00320010
03310010	01321010	00321010	00321010	00120014
01121002	01321010	00321010	00321010	00320014
01121002	00321010	00021010	00321010	02320014
00020012	00321014	00321010	00320010	00321010
00010011	00321010	00321010	00321010	00120014
02010011	00321010	00321010	00320010	02320010
03101002	00321010	00321010	00321010	02320014
03211001	00321010	00321010	00321010	02320014
03210013	00321014	00321010	00320010	00120014
10320004	01321010	00321010	00321010	00320010
11222012	00321010	00321010	00321010	00122010
12100010	02221010	00021010	00321010	00320010
03321003	12202004	00321010	00321010	00320010

Table 8: Convergence

7. Conclusions

There are some surprises in the results produced by the simulation. One generation of the simulation can exhibit sure signs of evolution, but the next with the same parameters does not. In one generation a certain gene combination can do very well, but the same combination goes bankrupt in the next. These inconsistencies lead us to believe that there are several defects in the current version of the simulation.

In the first version of the simulation there was only one type of dealer and only one commodity. Trucks had unlimited capacity and dealers could produce goods at an unlimited rate of production. This version of the simulation produced no signs of evolution at all. In each generation of the simulation there would be some real winners and some real losers, but there were no patterns in the gene combinations that would win or lose. Then in subsequent generations different combinations would win with no correlation to the previous generation.

The simulation was then expanded upon to present a more realistic version of the economic world that we live in. These changes produced somewhat more positive results, but still no consistent signs of evolution. We can therefore conclude that the economic world depicted by the simulation is still too simplistic to exhibit the results we are looking for. Implementing some of

what is mentioned in Chapter 8 may elaborate on the economic world of the simulation sufficiently so that this is no longer an issue.

There are 2880 different gene combinations in the current version of the simulation. The results of the simulation have been studied by another graduate student (Liang Yu) who concluded that the inconsistencies in the results of the simulation may be due to the fact that number of different strategies and actual variation between the strategies is insufficient to produce consistent convergence [7]. In other words, in some generations the simulation does find real apparent 'losers' among the trucks and real 'winners', and in other generations it does not. Any future work done with the simulation should most definitely include expanding not only the number of trucks in the simulation but also the actual strategies implemented must differ more than those implemented in the current version. In doing this, we may find that the simulation will in fact exclude the trucks whose strategies are 'weak' and converge more consistently to the trucks whose strategies are 'strong'.

Since we do in fact see some convergence in the results of the simulation (sporadically) we are lead to believe that there is in fact a chance for the simulation to produce the results that we are looking for, but to do so some or all of the Future Work in Chapter 8 will need to be implemented.

8. Future Work

There are several areas of the Truckin' project that could be expanded upon in the future. Some involve expanding the features and functions that are already in place in the current version and others entail expanding the economical world that Truckin' depicts.

As described in Chapter 7, the number of variations on the strategies and the actual differences in the strategies were insufficient to produce enough genetic variation in the trucks to draw any positive conclusions. In the current version of Truckin' there are 2*4*4*3*3*1*2*5 = 2880 different trucks. Although this number may seem large, Liang Yu found that to produce any real convergence towards an optimal solution more permutations of trucks are necessary [7]. Possibly the most important future work that can be added to the Truckin' simulation would be to increase the number of different strategies for each gene in the encoding mechanism and to add greater variation to the actual strategies implemented. Once these improvements are put in place, we can assess whether the Truckin' simulation would ever exhibit signs of evolution. Once some signs of evolution are established we can go on and improve the features in the simulation to show stronger signs of evolution.

Mutation is not implemented at all in the current version of Truckin'. Future work may involve adding mutation to the simulation (and nothing else)

and noting the differences in the results between the pre and post mutation versions of the simulation therefore extracting the effects of mutation alone.

The current fitness function is rather crude. A truck competes in the simulation for a predefined amount of time and is assessed by the amount of capital it has acquired at the end of the run. Other criteria could be implemented, such as number of profitable trades accomplished, largest amount of capital acquired at any point in the run, earnings per unit time of the run, or earnings per litre of gas consumed. With any of these criteria in place as the fitness function we may find that the program converges to a different set of genes and therefore can conclude which genes are best to accomplish the largest amount of trades, which genes are best for the largest earnings per unit time of the run, or which genes are best to accomplish any of the fitness functions implemented.

Changes to the actual economic world that the trucks compete in can include Bankruptcy and Franchising. Bankruptcy is in place for trucks, in that once a truck is out of money and out of gas they can no longer perform any tasks and therefore dies out of the simulation, but no such hurdle is in place for retailers. Retailers pay for the crates that they buy and acquire money for the items that they sell, but no budget is kept for them. Future work can include keeping track of money acquired and spent by retailer and making a retailer go bankrupt once he is out of money. On the other hand, if a retailer does very well,

then that retailer should be allowed to franchise and occupy two intersections on the map but continue to share one 'bank account'.

In the current simulation, trucks move around the map with no knowledge of other trucks and their locations. Another area of improvement for the simulation to mimic our real economic world would be the introduction of traffic. If there are a certain number of trucks at a certain intersection, then no new trucks are allowed to move into that intersection until someone has moved out. This would cause the truck to waste simulation time and gas without being able to accomplish anything. Some differences in the move strategy could also be implemented to prevent a truck from ever encountering heavy traffic.

All of these changes mentioned above can be implemented independently of all other changes and therefore should be added to the simulation one at a time. Once one change is in place data should be analyzed to establish the impact this one change had on the entire project.

References

[1] Melanie Mitchell, *An Introduction to Genetic Algorithms*, The MIT Press, 1996.

[2] M. Srinivas and Lalit M. Patanaik, *Genetic Algorithms: A Survey*, IEEE, 1994.

[3] Thomas Back, Ulrich Hammel and Hans-Paul Schwefel, *Evolutionary Computation: Comments on the History and Current State,* IEEE, 1997.

[4] Robin Biesbroek, GA Tutorial Home Page, http://www.estec.esa.nl/outreach/gatutor/Default.htm

[5] Jose L. Ribeiro Filho ang Philip C. Treleaven, *Genetic - Algorithm Programming Environments*, IEEE, 1994.

[6] John Holland, Adaptation in Natural and Artificial Systems, Ann Arbot, MI: Universoty of Michigam Press, (1975).

[7] Liang Yu, *Truckin' Simulation and Visual Interface,* Masters of Computer Science, Concordia University, March 2001.