A Comparison of Evolutionary Algorithm Operators

By Jonathan Brooks (job64@aber.ac.uk)

Abstract

Contents

Project Background

Aim of the Project

The project aim is to conduct an Empirical analysis to compare Evolutionary algorithm operators and how they perform to compared to each other.

Objectives

- Produce an Evolutionary algorithm with varying operators.
- Collect performance data for Evolutionary algorithm operators.
- Produce a performance comparison of Evolutionary algorithm operators.

Deliverables

- Collect/Compile Evolutionary algorithm operator data.
- Source code of an Evolutionary algorithm which demonstrates an increase in performance within the Simulation.
- This project report that discusses the final solution to the project including the Design and critical evaluation.

Evolutionary Algorithm Background–

An Evolutionary algorithm is a process of optimising a pre-existing solution over incremental changes improving performance to the solution. The common underlying idea among the various versions of Evolutionary algorithms is that given a population of individuals, operators are used to apply pressure to force evolution to these individuals over several generations [1]. This related to Survival of the fittest where the fittest induvial within a Population is the most dominant in performance or strength.

The operators used within an Evolutionary algorithm come under Selection, Crossover and Mutation operators. Each of them has varying functions which are different in terms of how they act and how they perform.

Selection operators or parent selection operators (PSO) are aimed at exploiting the best characteristics of individuals of a given population. This type of operator chooses the most suitable individuals to have their recombined. Selection operators are varied in terms of how much selection is applied to the population, whether individuals are taken from the local or global population and the criteria in which an induvial is chosen [2].

Crossover operators within an Evolutionary algorithm are ways in which the algorithm takes more than one individual solution and recombines individual's data based on a set of criteria to reproduce children for the next generation of individuals [3].

Mutation operators are used within Evolutionary algorithms to maintain genetic diversity from generation to generation. The basic principle of the operator is that there is that data within an individual would have a Random mutation which helps keeps genetic diversity within the population preventing individuals becoming too similar which helps avoid local minimum solutions [4].

Simulation Framework –

The project uses a pre-existing framework which provides the simulation space in which the Evolutionary algorithm is implemented. The framework is written in JavaScript and can only be run within an Internet Browser. The framework uses Box2D which is a 2D physics engine for games and is used as the base for the frameworks simulated environment, the framework itself handles all the interaction with Box2D.

The data for the Simulation such as average population performance and car fitness function is internally created within the framework and provides a means of access between an Evolutionary algorithm implementation.

Methodology –

The methodology used in this project is Scrum. The approach allowed the project to proceed in incremental steps or in sprints where one component of the Software is implemented and tested within a specified time-frame. The approach allowed for different parts of functionality to build upon the other.

Version Control –

The version control used during the project was GitHub where a private repository was used to store the Technical and Document work for the project. Using GitHub allowed for the project to be downloaded and changes to be applied from anywhere with an internet connection. Local storage was also used on multiple machines, but GitHub served as a back-up and updated after incremental changes have been applied to the project.

Relevant Technologies –

Node.js –

Node.js is an open source server environment which provides a development environment allowing for JavaScript code to be run. The environment can run a various platform such as Windows, Linux, Mac OS and Unix. Node.js uses asynchronous programming allowing for a sequence of events to run side by side [6].

NPM –

NPM is an open source Software Package Manager and allows access to a database of software packages and the package manager can be downloaded onto the client side. NPM allows version control of projects allowing for easier management and error correction during testing. The components consist of the website, Command Line interface and the registry which stores all the software packages [5].

Unit Testing Frameworks –

Unit testing is a method of using a framework allow for the testing of bottom level code components whose purpose is to validate functions or data based on a set of pre-defined criteria such tests cannot be testing from the front-end. There are many JavaScript unit test frameworks that can be used such as the following, Mocha, Jasmine and Node.js assert.

Data Collection –
Data collection within the project is conducted by producing the average performance of the Cars over several generations and then added to the Browsers local storage which would then be added to an Excel spreadsheet.

LocalStorage is a Google chrome web storage which works within the browser's own framework memory. There is an inherent restriction to adding data produced by the simulation to the Client of the computer because of browser security which is the main reason for LocalStorage being used [7]. Other means could be used such as sending the data to a server-side script which would then add data to the client, but LocalStorage provides a more simplistic approach which does not over complicate the process.

Internet Browsers –
Google chromes provides DevTools which provides an in-build IDE to debugging and testing browser run scripts. This IDE was using throughout the process of implementing and testing the different components of the Evolutionary algorithm. The IDE provides an in-depth way of tracking the varying variables which made the testing process simple looking for complications. Directory access is also provided for accessing the different scripts while also providing tabbing for scripts to be viewed on the same window [8].

Design, Implementation and Testing

Design

The design of the Evolutionary algorithm consisted of 4 Selection operators, 2 Crossover operators and 3 Mutation operators.

Selection Operators
Roulette-wheel Selection – Members are assigned a portion of the roulette wheel based on the members fitness where the highest-ranking member would get a larger portion than the lowest ranking member. The selection randomly goes though choosing a member. The function will sum the score of all the cars in a generation into one variable, and then a random float will be timed against the sum. The function will then iterate over the population of cars decrease the sum score until it is below 0, and whatever car the iterator is on when the sum score is below 0 is the chosen car to be returned.

Tournament Selection – The highest performing individual is chosen from a subset number of individuals which are randomly chosen. This function will use an ordered array where a set of cars are randomly chosen from the array and placed into a new smaller array and ordered. The new set of cars will then have the highest performing car extracted and returned from the function.

Uniform random Selection – Members of a population have an equal chance of being chosen as a parent. This function will use a random number generator to select an individual car from an array of cars and returned from the function.

Hybrid Tournament Selection – Two members are chosen from the population one is the weakest members the other the strongest. This function works the same as Tournament selection but also provides the option to get the weakest member from the smaller array instead of the strongest.

Crossover Operators

One-point crossover – 1 point is chosen at random from the parent's chromosomes and bits are swapped across the parent members from either side of the point to create new members.

Two-point crossover – 2 points are chosen at random in the members chromosomes and are data is swapped between the members.

Mutation Operators

Single Mutation – A single mutation is applied to an individual's data, there is a 1 in 5 chance of 1 of the data points changing which then use a Random floating-point number to increase or decrease the original datapoint by. Within this type of mutation there is a certain amount of uncertainty because of the randomness such as amount of change applied and whether the data will increase or decrease. The function that will handle the change will also deal with a maximum number of 1 and a minimum number of 0, the function will be checking that these are not passed and make the necessary corrections.

Multi-Mutation – An individual's data point chance of being changed depends on the individual's performance in the population such as the weaker individual having more mutations and the highest performing having the least mutations. This function will use the single mutation operator but multiple times, the function will also make sure the same mutation is not applied again to the same datapoint which more than the same data point will be changed. This means the function will record was data point of the car has been changed as to not apply a mutation on it again.

Cluster Mutation – A single mutation is applied to 1 of the 5 data points, the mutation is gotten by classifying the mutation into a clustering and using KNN to get its closest neighbours. And then increasing or decreasing the datapoint to get closer to the highest performing datapoint. The function will either increase or decrease the datapoint depending of the highest performing neighbouring car is higher or lower. A version of Particle Swarm optimisation is used to change the mutation to move closer to the ideal number.

Scoring Operator

Using Clustering to re-score a car by using KNN, a car that has its score changed uses the summed score of its surrounding neighbours within a cluster. This operator would be applied before the Evolutionary algorithm by adding the cars into the clusters and then applying the change to the cars original car using KNN. The clustering used has 5 main clusters for the cars data points and sub clusters for every position in the array. The clusters are ordered arrays of the floating point numbers.

Elitism

Elitism is a mechanism with an Evolutionary algorithm to apply more selection pressure to the population by allowing a set member/s of a population to be passed onto the next generation based on a set of criteria and without any alteration, the criteria I will be used is that the top performing car/s will be passed onto the next generation and they will still be used in the crossover process.

Development Choices

Programming Language

The framework that the project uses is built in JavaScript which is the main reason for JavaScript being used to implement the Evolutionary algorithm. Other alternative languages had been considered such as PHP, but JavaScript was the easier option to consider because data passed from the simulation were passed via JavaScript objects.

Development Environment

The project was compiled in the command console environment using the package manager NPM for Node.js, this allowed for automated dependency within the project and allowed for client-side compilation of the software.

Updating the project code in JavaScript was done using Notepad++ which is a free code editor which can be used in the Microsoft environment and supports several languages including JavaScript. Notepad++ was used because of it's easy use and simplicity. It also provides tabbed editing where multiple scripts can be seen and edited at the same time in a single window.

The framework the Evolutionary algorithm is implemented in was not able to be run client side on the computer because of the framework using frontend HTML/CSS to start and show the simulation. The framework was tested running in Google Chrome because of previous experience using the browsers development environment to debug software.

Implementation

The following are the various sprints that were conducted during the project.

Sprint 1, 3rd Feb – 7th Feb

The first sprint consisted of setting up a version control/file system which would be used to store the project, A private repository on GitHub was setup with a copy of the original JavaScript framework and the project outline. The project outline was also created which consisted of a brief explanation of the project and a general predicted timeline of the different sections of the project.

A review of the original Cars simulation framework was conducted to understand the code structure and how the Cars data was formatted. Further learning of the JavaScript was also conducted while reviewing the code which helped lessen the complications in later sprints.

Sprint 2, 7th Feb – 14th Feb

During the second sprint Roulette Wheel selection was implemented with a basic skeleton outline of an Evolutionary algorithm code. No issues were encountered with implementing this functionality, but it required further learning of how to format a cars data to be manipulated to check that what was returned from the function was the correct format as not to course future complications.

A basic version of a Mutation operator was implemented with a simple random change affecting a cars piece of data. The original Mutation operator of the framework was reviewed to see if there were any point that could be applied in the new Evolutionary algorithm, but I could not grasp the purpose of parts of the original code so a completely new one was implemented.

Nearing the end of this sprint a version of One-point crossover was implemented with a pre-defined crossover point. This implementation had complication with the combination of two cars data to create a new car, the problem was rooted in my lack of knowledge of how JavaScript implemented object creation and manipulation. Though this was overcome with further studying the language and adapting existing code from the internet as a reference.

During implementing the previously mentioned implementing I was also getting used to using Google Chrome Dev Tools for debugging the code and testing the data format.

Sprint 3, 14th Feb – 21st Feb
Tournament Selection was implemented in this sprint which used the global population of the generation not using Sub set arrays. While implementing this functionality I expanded it adding the option to get the weakest car or the strongest car which acts as a separate selection operator allowing for more experimentation.

Multi-mutations functionality was implemented during this sprint which expanded on the original Mutation operator but allowing for it to be run more than once on other data points of the same car. After the implementation I realised there needed to be a way for the operator to define if the car needed Multi-mutations, so the Cars parent's performance scores was used and passed to the operator.

I implemented the Clustering of the cars data points and the re-scoring using KNN. The first part of the functionality was the clustering back-end and setup of the clusters. Each data point type would have its own cluster with the cars Id. The second part of the functionality was to add the K-nearest neighbour algorithm which got the surrounding car data points within the cluster by a specified margin.

When testing this implementation, the specified number of numbers had to be decreased during the first couple of generations of the simulation because of the cluster size not being big enough, when the simulation reaches a certain generation the number of neighbours increase.

Sprint 4, 21st Feb – 28th Feb
Two-point crossover functionality was added to the Evolutionary algorithm which allowed for two points in the data to be combined to create new child, this expanded on the one-point crossover function by adding extra conditionals statements checking for the second pre-defined point of crossover not just one.

The performance of the cluster re-scoring when used within the Evolutionary algorithm was not performing as expected so I changed the scoring method to include the cars original score not just its neighbours which were not having as big enough of an impact compared to the using the cars original score with the neighbouring cars scores.

The crossover operators were originally using a fixed point of crossover when being run on the cars data, I applied a change where the points of crossover were set to be random, so they are no longer pre-defined.

The original car ids were being produced using a random String generator, but I found a duplication problem where when a car was being created its id had a chance of being a duplicate of one of the cars that have already been created. I switched the id to a number system for the number of cars that have already been created which prevents duplicated being created.

I added an implementation which allowed for the top ranked car during a generation to not be deleted from the local population allowing it to go through the crossover process a second time. This implementation allowed for the best cars data to be spread more to the next generation of cars.

Sprint 5, 28th Feb – 7th Mar
During this sprint I changed tournament selection to include sub-set arrays which include randomly selected member of the population. The tournament would then be applied to the sub-set population not the global population.

Elitism was implemented during this sprint allowing for cars to be passed on to the next generation without any alterations. This implementation only taken the top performing cars during a generation to be elites, the number of elites is pre-defined and is not altered during the simulation running.

Uniform random selection was implemented which gives all cars within a generation an equal chance of being chosen.

Unit tests were implemented using Jasmine framework for most of the Evolutionary algorithm operators.

Sprint 6, 7th Mar – 15th Mar
During this sprint data collection was implemented by Storing the Simulations average performance results in Google Chromes local Storage. The data collection required 10 simulations runs for one version of an Evolutionary algorithm so apart of the implementation refreshes the web page after the simulation reaches 151 generations. An excel spreadsheet was setup with template graphs to put the data into.

Testing

Operator testing
During the various sprints of implementing the operators there were various issues that occurred which did not cause a program crash.

One of the major problems that resulted in big set-ups was a mistake made in the Random int generator function which caused a repeat in data collection. The randomInt function was not checking the generated int against an array of numbers that are passed through as an array. This caused a repeat of testing of the Multi-Mutation operator and Crossover operators because they both use this part of the randomInt function.

Visual Testing
During the project one of the main types of testing was analysing the visual representation of the cars running in the simulation. This type of testing allowed me to check that the Evolutionary algorithm was performing as expected quickly without debugging in-depth. Through this type of testing I found a few problems such as duplicate cars being created due to ID duplication and crossover not working as expected.

Unit Testing
The unit testing done within the project used Jasmine to implement the unit tests. For the varying operators within the Evolutionary algorithm each had their own unit test to validate the return of the main functions. One of the biggest problems found with the unit testing is validating the outputted data of the operator as an operator used other functions to manipulate data.

Research Experimentation Results

Introduction

This chapter discusses the data collected from the Evolutionary algorithm using the varying operators. The data collected is for varying combinations of Evolutionary algorithm operators which below the results will be shown and discussed.

Elitism

The following Figures show an Evolutionary algorithm results using Elitism where the number of elites increase in the different figures, each of the figures use the same Evolutionary algorithm but with varying number of elites used.

The following will be an analysis of the changing Selection pressure by increasing the number of elites used in an Evolutionary algorithm and how this selection pressure change compares with using other operators.

The Graphs are a summarised average of 10 run simulations of the same Evolutionary algorithm to better account for uncertainty with the running Evolutionary algorithm. The standard deviation in the figures refers the differing of performance between the 10 run simulations during data collection.

Figure 1

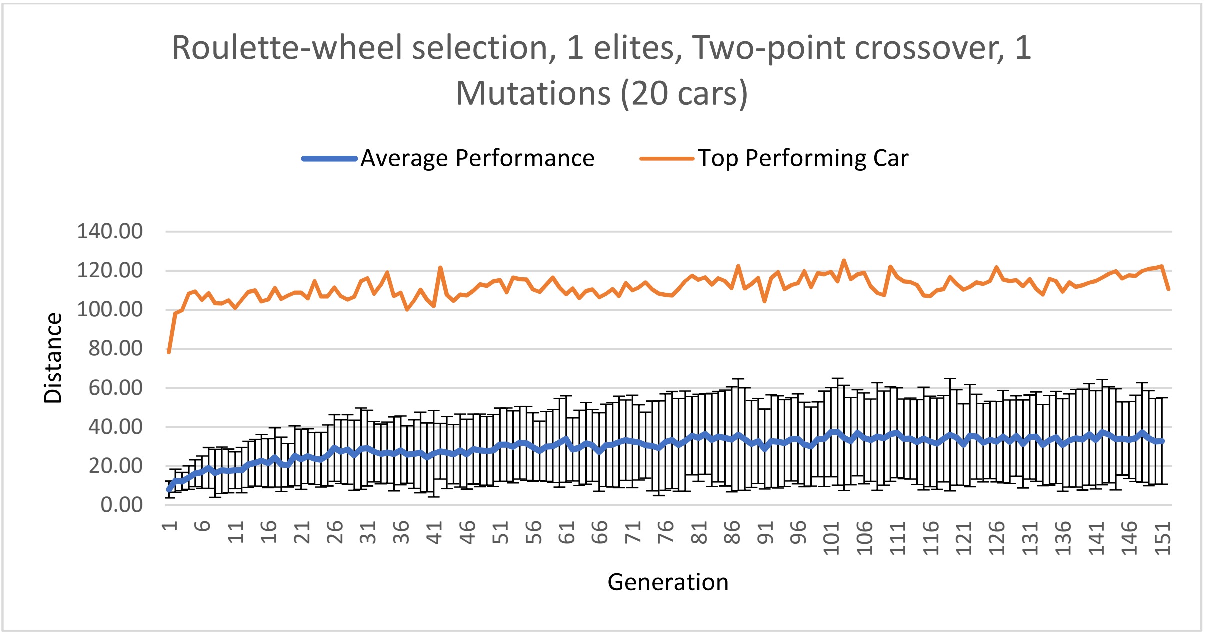


Figure 2

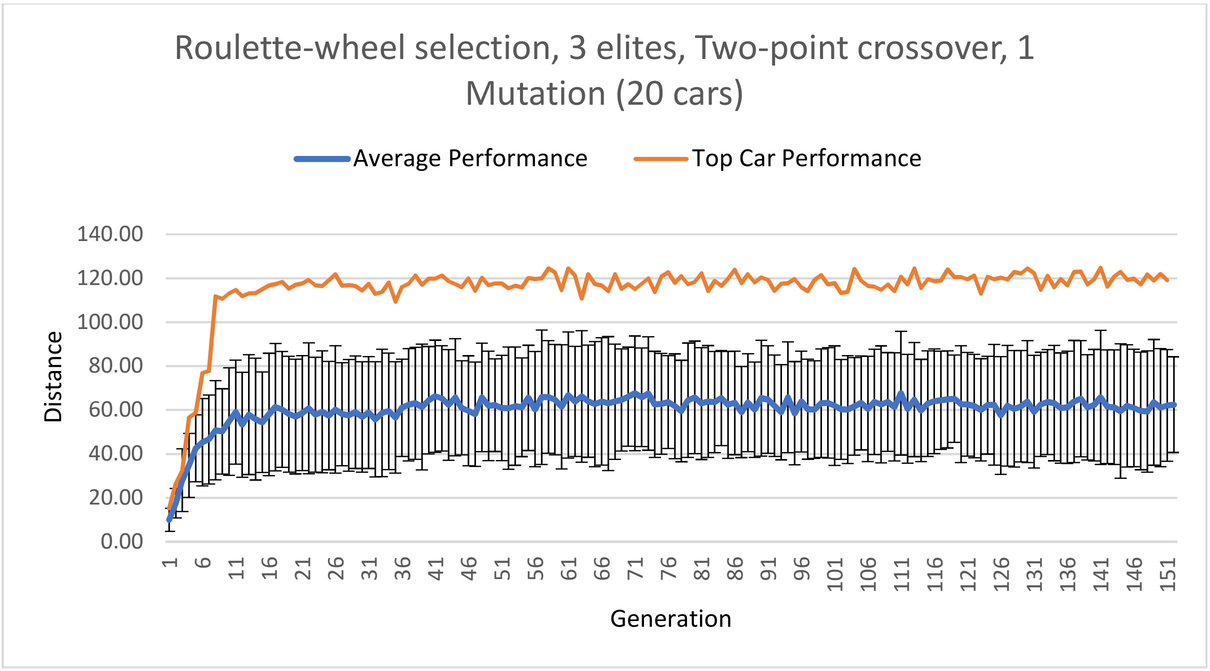
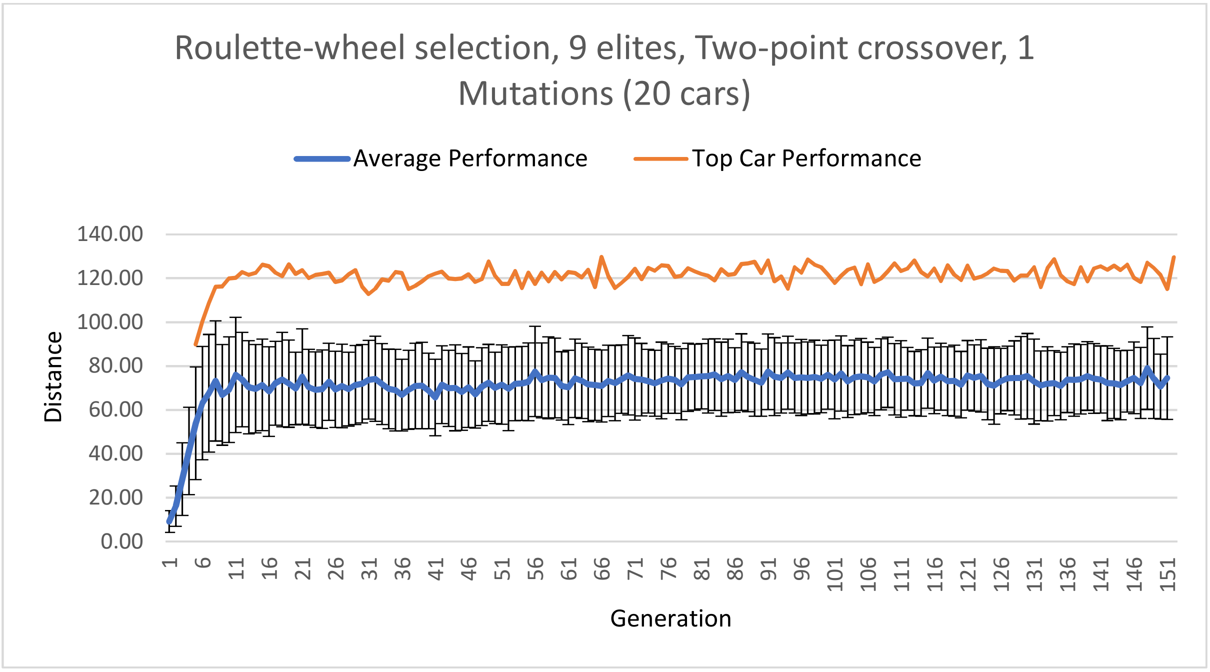


Figure 3



Selection Operator Analysis

The operator with the highest average performance using 1 elite is Tournament selection at a which is shown in Appendix B figure 1. This selection operator has the highest standard deviation, which relates to the difference in performance in the 10 sets of data which summarise the average performance, this brings questions to its reliance compared to the other operators.

The operator with the lowest standard deviation only using 1 elite is Uniform Random selection in Appendix A figure 1. But this operator has the lowest average performance of the 3 operators.

The selection operator with he highest performance using 3 elites is Roulette-wheel selection in figure 2 which barely above Tournament selection in Appendix B figure 2. Out of the two operators Tournament selection has a lower standard deviation but the difference between the two is not much. The operator with the over lowest standard deviation is Uniform random selection though by not much of a margin and it has got the lowest average performance.

The operator with the highest performance using 9 elites is Tournament Selection in Appendix B figure 3. The operator with the lowest average performance is Uniform random selection. Roulette-wheel selection and Tournament selection have the lowest standard deviation compared to Random-uniform selection.

Scoring Operator

Selection Pressure Increase

Figure 1-3 shows an increase in average performance the further the number of Elites are increased, and this also corresponds in Appendix A-C which uses a Different selection operator but also shows an increase in the average performance when the number of elites are increased.

The same principle is used in both versions of the Evolutionary algorithm because there is a larger area of bias where the best cars are favoured which applied to the number of elites are used.

There is a limit to the population ratio that should be elites compared to those that are not because the selection pressure effectiveness will decrease at a certain point which can be seen in the difference in average performance in figure 2-3 which is lower compared to figure 1-2, this can be attributed to the Selection operator because in Appendix Figure 1-3 this is not visible but the difference in performance for Appendix A Figure 1-2 is equal to Appendix A Figure 2-3 with the difference in performance not getting any bigger even with the larger margin of elites used.

Comparing the standard deviation of increasing the number of elites, the differences between the different version of the Evolutionary algorithms decrease as the number of elites are increased in all the different instances.

Critical Evaluation

Appendices

Appendix A

Figure 1

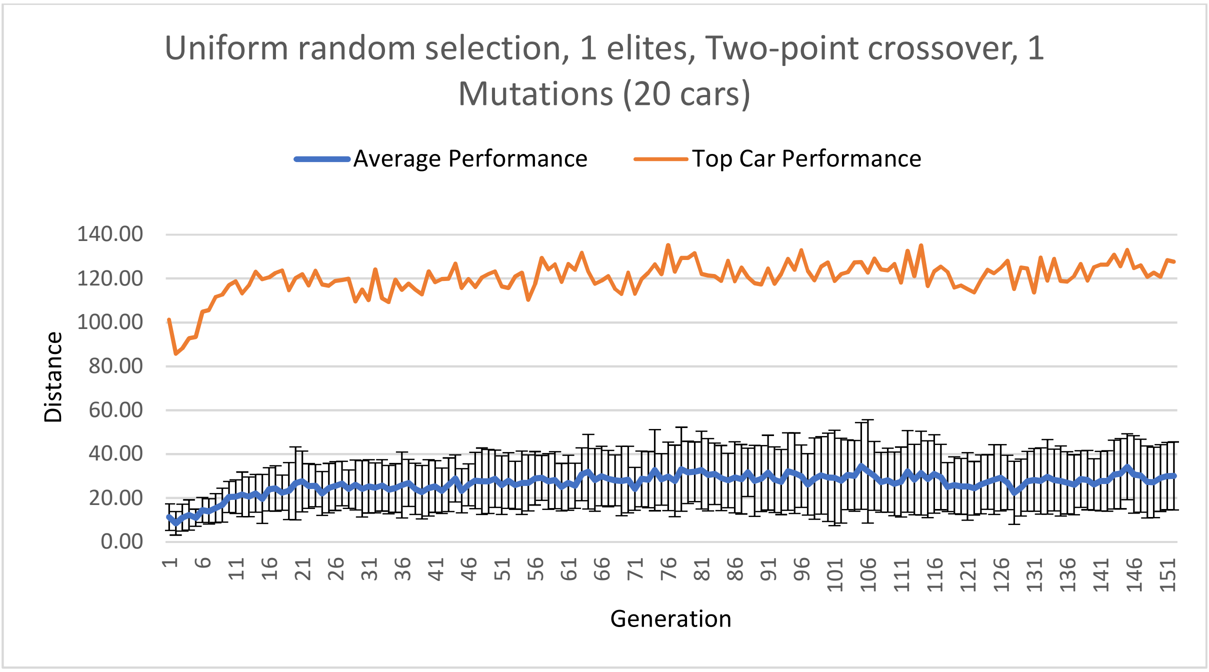


Figure 2

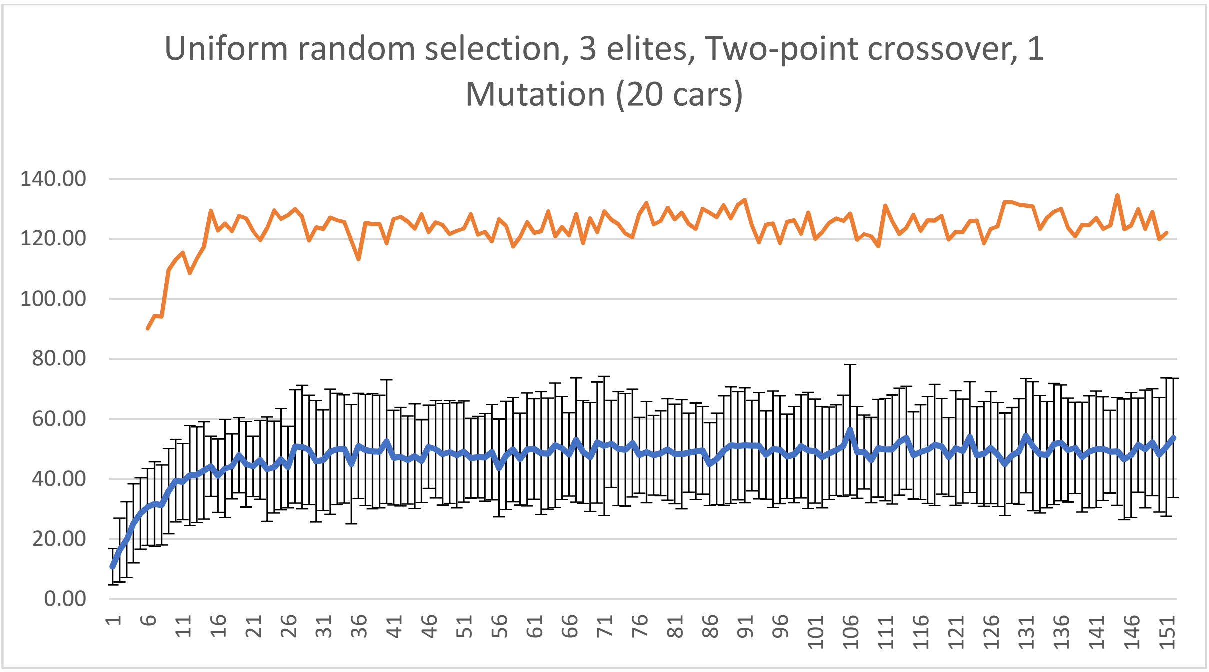
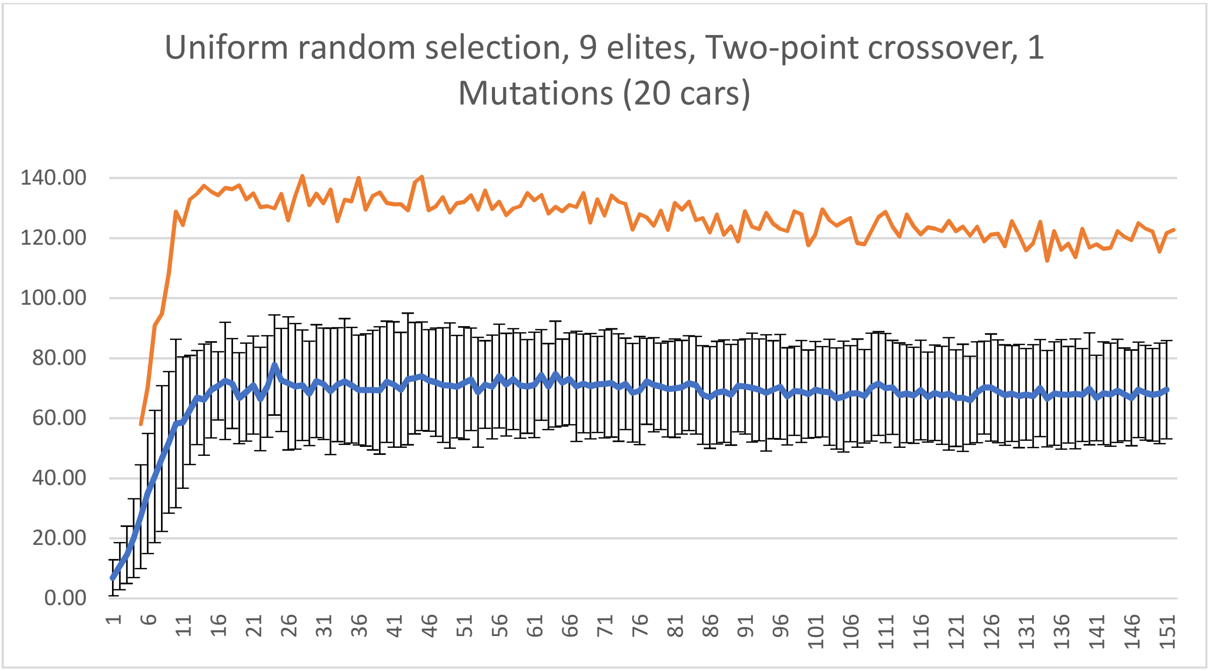


Figure 3



Appendix B

Figure 1

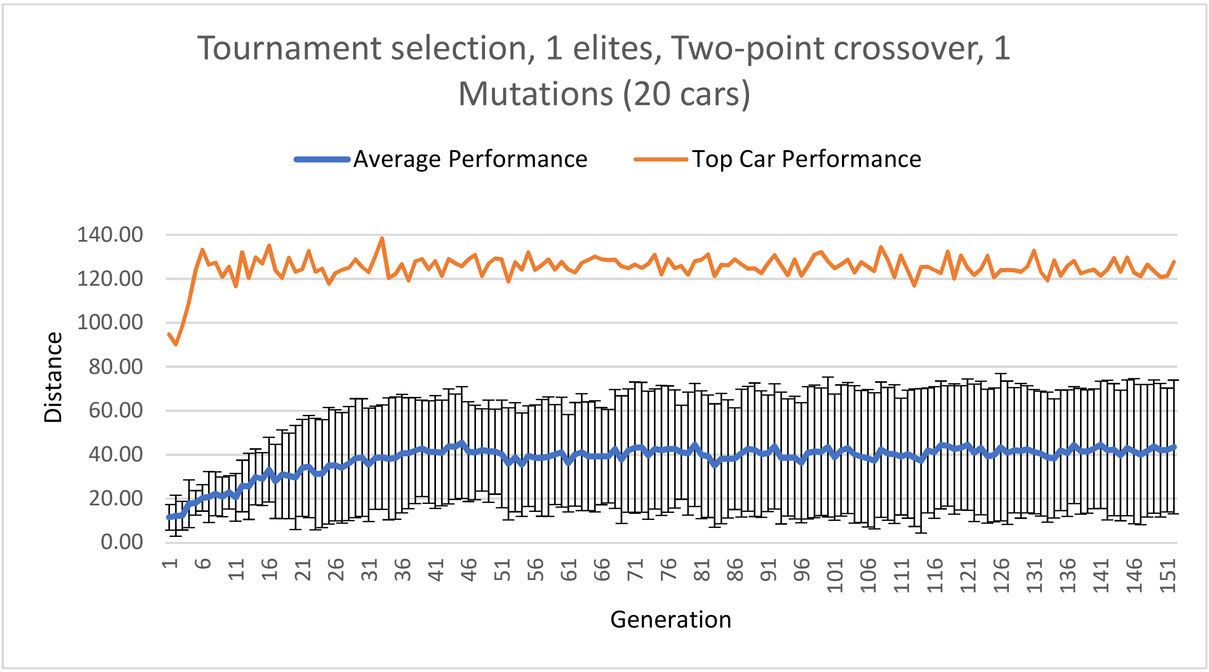


Figure 2

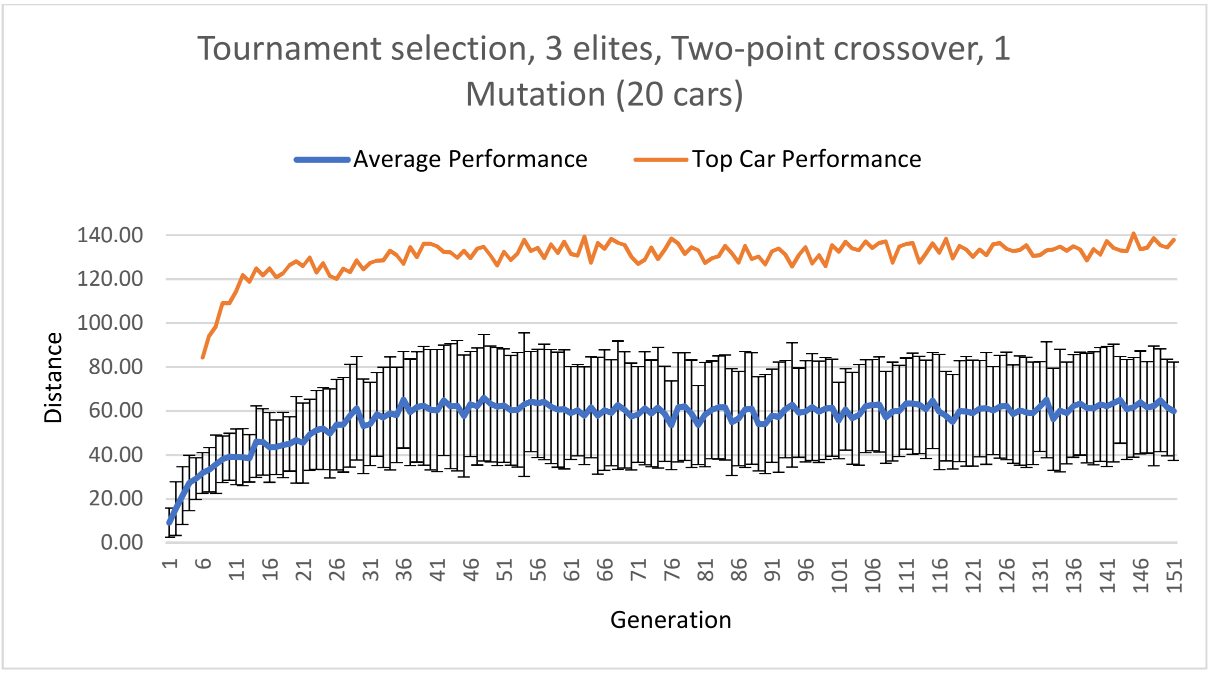
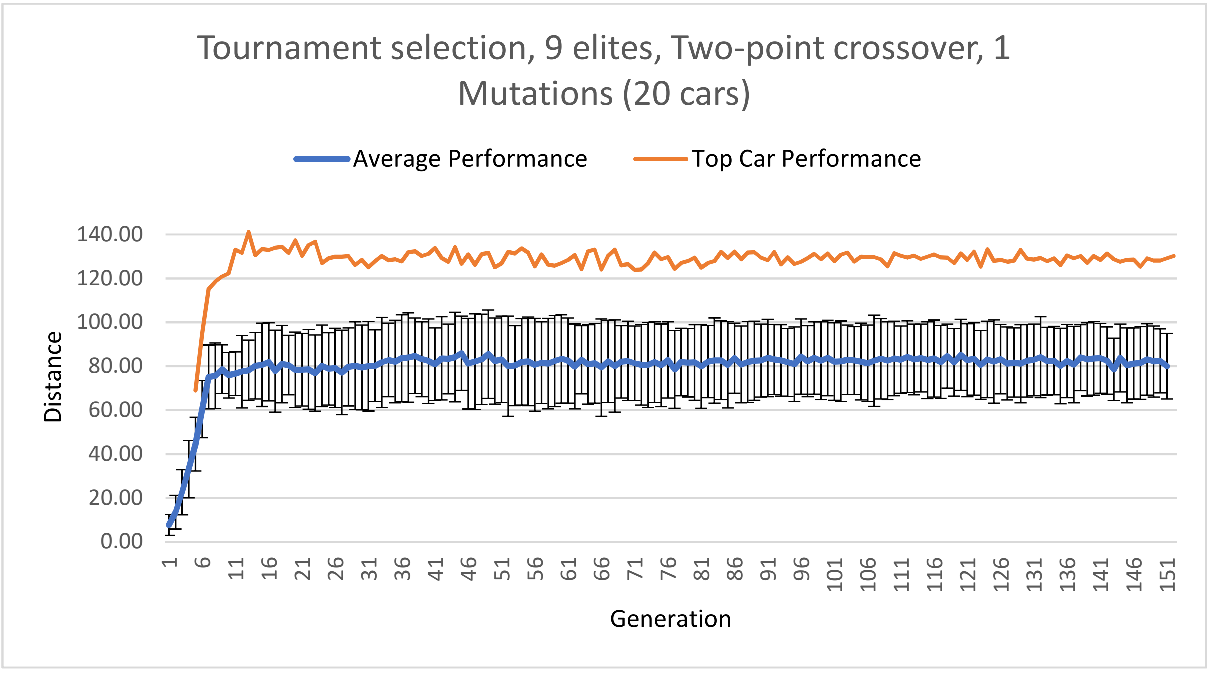


Figure 3



Appendix C

Figure 1

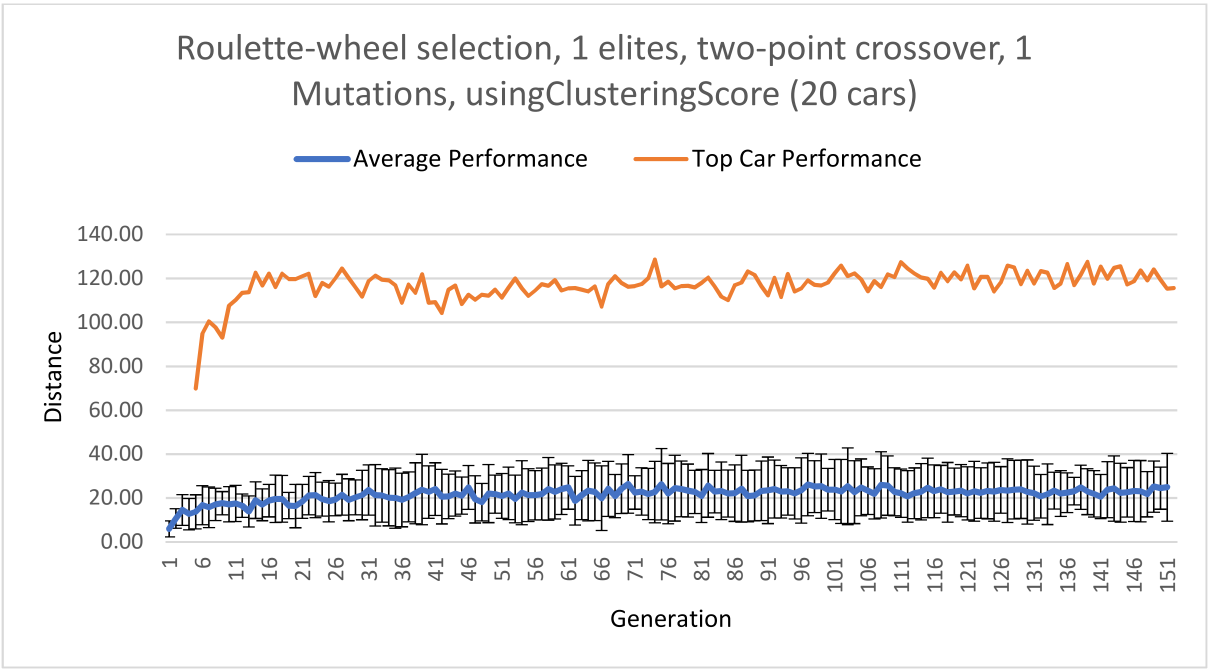


Figure 2

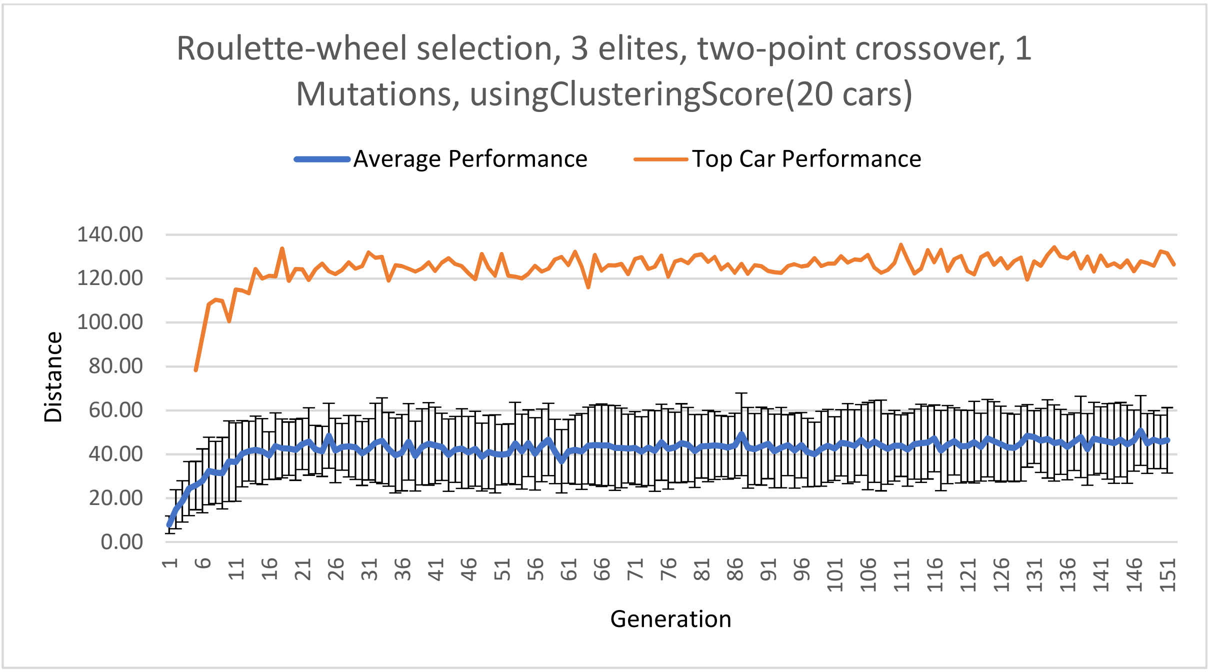
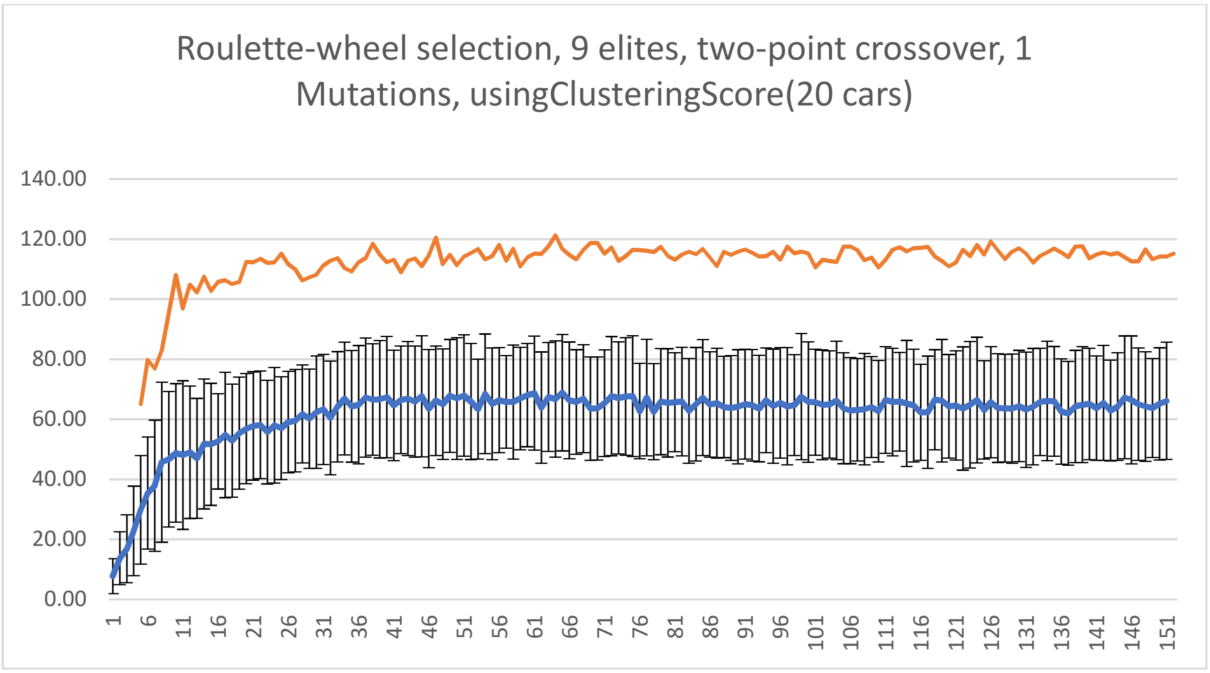


Figure 3



Annotated Bibliography

[1] A. Eiben and J. Smith, Introduction to evolutionary computing, 2nd ed. pp. 15-35.

> The common underlying idea among the various versions of Evolutionary algorithms is that given a population of individuals, operators are used to apply pressure to force evolution to these individuals over several generations.

[2] J. Khalid, Selection Methods for Genetic Algorithms, International Journal of Emerging Sciences, 2013. [Accessed 6 March 2019].

> Selection operators or parent selection operators (PSO) are aimed at exploiting the best characteristics of individuals of a given population. This type of operator chooses the most suitable individuals to have their recombined. Selection operators are varied in terms of how much selection is applied to the population, whether individuals are taken from the local or global population and the criteria in which an induvial is chosen

[3]"Crossover (genetic algorithm)", Academic Dictionaries and Encyclopedias. [Online]. Available: http://enacademic.com/dic.nsf/enwiki/302339#One-point_crossover. [Accessed: 07- Apr- 2019].

> Crossover operators within an Evolutionary algorithm are ways in which the algorithm takes more than one individual solution and recombines individual's data based on a set of criteria to reproduce children for the next generation of individuals.

[4]"Mutation (genetic algorithm)", Academic Dictionaries and Encyclopedias. [Online]. Available: http://enacademic.com/dic.nsf/enwiki/302648. [Accessed: 07- Apr- 2019].

Mutation operators are used within Evolutionary algorithms to maintain genetic diversity from generation to generation. The basic principle of the operator is that there is that data within an individual would have a Random mutation which helps keeps genetic diversity within the population preventing individuals becoming too similar which helps avoid local minimum solutions.

[5]"About npm | npm Documentation", Docs.npmjs.com, 2019. [Online]. Available: https://docs.npmjs.com/about-npm/. [Accessed: 10- Apr- 2019].

NPM is an open source Software Package Manager and allows access to a database of software packages and the package manager can be downloaded onto the client side. NPM allows version control of projects allowing for easier management and error correction during testing. The components consist of the website, Command Line interface and the registry which stores all the software packages

[6]"Node.js Introduction", W3schools.com, 2019. [Online]. Available: https://www.w3schools.com/nodejs/nodejs_intro.asp. [Accessed: 10- Apr- 2019].

Node.js is an open source server environment which provides a development environment allowing for JavaScript code to be run. The environment can run a various platform such as Windows, Linux, Mac OS and Unix. Node.js uses asynchronous programming allowing for a sequence of events to run side by side

[7]"chrome.storage - Google Chrome", Developer.chrome.com, 2019. [Online]. Available: https://developer.chrome.com/apps/storage. [Accessed: 11- Apr- 2019].

LocalStorage is a Google chrome web storage which works within the browser's own framework memory. There is an inherent restriction to adding data produced by the simulation to the Client of the computer because of browser security which is the main reason for LocalStorage being used

[8]"Chrome DevTools | Tools for Web Developers | Google Developers", Google Developers, 2019. [Online]. Available: https://developers.google.com/web/tools/chrome-devtools/. [Accessed: 11- Apr- 2019].

Google chromes provides DevTools which provides an in-build IDE to debugging and testing browser run scripts. This IDE was using throughout the process of implementing and testing the different components of the Evolutionary algorithm. The IDE provides an in-depth way of tracking the varying variables which made the testing process simple looking for complications. Directory access is also provided for accessing the different scripts while also providing tabbing for scripts to be viewed on the same window.

[9] D. Therens, Selection schemes, elitist recombination, and selection intensity. Utrecht University: Information and Computing Sciences, 1998, p. 2.