A Comparison of Evolutionary Algorithm Operators

Final Report CS39440 Major Project

Author: Jonathan Brooks (job64@aber.ac.uk)

Supervisor: Dr Christine Zarges (chz8@aber.ac.uk)

2nd May 2019

Version 1.0 (Release)

This report is submitted as partial fulfilment of a BSc degree in Computer Science (G400)

Department of Computer Science
Aberystwyth University
Aberystwyth
Ceredigion
SY23 3DB
Wales, UK

Declaration of Originality

I confirm that:

- This submission is my own work, except where clearly indicated.

- I understand that there are severe penalties for Unacceptable Academic Practice, which can lead to loss of marks or even the withholding of a degree.

- I have read the regulations on Unacceptable Academic Practice from the University's Academic Registry (AR) and the relevant sections of the current Student Handbook of the Department of Computer Science.

- In submitting this work, I understand and agree to abide by the University's regulations governing these issues.

Name: Jonathan Brooks

Date: 23/05/2019

Consent to share this work

By including my name below, I hereby agree to this project's report and technical work being made available to other students and academic staff of the Aberystwyth Computer Science Department.

Name: Jonathan Brooks

Date: 23/05/2019

Abstract

Evolutionary algorithms are optimization algorithms which can be used in a broad range of applications which can stem from Medical research to Gaming. Within the spectrum of Evolutionary algorithms there consist a lot of components that are of vital importance that can impact an overall solutions performance.

This body of work aims to investigate some of the varying components within the means of a 2D car racing simulator written in JavaScript. And to compare these components within the realm of how they each affect a solutions performance and how they compare with each other.

In this project I will create an Evolutionary algorithm which can provide a framework to use multiple operators and provide a means of outputting data to evaluate the performance of varies versions of Evolutionary algorithms.

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, critical and data 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 [9]. 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 [10].

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 analysis to apply a change to the Car scores to optimize the selection process. The assumption being made is that a global pattern of good performance can be applied to the individual datapoints/chromosomes of a car, applying a score or rank to said datapoint to be added to a new summed score for the car. For example, a Local generation car is ranked in the middle of the population, applying the new score finds the datapoints of the car is rank in the top set of the population instead due to past existing cars being ranked at the top with similar datapoints [11].

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 using is that the top performing car/s will be passed onto the next generation and they will still be used in the crossover process.

Reproduction Increase

During the recombination process of the Evolutionary algorithm the cars that are selected to be parents are removed from the selection process so that they do not produce more than children. This mechanism will not remove a car or a set of cars from the selection process after they recombined once based on a set of criteria. The criteria will use 3 selection methods for choosing which car will not be removed from the selection process such as Roulette-wheel selection, Tournament selection and Random uniform selection.

The car that will not be removed will have the chance to be selected again to pass its data onto the next generation.

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.

The data collected during the project was produced via the simulation and taken to be put into an excel spreadsheet that provided a means of compiling graphs of the gathered data. The data collected consisted of the average performance of simulated rounds that have the average performance of the cars. One version/combination of Evolutionary algorithm operators have an average performance that ranges just over 151 generations. The average performance of the Evolutionary algorithm consists of 10 data sets of the simulation being run over 151 generations 10 times and averaged to be put into graphs with a corresponding standard deviation calculated over the 10 data sets.

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

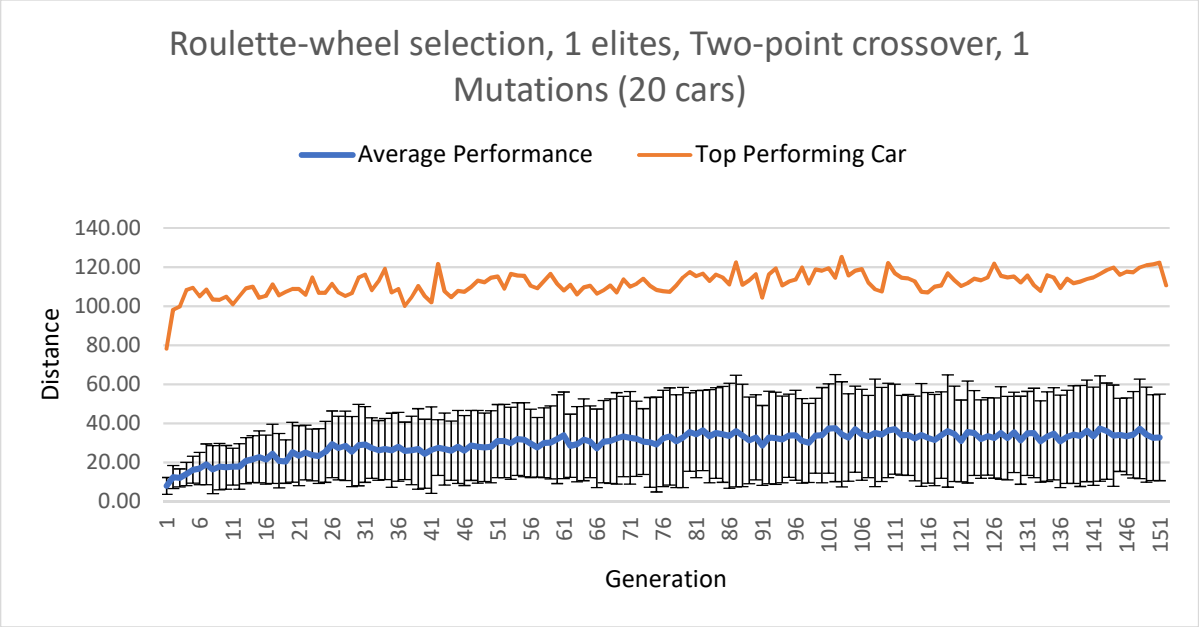


Figure 2

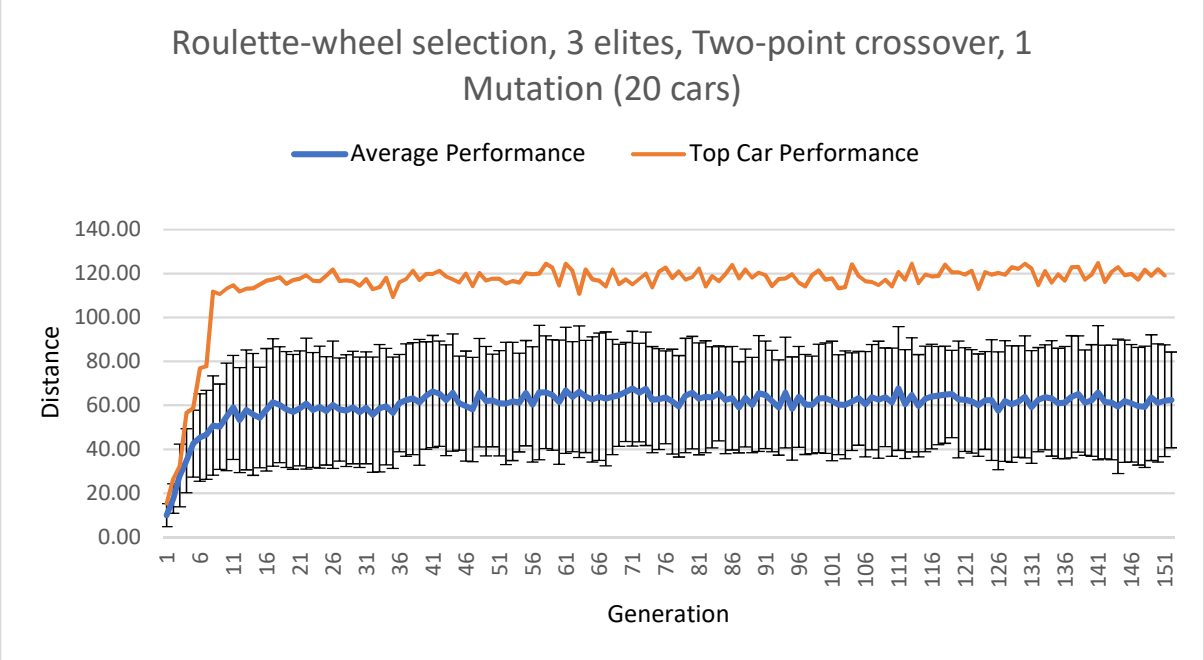
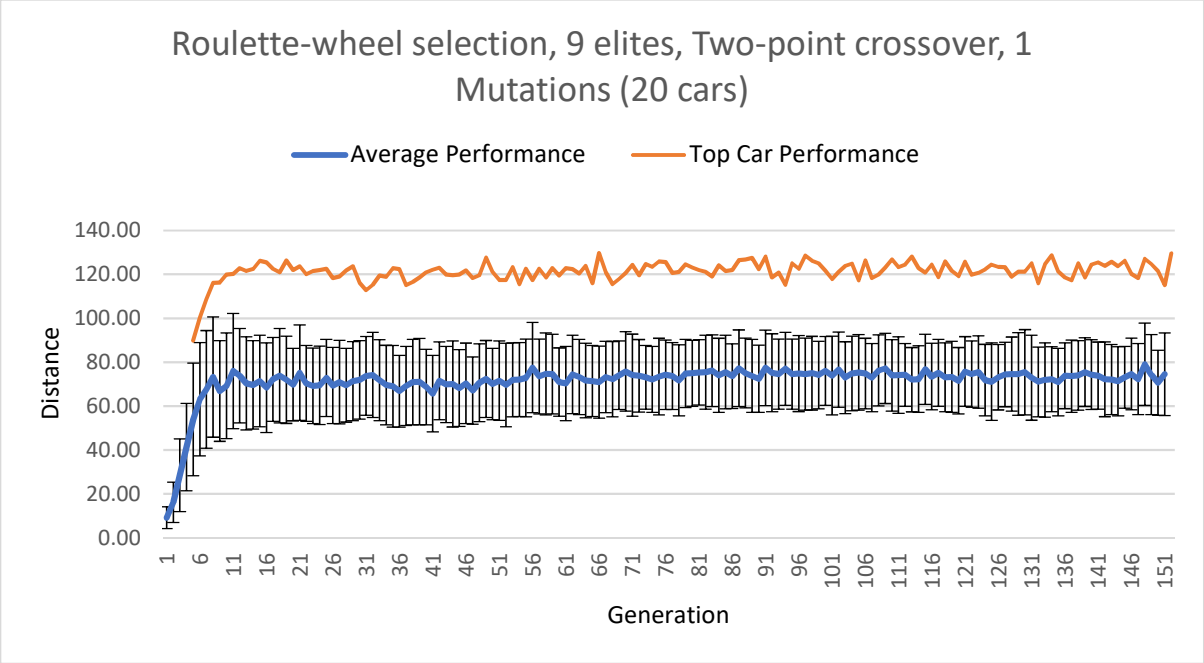


Figure 3



Selection Operator Analysis

The following with be an analysis on the Selection operators Tournament selection, Uniform random selection and Roulette-wheel selection. The analysis will look at the top/lowest performing operators.

1 Elite Analysis –

The operator with the highest average performance of 40 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 lowest performing operator has an average performance of 32 in Appendix A figure 1 Uniform random selection, this operator also has the lowest standard deviation by a large margin compared to the other two operators.

All 3 operators that use 1 elite all have an average performance that range between 30 and 40. The biggest difference between the 3 is the difference between the standard deviation. The lowest Uniform random selection is the most reliant due to the low standard deviation because of the higher probability of the results being around the recorded average performance. With width of Tournament selections standard deviation goes beyond the average performance of Uniform random selection which shows the increased possibility/range of the average performance of Tournament selection not being as close to the average performance compared to Uniform random selection.

3 Elites Analysis –

The selection operator with the highest average performance of 60 using 3 elites is Roulette-wheel selection in figure 2 which is equal to Tournament selections performance in Appendix B figure 2. Tournament selection though reaches its peaks performance in further

generations at generation 36 compared to Roulette-wheel selection reaching peak performance at generation 11.

Out of the two operators Tournament selection has a lower standard deviation but the difference between the two is small both having a standard deviation over 20. The operator with the lowest standard deviation is Uniform random selection with distance being lower than 20 though by not much of a margin and it has got the lowest average performance at 50.

9 Elites Analysis –

The operator with the highest average performance of 80 using 9 elites is Tournament Selection in Appendix B figure 3. The operator with the lowest average performance of 73 is Uniform random selection in Appendix A figure 3. Roulette-wheel selection and Tournament selection have the lowest standard deviation compared to Random-uniform selection with the highest.

The standard deviation of Tournament selection decreased when the number of elites have increased it being the smallest having 9 elites with the highest performance. Comparing this with Uniform selection the standard deviation has increased the number of elites has increased from 1-3 but stayed around equal from 3-9 elites which is the opposite of Tournament selection which decreased on every elite increase 1-9. Roulette wheel selection saw a slight increase in standard deviation between 1 and 3 elites, but the standard deviation got small bet 3 and 9 elites it being the smallest out of roulette-wheel selection.

Scoring Operator Analysis

The following will be an analysis of how the different Evolutionary algorithms that use elitism are affected by using the Re-scoring operator that uses cluster and KNN.

1 Elite Analysis –

The version that has the highest average performance using 1 Elite is Tournament selection with the average performance at 25 in Appendix D figure 1, This operator has a lower average performance compared to Tournament selection in Appendix B figure 1 that does not use the Scoring operator by a difference of 15. And is even lower than the lowest average performing operator Uniform random selection in Appendix A figure 1 that do not use the Scoring operators by a difference of 7.

3 Elites Analysis –

Using 3 elites the operator with the highest performance is Roulette-wheel selection with an average performance of 43 in Appendix C figure 2, This average performance is lower than the highest average performance of the non-scoring operators by a difference of 17. The scoring roulette wheel selection has an increased performance using more elites with a difference of 10 with the standard deviation increasing by a 3rd of the 1 elite standard deviation in Appendix C figure 1.

The lowest average performance of 40 is Uniform random selection in Appendix E figure 2. There is an increase in performance compared to using 1 elite by a difference of 10 in

Appendix E figure 1 which is the same as the highest performing scoring operator but there is a larger increase in standard deviation for Uniform random selection compared to Roulette-wheel selection. This large increase in standard deviation when increasing the number of elites is not present in the non-scoring Uniform random selection.

The 3 operators all have a small difference in terms of standard deviation but the one with the smallest standard deviation is Roulette-wheel selection but only by a small margin.

9 Elites Analysis –

The highest performance operator using 9 elites is Tournament selection with an average performance of 80 in Appendix D figure 3. This average performance is equal to the non-scoring tournament selection operator of 80 with around the same standard deviation which is the highest average performance. The average performance of the new scoring Tournament selection operator using 9 elites increased from used 3 elites with a difference of 20 with the standard deviation staying near enough equal to what it was with 3 elites. The standard deviation did not get smaller increasing the number of elites from 3-9 compared to the non-scoring tournament selection which had the standard deviation decrease as the number of elites increased.

The lowest average performance of 64 is Roulette-wheel selection in Appendix C figure 3. This low average performance is less than the lowest average performance of the non-scoring operators Uniform random selection in Appendix A figure 3 by a difference of 9.

The standard deviation of Uniform random selection is the lowest compared to Tournament selection and Roulette-wheel selection which are near equal in margin difference compared to Uniform random selection which is the second best in Average performance at 70 in Appendix E figure 3.

Conclusion

The highest performing analysed versions of the Evolutionary algorithm use 9 elites but comparing the highest performing version that uses the scoring operator Tournament selection with the one that does not they use the same Selection operator Tournament selection. The one that uses the scoring operator has a similar standard deviation with little difference in margin difference.

There is not any major difference that can be compared which would lead to the Scoring operator not having any noticeable effect on the car's performance.

Multi-Mutation Analysis

The assumption for using multi-mutation is to keep diversity within the population and to help increase the overall population performance when the performance gets stuck. This is done by increasing the number of mutations applied to the weaker performing cars in the population increasing their chances of getting a god mutation trait.

1 Elite Analysis –

The Multi-mutation operator with an elite of 1 has an average performance of 23 in Appendix H Figure 1 compared to other operators which use the standard singular mutation and the same number of elite's, Multi-mutation has the lowest performance in comparison. Multi-mutation does however have the lowest standard deviation compared with the other 3 Evolutionary algorithms which points to better reliability in repeating the same results.

3 Elites Analysis –

Comparing the Multi-mutation operator with 3 elites have an average performance of 38 in Appendix H figure 2 with the standard mutation there is still large difference in average performance with the standard mutation being just around 60 in the 3 different versions that use the standard mutations. The difference in performance between the 1 elite multi-mutation and the 3 elite multi-mutation is like the standard mutation operators' difference of performance around 20 which can be related to the use of elites not specific to the Mutation operator.

The standard deviation for the 3 elite multi-mutation is still lower than the standard mutation operators, there is a small marginal increase in the standard deviation of the multi-mutation when increasing the number of elites from 1 to 3. The increase of standard deviation is also present in the standard mutation though the difference of standard deviation increase is a lot more in the standard mutation then Multi-Mutation.

9 Elites Analysis –

The 9 elite multi-mutation has an average performance of 61 in Appendix H figure 3 compared to the standard mutation it's the lowest but it has the lowest standard deviation which is a common factor comparing the lowest elite multi-mutations with the standard mutation. This can also be attributed to the average performance being lower than the standard mutation average performance.

Cluster Mutation Analysis

Cluster Mutation is assumed to mutate a random cars data point to a local best which is gotten via KNN on a cluster of the data points data type.

1 Elite Analysis –

Cluster mutation with 1 Elites has an average performance of 40-50 between generation 66 and 151 in Appendix I figure 1. Which is the highest in performance compared to Multi-mutation and the standard mutation operators which have at least a difference of 10, the closest operator is Tournament selection which reaches an average performance of 40 in Appendix B figure 1 compared to cluster mutation with its highest reaching 50 by generation 145.

Comparing the standard deviation of Cluster mutation to the second-best average performing operator Tournament selection, the standard deviation is lower by about a 3rd of Tournament selection's standard deviation.

3 Elites Analysis –

Cluster mutation using 3 elites has an average performance of 60 in Appendix I figure 2, It's higher than Multi-mutation in Appendix H figure 2 by a large margin with a difference of 22. And comparing the Cluster mutation with the standard mutation Appendix A-B figure 2 the average performance is near enough equal with little to no difference in average performance with the standard operators also equalling around 60. The standard deviation of the Cluster mutation is around equal to the standard mutation and higher than Mutli-mutation.

When comparing the cluster mutation with 3 elites and 1 elite there is a small increase change when increase from 1 elite to 3 compared to other Mutation operators, this can also be seen in Multi-mutation when comparing with the standard mutation operators the increase change is small in comparison.

9 Elites Analysis –

Cluster mutation using 9 elites have an average performance of 63 in Appendix I figure 3 which is equal to Mutli-Mutation that uses 9 elites but still much lower when compared to the standard mutation operators that use 9 elites with their average performance being above 70 in Appendix B figure 3.

When looking at the previous increase from 1 elite to 3, and the smaller increase in performance compared with the standard mutation it can be expected that the change would be even smaller, but the performance stayed the same. Though the average performance stayed the same the standard deviation is much lower compared to the standard mutation operators by a large margin.

Cluster Mutation Selection Pressure Increase

The selection pressure within cluster mutation can be changed by changed the value of neighbours that are gotten through the KNN process of getting data from clusters. A selection pressure increases it done by increase the original number of gotten neighbours. The cluster mutation was used alongside Roulette-wheel selection, Two-point crossover and using 1 elite.

The first cluster mutation uses KNN to get the neighbouring 40 data points in a cluster, the average performance of using this Is 44 in Appendix K figure 1. Comparing this to cluster mutation which gets 200 neighbouring data points in Appendix K figure 2 the average performance is the same but the average peaks in performance in earlier generations by 20.

The standard deviation of cluster mutation getting 40 neighbours compared to 200 is a lot lower when compared by a large margin, this is also the same with increasing the number of neighbours to 700 the average performance and the standard deviation stays the same.

The previous assumption made for the implementation of cluster mutation was to increase/decrease a cars data point to an optimal value using all previous existing cars. Increasing the number of neighbours gotten would increase the target optimal value which

would in tern increase the value change of a cars data points to be closer than getting a lower number of neighbouring data points.

The data shows that even increasing the number of neighbours there is a higher risk of the Evolutionary algorithm not performing above the recorded average performance due to the higher standard deviation, because the cluster mutation that gets 40 neighbours has the lowest standard deviation making it more reliant or more likely for the algorithm to perform around or above the recorded average performance.

Selection Pressure Increase Analysis
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 number 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.

Reproduction Increase

This section will discuss the performance of the Reproduction increase mechanism which allows a car to have more than two cars by not removing it from the selection process. The assumption before implementation for this mechanism was to apply a highest selection pressure by increase the amount by which the best cars data is spread to the rest of the population.

Selection Operator Analysis
Reproduction 1 Analysis –

The operator with the highest average performance using a reproduction increase of 1 is Roulette-wheel selection with an average performance of 29 in Appendix F figure 1 compared to Uniform random selection with an average performance of 25 in Appendix G figure 1. Roulette-wheel selection also had a smaller standard deviation though only by a small margin compared to Uniform random selection.

Reproduction 3 Analysis –

Both operators have a near equal average performance with the Reproduction increase of 3 with the average performance of both operators sitting between 35 and 40. The standard deviation are also near the same in margin difference in Appendix F-G Figure 2.

There is a notable performance increase from increasing the reproduction from 1 to 3 by a difference of around 7-10. There is also a slight increase in the standard deviation with both being relatively similar but bigger than the lower reproduction value.

Reproduction 6 Analysis –

The operator with the highest average performance with the Reproduction increase of 6 is Roulette-wheel selection with an average performance of 34 from generation 36 onward in Appendix F Figure 3 compared to Random uniform selection the average performance changes from 28 to just above 40 after generation 76 in Appendix G Figure 3. Random uniform selection has the lowest standard deviation compared to Roulette-wheel selection.

The performance difference from increasing the reproduction value from 3-6 has not changed with the best being 40 which is the same with the best performing Reproduction 3 operator. The standard deviation of Roulette-wheel selection has not changed and stayed around the same with a slight increase in the standard deviation by the left number of generations. Standard deviation for Uniform random selection has a lower standard deviation with an increase in the reproduction value but this is not a big change.

Multi-Mutation Analysis
Reproduction 1 Analysis –

Multi-mutation with 1 increase reproduction has an average performance of 15 in Appendix J figure 1, Multi-mutation in comparison to the standard mutation with the same reproduction value is quite low where the standard mutation is on average around just below 25 in average performance in Appendix F-G figure 1. The standard deviation of the multi-mutation is lower than the standard mutation, but the average performance of the multi-mutation is also lower.

Reproduction 3 Analysis –

The Multi-mutation with an increase reproduction of 3 has an average performance of 28 in Appendix J figure 2 compared with the standard mutation with an average performance of 38. Comparing the average performance increase after increasing the Reproduction value to 3 there is not a large increase in the average performance this is also seen in the standard mutation which means it's an inherent trait of using the Reproduction increase.

Reproduction 6 Analysis –

The average performance of Multi-mutation with a reproduction increase of 6 in Appendix G figure 3 is near enough the same with having the reproduction value of 3 in Appendix G figure 2, compared to the standard mutation there is a slight increase in the average performance in Appendix F-G figure 2-3 but this is not seen in Multi-mutation.

The overall standard deviation of the multi-mutation operator is small compared to the standard mutation, but this is related to the difference in the average performance between the two operators where multi-mutation overall has a much lower average performance base compare to the standard mutation operator.

The performance increase difference after increasing the reproduction value is similar in the first increase from 1-3 but not for 3-6 though there was a slight increase in the standard mutation this did not occur in the Multi-mutation though there was a sudden jump in performance during generation 126 in Appendix J figure 3.

Cluster Mutation Analysis
Reproduction 1 Analysis –

The cluster mutation that uses a reproduction value of 1 has an average performance of 40 which peaks at generation 56 in Appendix L figure 1, which is above multi-mutation by about 25 in Appendix J figure 1. And above the standard single mutation by 11. The standard deviation of the cluster mutation is at 30 which is one of the highest out of all the data collected.

Cluster mutation has a higher average performance than the standard mutation operator by a difference of 11 from Appendix F figure 1. And the also being higher than the Multi-mutation operator with a difference of 25 in Appendix J figure 1. The standard deviation of the cluster mutation compared to the other mutation operators is much higher with it being above 30.

Reproduction 3 Analysis –

Cluster mutation used with a reproduction value of 3 has an average performance of 64 in Appendix L figure 2 at generation 71 where the performance stops increasing. The performance peaks at generation 21-41 to an average performance of 80 and then the performance drops to 64. The performance compared to multi-mutation is higher with a difference of 36, and higher than the standard mutation with a difference of around 20.

Comparing the cluster mutation before the value increase from 1-3 there is a relative increase of 24 in average performance with a slight decrease in standard deviation. The decrease in standard deviation is quite different from data collected in the standard mutation in Appendix G-H figure 1-2 and multi-mutation in Appendix J figure 1-2, where the standard deviation either stayed the same or slightly increased when increasing the reproduction value from 1-3.

Reproduction 6 Analysis –

The cluster mutation using a reproduction of 6 has an average performance of 58 in Appendix L figure 3 which compared to before the increase in the reproduction value the average performance was higher at just above 60 in Appendix L figure 2. The standard deviation of reproduction 6 is lower than before the increase by a third of the standard deviation of reproduction 3 in Appendix L figure 2.

Cluster mutation has the highest average performance compared to both the standard and Multi mutation operators by a large margin with the standard mutation only being second with a difference of 18. The standard deviation of the cluster mutation is also the lowest being slightly less than Multi-mutation in Appendix G figure 3 but there is a large difference in the average performance making the Cluster mutation more reliable.

Selection Pressure Increase Analysis

The selection pressure changes for the Reproduction Increase mechanism is changed by increasing and decreasing the number of Cars within the population that can be chosen to reproduce children into the next generation twice instead of once. The following is an analysis of the effect on the average performance by increasing the number of Cars within the population that has the chance to reproduce twice.

Looking at the two instances of Evolutionary algorithms which used different selection operators, there was an increase in performance in both instances. Looking at the first increase of 1 Reproduction increase can amount to 1 of the cars within the population passing down its data to 4 new cars instead of two, the selection method used for choosing the cars is Tournament selection so there is a high chance the chosen car being in the top bracket of performing cars.

Increasing the Reproduction value from 1-3 has show an increase in the average performance of the two selection operators with a slight increase in standard deviation. And increasing the reproduction value from 3-6 shows no- maximum increase in the average performance of the two operators. In one instance there is a decrease in standard deviation but not by a large enough margin to have a big effect.

Overall the Reproduction operator does increase the average performance of an operator with a selection pressure increase but not by a large margin and the limit of the Reproduction value being 3 looking at performance not increasing while increasing the reproduction value from 3-6.

Overall Comparison

The following will be a comparison of ho the operators performed using Elites and the Reproduction increase.

Selection Pressure Comparison

The two types of selection pressure used in the Evolutionary algorithm are Elitism and Reproduction increase, I test both types with 3 increases in Selection pressure and the following will be a comparison of their varying performance changes and how they compare to each other at each Selection pressure increase.

The selection pressure changes between the two types vary with using elites having the highest average performance in each iterative increase in selection pressure. Elitism though all the Selection pressure increases have a higher performance compared to the Reproduction increases. During the second and third selection pressure increase for the Elites from using 3 to 9 elites there is a noticeable average performance increase from 60 to 80 where compared to the Reproduction increase from 3 to 6 the average performance near

enough stays the same. There is also a drastic decrease in the standard deviation when the selection pressure of the elites increases from 3 to 9.

Overall Elitism has a bigger effect in terms of influence of the simulation's selection pressure compared to the Reproduction mechanism though it is seen in the data Reproduction does hold some influence in increasing the average performance not enough to match Elitism.

The best operator performing selection operator from the Elites is Tournament selection with an average performance of 80 with one of the lowest standard deviations with a selection pressure of using 9 elites in Appendix B figure 3. Compared to the best Reproduction selection operator with an average performance of just above 40 Roulette-wheel selection with a selection pressure of 6 reproduction in Appendix F figure 3.

Mutation Comparison

When comparing Cluster mutation and Multi-mutation the assumptions made for either are difference as the Cluster-mutation is meant to optimise a cars data point to a local best using KNN and cluster. Multi-mutation however is meant to provide data diversity for the bottom ranked cars on the assumption that it can overcome the bottleneck of when the average performance peaks and stays the same.

The highest average performance of the Multi-mutation operator using elites is 61 in Appendix H figure 3 when compared to the Reproduction Multi-mutation operator with the highest average performance of 28 in Appendix J figure 2. This is a large difference in performance for-which the biggest influence of these performances are the Selection pressure mechanisms used. Because the Reproduction mechanism has proven to have a lower average performance in multiple versions of the Evolutionary algorithm compared with Elitism which in all instances is better in average performance when compared.

The cluster mutation with the highest performance of 63 uses Elitism as a selection pressure mechanism in Appendix I figure 3 compared with the Reproduction mechanism with an average performance of 58 in Appendix L figure 3. The Elitism cluster mutation also has the lowest standard deviation when compared.

When comparing the two Multi and cluster mutation operators with the standard single mutation operator in terms of the best average performance, the single mutation operator is the best with the highest average performance of above 80 in Appendix B figure 3.

Critical Evaluation

This section will provide an analysis on the work done during the project and whether further improvement could of be done and whether some could have been done differently.

Aim

The aim of this project is to provide a comparison of Evolutionary algorithm operators how the varying components effect the performance of an Evolutionary algorithm and how they compare to operators of the same type. Overall the aim of this project has been completed with the provided solution providing an Evolutionary algorithm with the specified operators mentioned in the Design section. The solution also provides a means of collecting data such as a cars generation average performance over 150 generations.

The solution though meeting the aim criteria could have been improved if more time is provided in different areas such as a more efficient data collection method which could reduce time spent setting up simulations. More operators could have been implemented providing more versions of an Evolutionary algorithm which could be compared. More information could have been gotten from the simulation such as specific car mutation rates, and crossover ranges used for producing the next generation of cars. And a Cars data point average performance where the best car components can be compared and analysed.

Objectives

This section goes over information about the specified object that were set for this project to complete by the deadline of the project.

Produce an Evolutionary algorithm with varying operators –

One of the aims of the project was to provide an Evolutionary algorithm which would provide the means to produce data that could allow someone to do an analysis on the performance of said Evolutionary algorithm. An Evolutionary algorithm was successfully implemented with the operators mentioned in the design section of the report implemented as a component of the Evolutionary algorithm. Two of the operators that were implemented were not originally considered but were implemented further into the project when it was considered that the data that would be collected would provide interesting results, these operators were Cluster-Mutation which was recommended during the Mid-project demonstration and a Hybrid/Mixed tournament selection which is discussed in the Design/Selection operator section.

A decision was made to try to implement the operators as separate as possible in terms of programmability and file organisation which helped manage the function handling and speed of the project which prevented confusion as to where things were. Separating the supporting functions of the operators as much as possible reduced the dependence the operators might have on each other and reducing possible risk of one error hurting the performance of more than one operator which stops the need to do repeat testing on more than the needed number of things.

Each of the operator's type were separated into different JavaScript files such as Selection, Crossover and Mutation which helped manageability. These JavaScript files were then

included into a higher-level file with accessibility to the separate lower level operator files. The higher-level file “manage-round” handled the overview of the Evolutionary algorithm such as passing and receiving data to the operators and passing the new generation back to the Simulation.

Given more time to the project I would implement more operators for Selection, Crossover and Mutation that would provide more variety in usage.

Collect performance data for Evolutionary algorithm operators –

The second aim of the project was to gather data that correspond to an Evolutionary algorithms performance of producing cars over a period of generation and producing a final performance with the expectation of the average performance of the Simulated cars to increase over the increasing number of generations.

Data collection was successful by adding the data into Google chromes local Storage and manually copying and pasting the data into an excel spreadsheet which corresponds to the combination of Evolutionary algorithm operators. This solution was used instead of sending the data to a server because the implementation would need the setting up of a server and changing the file format of the data for it to be added into a spreadsheet of setting up a python script to analyse the data and put them into graphs. The solution used was tedious and required constant attention for the completion of simulation runs and manually putting the data into a spreadsheet.

For one data collection of a combination of Evolutionary algorithm operators to be collected the Cars simulations had to run for between 45 minutes to an hour. This required the Desktop used to be on for 6-7 hours a day running simulations. After investigating the simulation software, I could not find a way to decrease the running time of running a simulation, but I could automate the re-running of a simulation which did not require me to re-run the simulation for the same version of Evolutionary algorithm.

I changes were to be made to this part of the project I would better define what combination of operators will be tested and provide a more automated means to testing which does not require tedious management of setting of different simulation runs, this would reduce the time required to manage data collection which can be better spent creating and expanding pre-existing operators.

The data was successfully collected during the end of the project taking up a lot of time over a 3 week period but just before reach this end of the project I had not realised the extent of the amount of data that would be collected and the time required for the collect to be done. This part of the project was not well predicted which did not help in the planning section of trying to come up with solutions to better collect the data from the Simulation. In the week of ending the data collection I realised that a lot of more different types of data could have been gotten such as the previously mentioned Mutation rates for the cars and the different car form types. This data could have helped better compare the mutation operators providing better insight into how they could have been improved.

More data could have been testing with the pre-existing operators such as the Hybrid Tournament selection and the standard Tournament selection via changing the sub-set array size effecting the selection pressure of cars.

Design

The design of the solution was mostly just a general overview of what was to be implemented in the Evolutionary algorithm and a time-frame of when it's to be implemented by. The design also changed during the project by implementing more than the pre-determined number of operators after coming up with them or finder other operators.

A lot more planning could have been done for the testing part of the project by applying a more Test-Driven Development approach mixing it with the Scrum methodology.

Project Management

The management of this project followed the Scrum methodology, but following this methodology provided flexibility I needed to micromanage the different Sprints in smaller objectives every time I started a new sprint which provided an efficient means of planning since a detailed Design specification was not used.

Scrum Values

Following the Scrum methodology, the implementation of the Evolutionary algorithm was split in the different components that were to be implemented throughout the project. The components were separated by the operators and split into lower level components to be implemented.

- Commitment of the sprints was not an issue since a general time-map was planned before time of how long each sprint would take, I ended up getting ahead of schedule by a couple of weeks which provided a but of breathing room which helped with commitment with the lower amount of mental pressure applied. Daily time-management consisted of 3-5 hours amount of work being conducted.
- Throughout the project I was open about the progress and challenges faces with the project supervisor and providing information about things I was worried about during the project.
- I knew when to ask for help during the project for general problems faced and things I did not understand through friends and the project supervisor.
- When a sprint was started, I focused on that body of work until it was completed and tested until I felt it met my standard.
- I respected the information and help given during the weekly project supervisor meetings which provided feedback on my progress and future work to be completed.

Testing

Testing through the project was a challenge as noticing problems came through noticing things that did not fit with an expected outcome from visually viewing the simulation

running and debugging the noticed potential problem checking the results of the Evolutionary algorithm in detail.

Following the Scrum methodology after a operator was implemented they were tested checking that they fit the required functionality but there were time during the data collection that problems were found with various operators which already had data collected for it, this made the data to be unreliable requiring repeat data collection to be conducted which took over 8 hours of data collection to be done again. This type of problem occurred 3 times during data collection which was cause by inadequate testing the output of some of the operators of the Evolutionary algorithm.

Unit testing was done to a certain extent through the project for some of the operators, but this could have been taken further for each of the Operators within the Evolutionary algorithm. The problems encountered during data collection could have been avoided by implementing the Unit testing further better validating the output of different combinations of the Evolutionary algorithm. This was due to lack of foresight on my part in terms of what to expect for the varying operators.

Quite a few of these problems would not have been noticed without noticing irregularities in the simulation running and watching the visual development of the cars on the web browser instead of just looking at the console data.

Further Work

To better provide an analysis of the varying operators more data would be acquired to better back-up pre-existing data from 10 run simulation for one combination of an Evolutionary algorithm to 30. An increase in the type of data to be gathered data such as the mutation rates for the cars data points and crossover changes between cars to create new cars. The environment the cars have raced in would also be recorded which would be used to compare how the car populations perform in the changing environments this would also make the data more accurate. The testing methodology would also be enhanced with expanding the number of unit tests better validating the output of the different operators.

The data collection method would also be changed to be more automatic which would not require micro-management every hour. This would be done by setting up a server to send the data to which would then re-format the data into graphs which can be used via python. Done on the severs side.

The number of operators used within the project would be expanded on taking from pre-existing research papers to be implemented. This would also expand the scope of testing and data collection increasing the time of processing simulation considerably.

Appendices

Appendix A

Figure 1

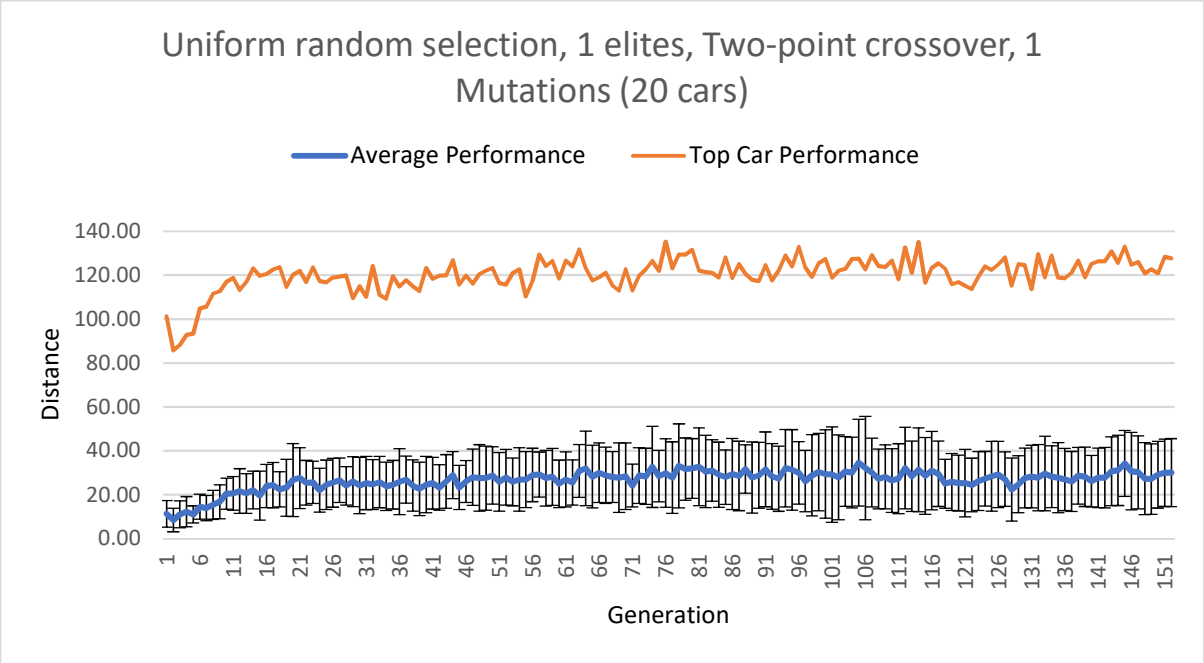


Figure 2

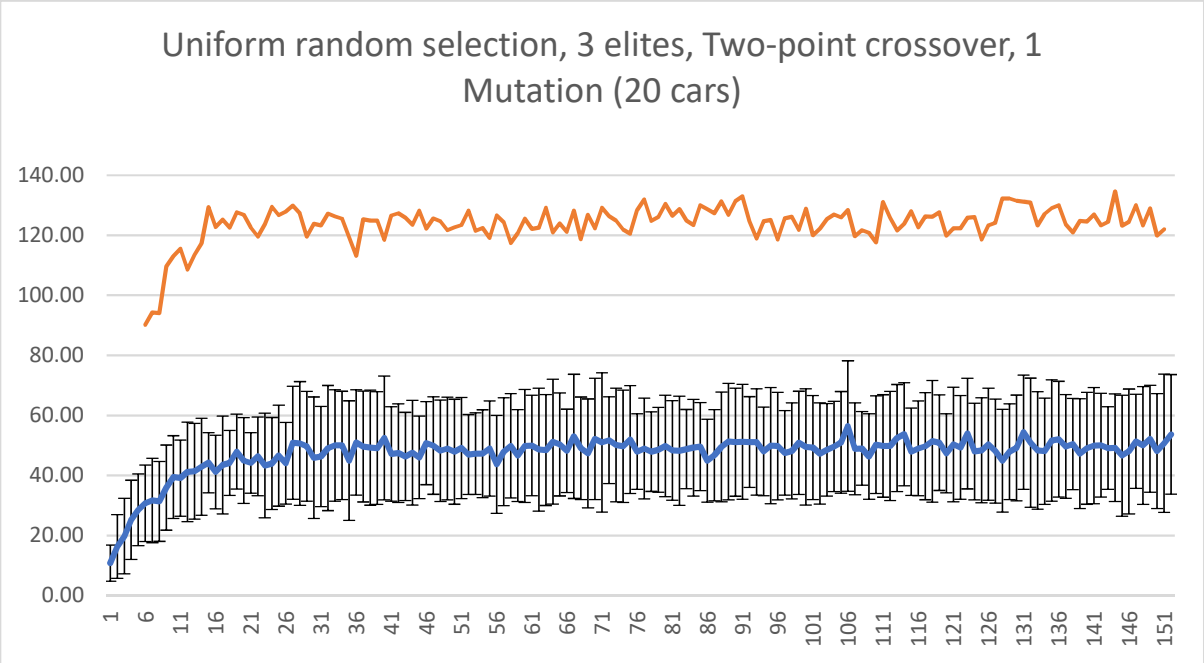
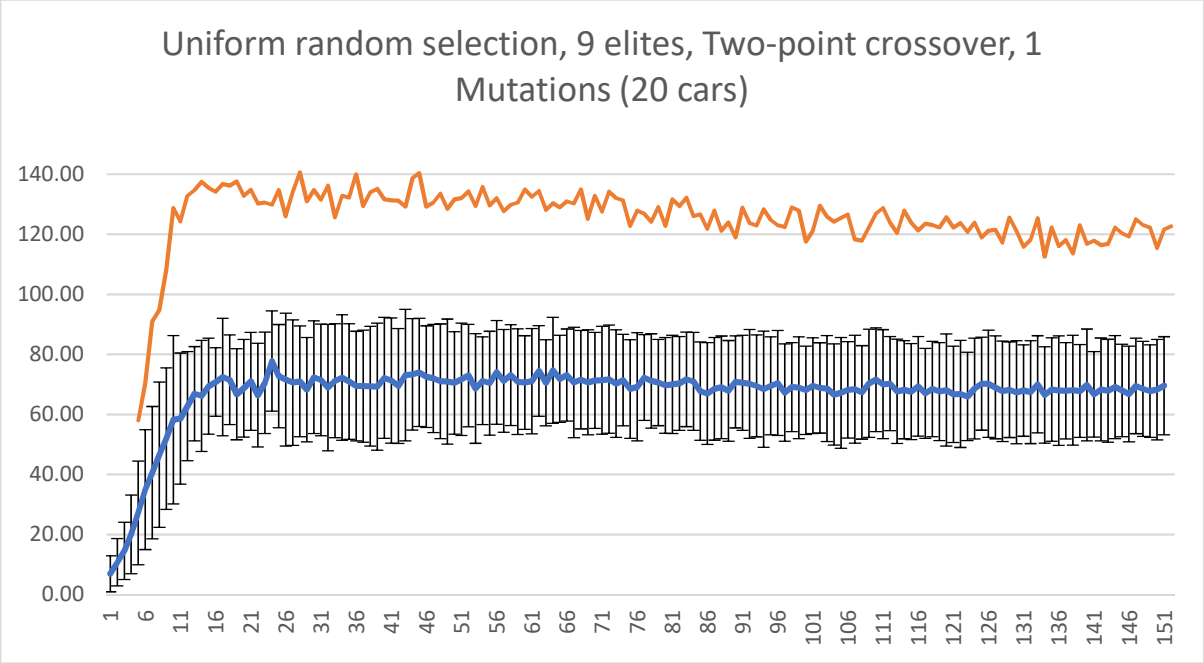


Figure 3


Uniform random selection, 9 elites, Two-point crossover, 1 Mutations (20 cars)
140.00
120.00
100.00
80.00
60.00
40.00
20.00
0.00
1
6
11
16
21
26
31
36
41
46
51
56
61
66
71
76
81
86
91
96
101
106
111
116
121
126
131
136
141
146
151

Appendix B

Figure 1

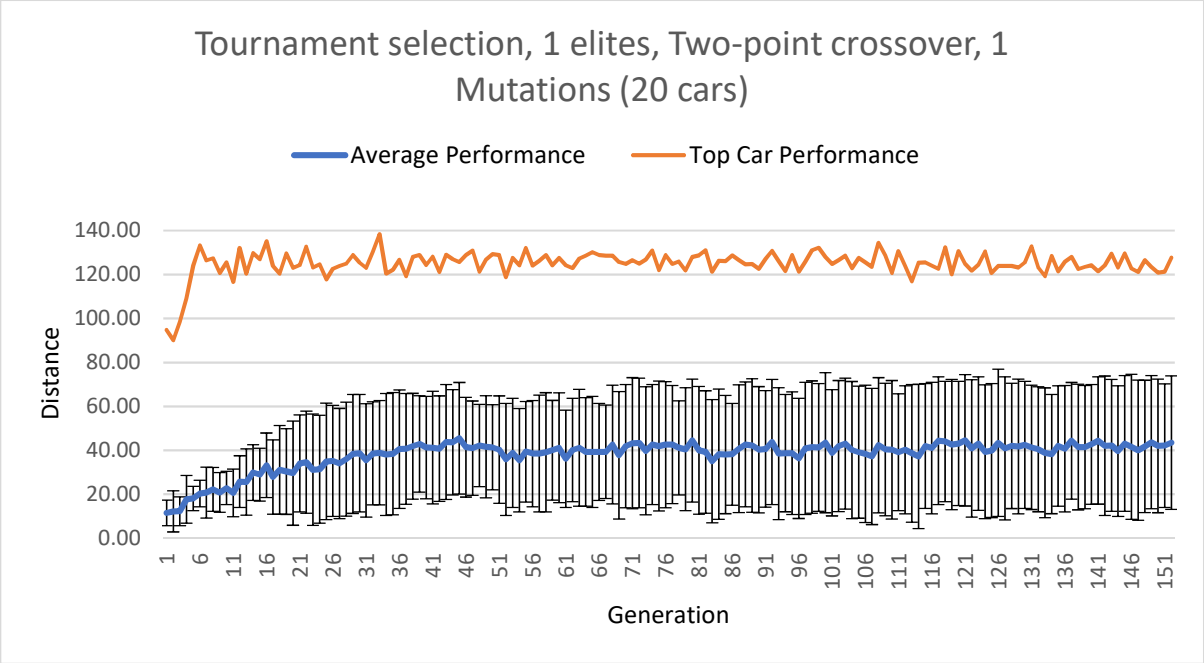


Figure 2

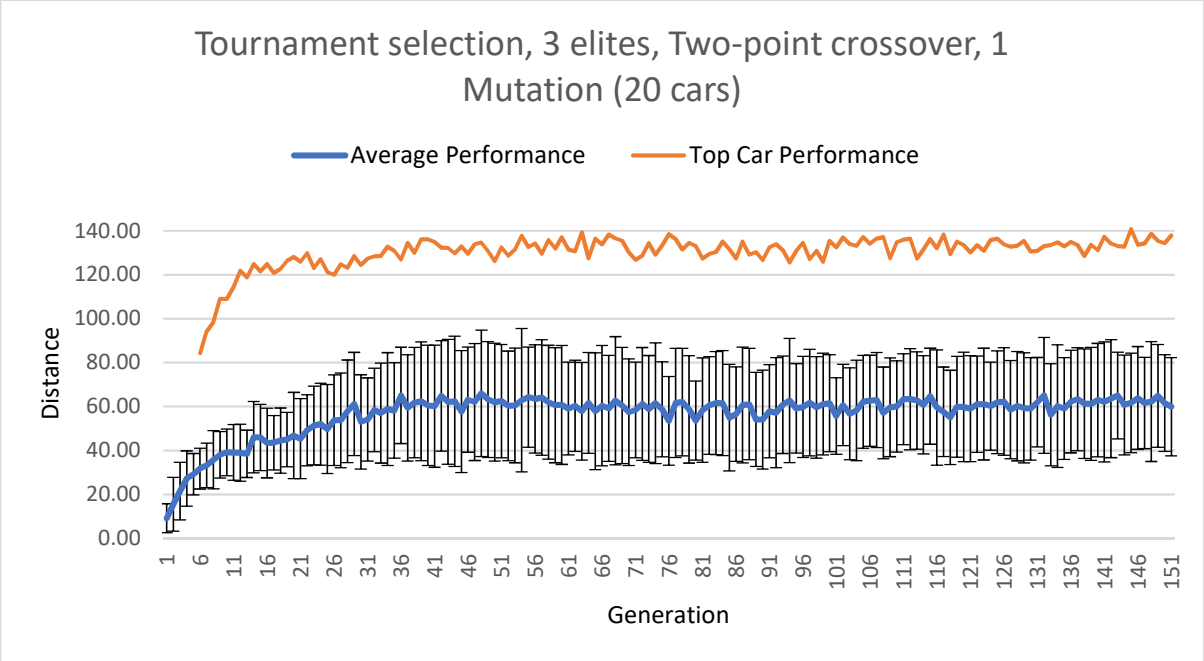
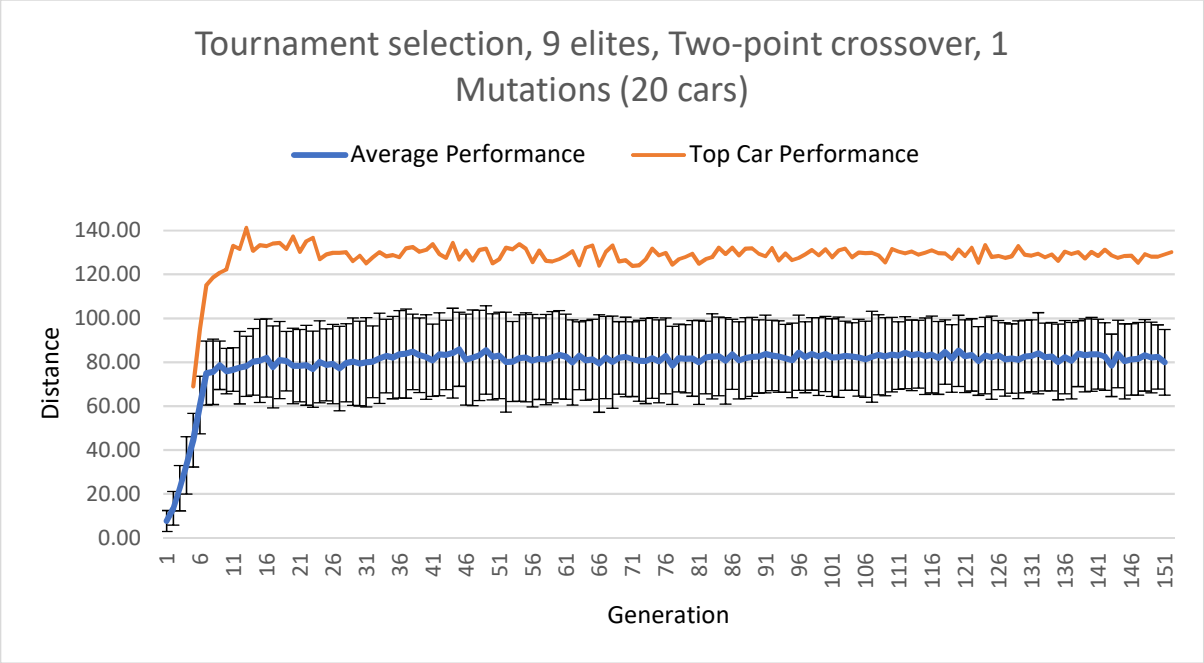


Figure 3


Tournament selection, 9 elites, Two-point crossover, 1 Mutations (20 cars)
Average Performance
Top Car Performance
Distance
140.00
120.00
100.00
80.00
60.00
40.00
20.00
0.00
1
6
11
16
21
26
31
36
41
46
51
56
61
66
71
76
81
86
91
96
101
106
111
116
121
126
131
136
141
146
151
Generation

Appendix C

Figure 1

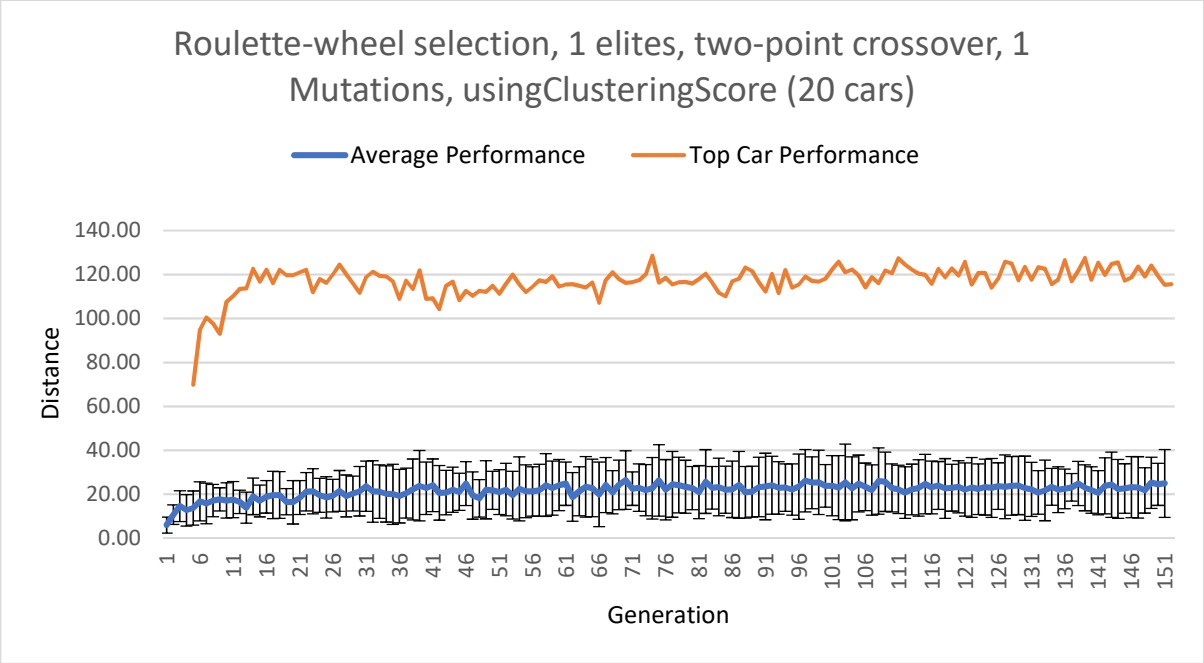


Figure 2

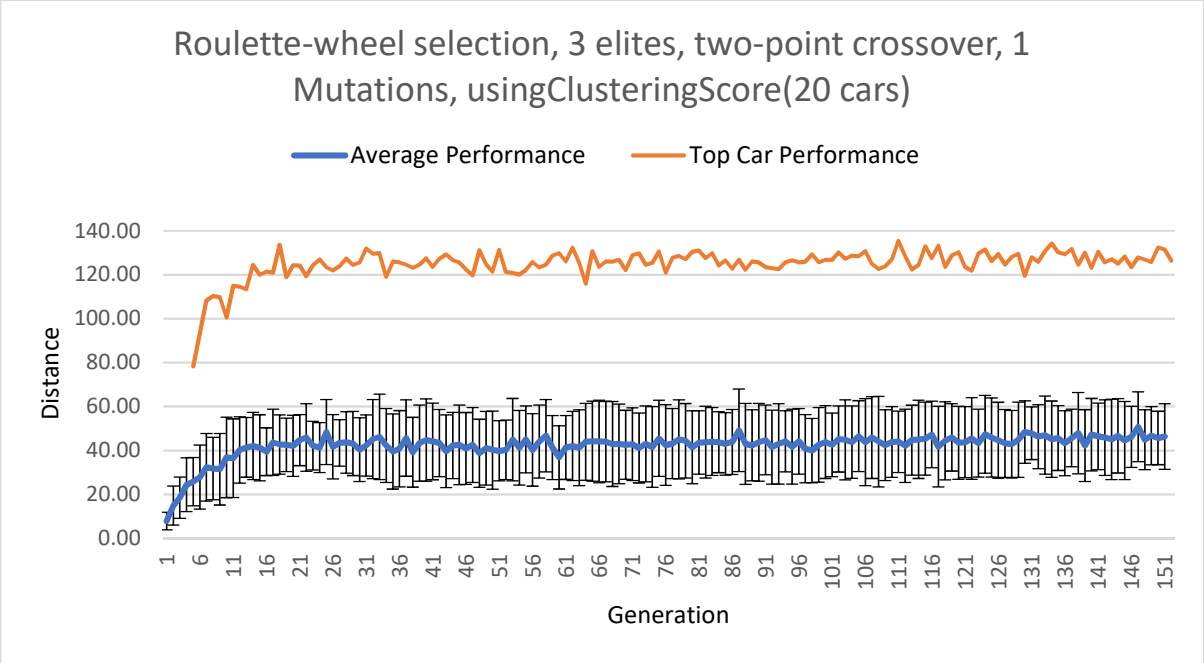
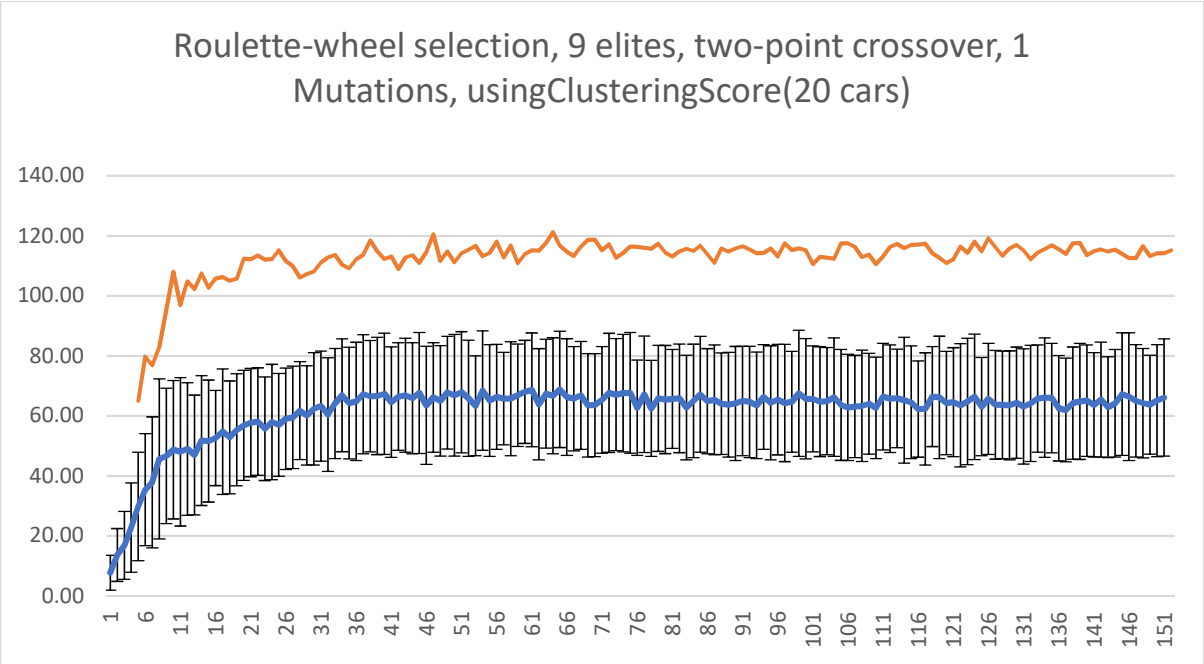


Figure 3


Roulette-wheel selection, 9 elites, two-point crossover, 1 Mutations, usingClusteringScore(20 cars)
140.00
120.00
100.00
80.00
60.00
40.00
20.00
0.00
1
6
11
16
21
26
31
36
41
46
51
56
61
66
71
76
81
86
91
96
101
106
111
116
121
126
131
136
141
146
151

Appendix D

Figure 1

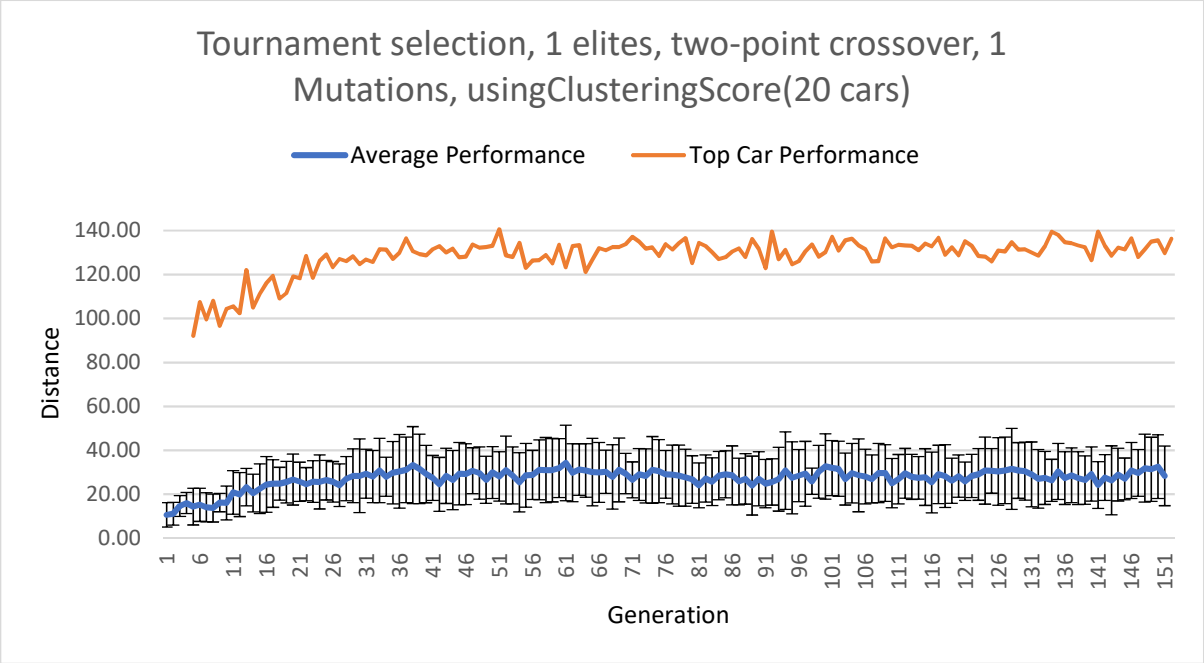


Figure 2

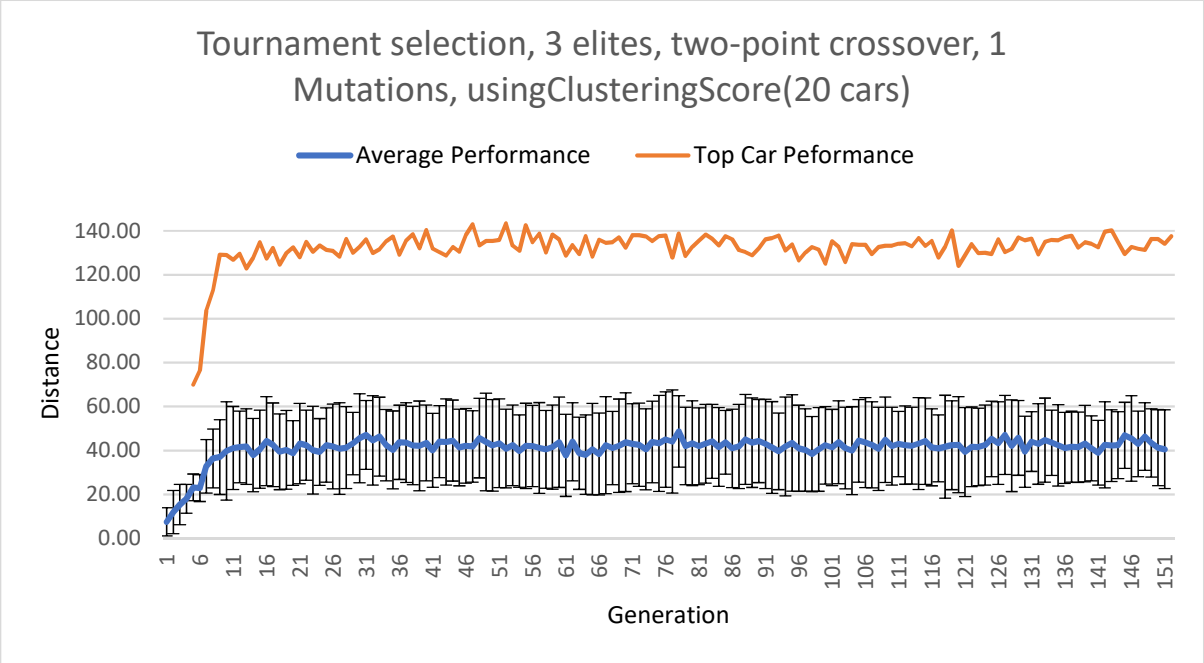
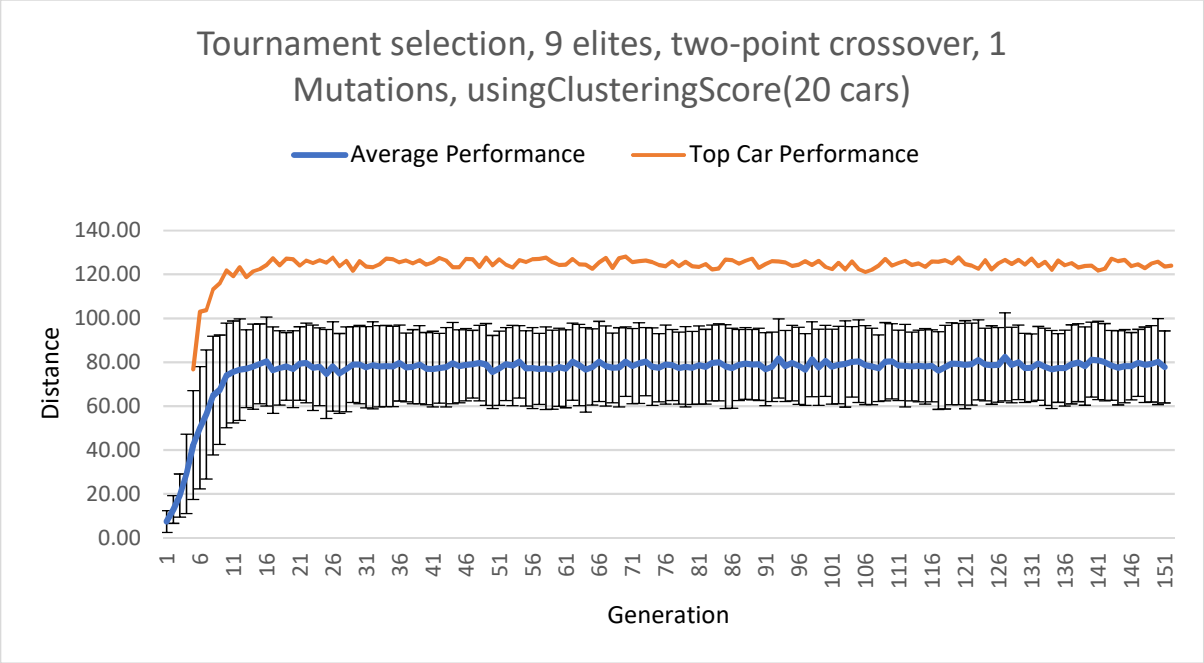


Figure 3


Tournament selection, 9 elites, two-point crossover, 1 Mutations, usingClusteringScore(20 cars)
Average Performance
Top Car Performance
Distance
140.00
120.00
100.00
80.00
60.00
40.00
20.00
0.00
1
6
11
16
21
26
31
36
41
46
51
56
61
66
71
76
81
86
91
96
101
106
111
116
121
126
131
136
141
146
151
Generation

Appendix E

Figure 1

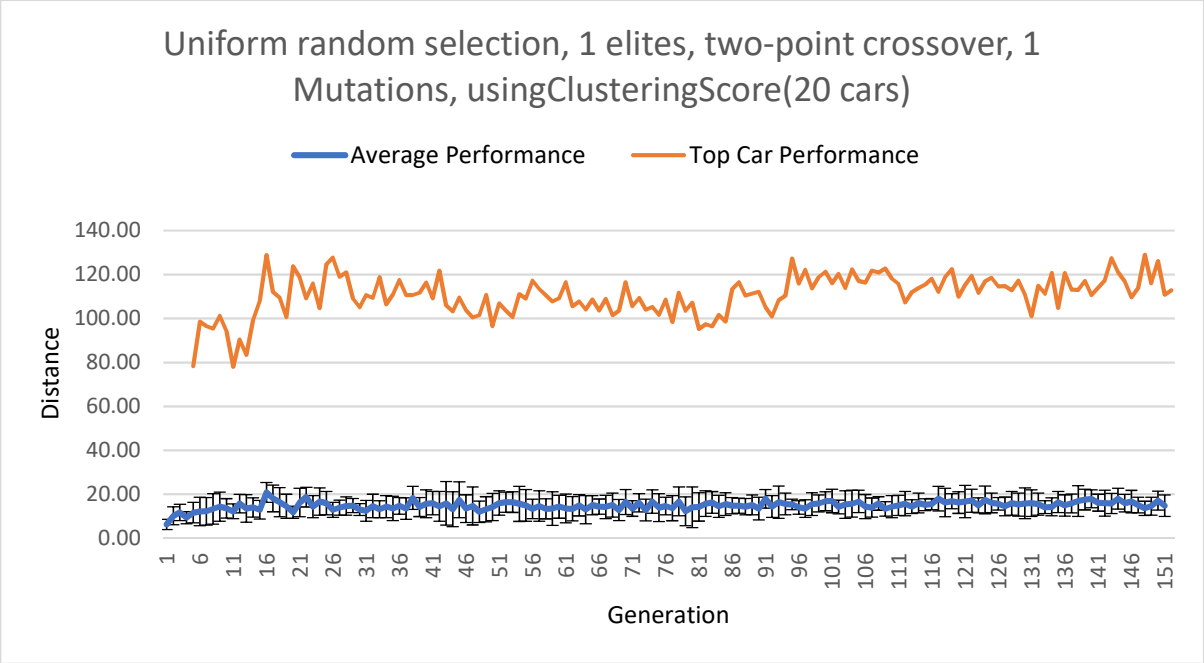


Figure 2

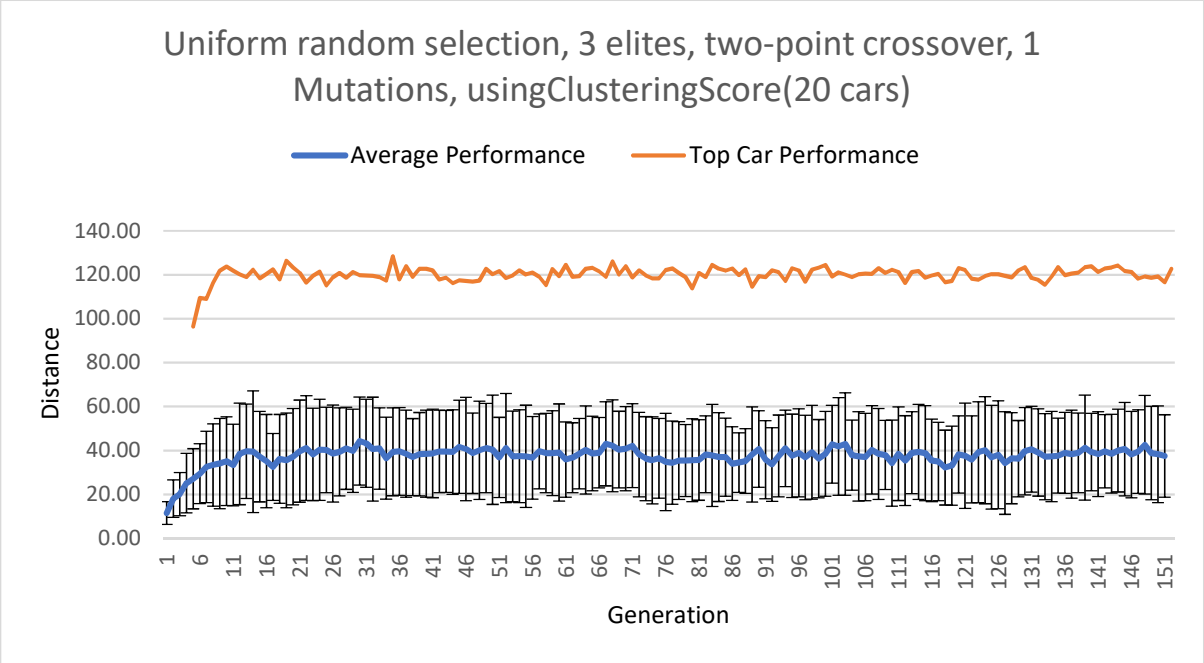
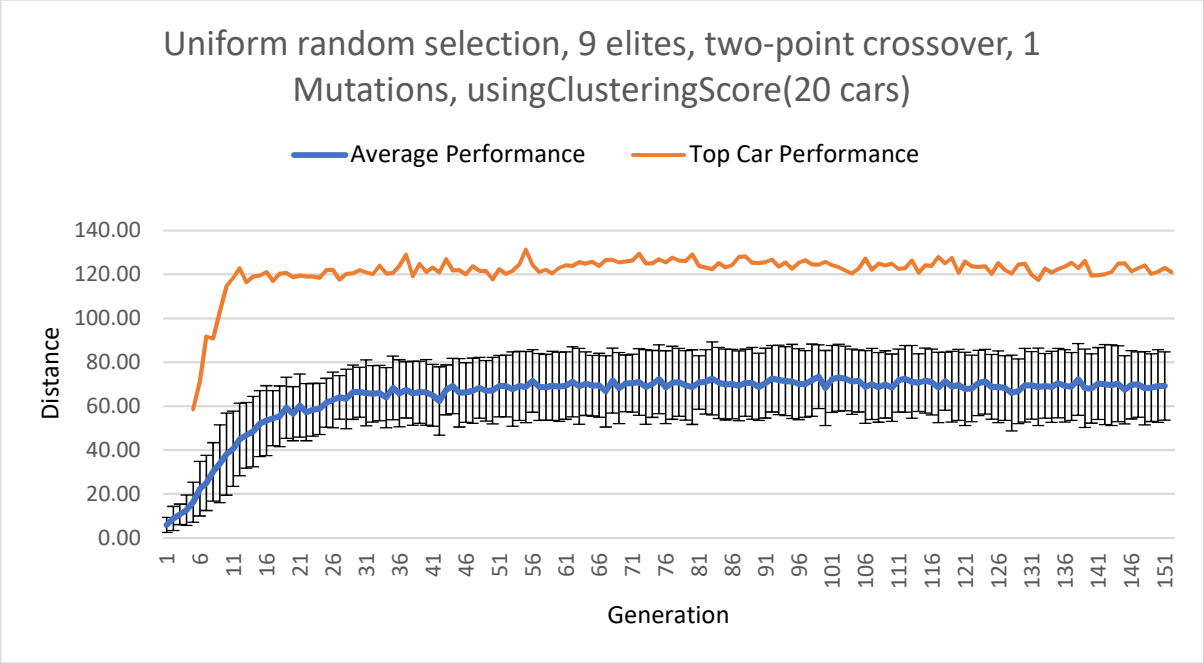


Figure 3


Uniform random selection, 9 elites, two-point crossover, 1 Mutations, usingClusteringScore(20 cars)
Average Performance
Top Car Performance
Distance
140.00
120.00
100.00
80.00
60.00
40.00
20.00
0.00
1
6
11
16
21
26
31
36
41
46
51
56
61
66
71
76
81
86
91
96
101
106
111
116
121
126
131
136
141
146
151
Generation

Appendix F

Figure 1

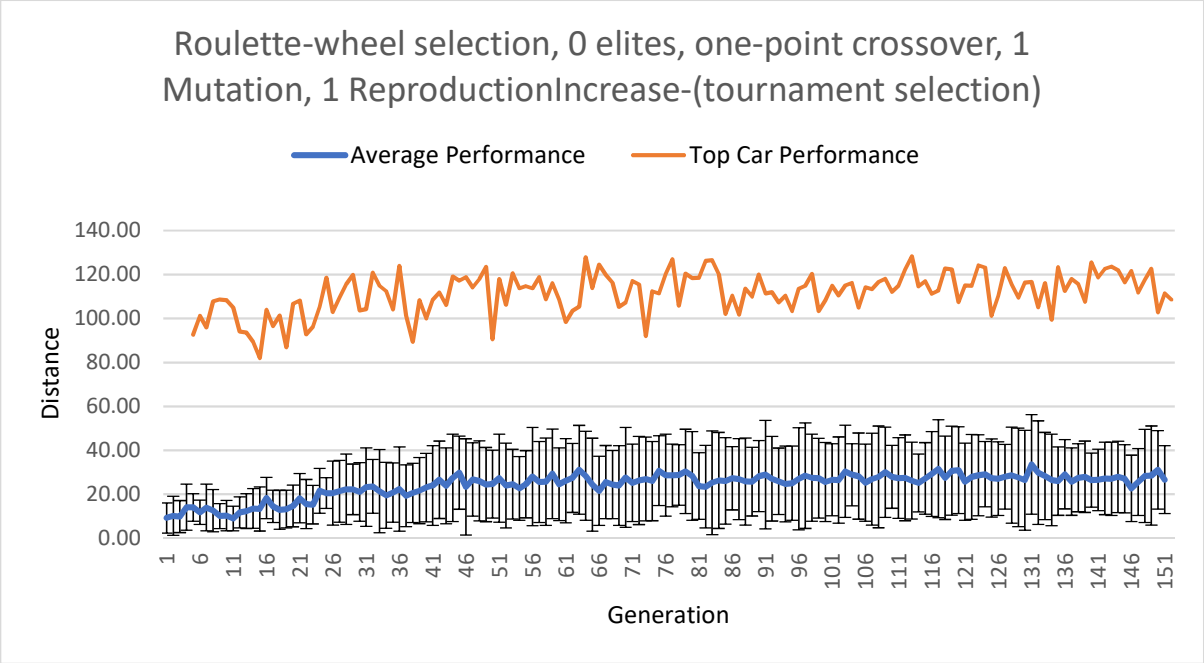


Figure 2

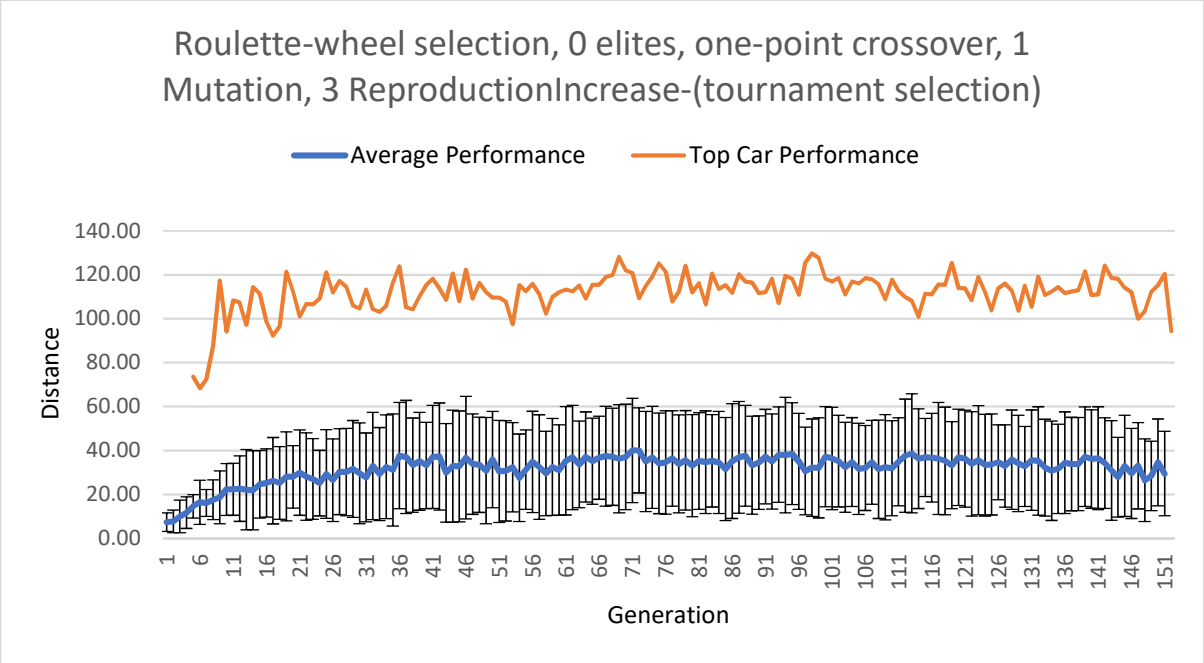
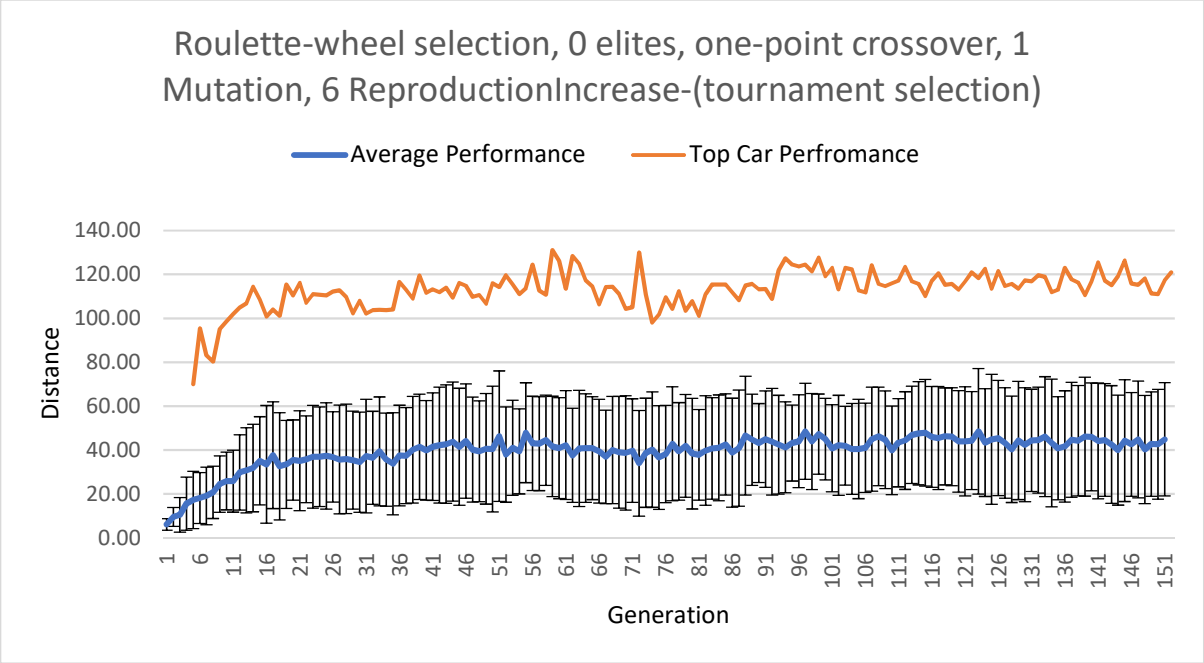


Figure 3


Roulette-wheel selection, 0 elites, one-point crossover, 1 Mutation, 6 ReproductionIncrease-(tournament selection)
Average Performance
Top Car Perfromance
Distance
140.00
120.00
100.00
80.00
60.00
40.00
20.00
0.00
1
6
11
16
21
26
31
36
41
46
51
56
61
66
71
76
81
86
91
96
101
106
111
116
121
126
131
136
141
146
151
Generation

Appendix G

Figure 1

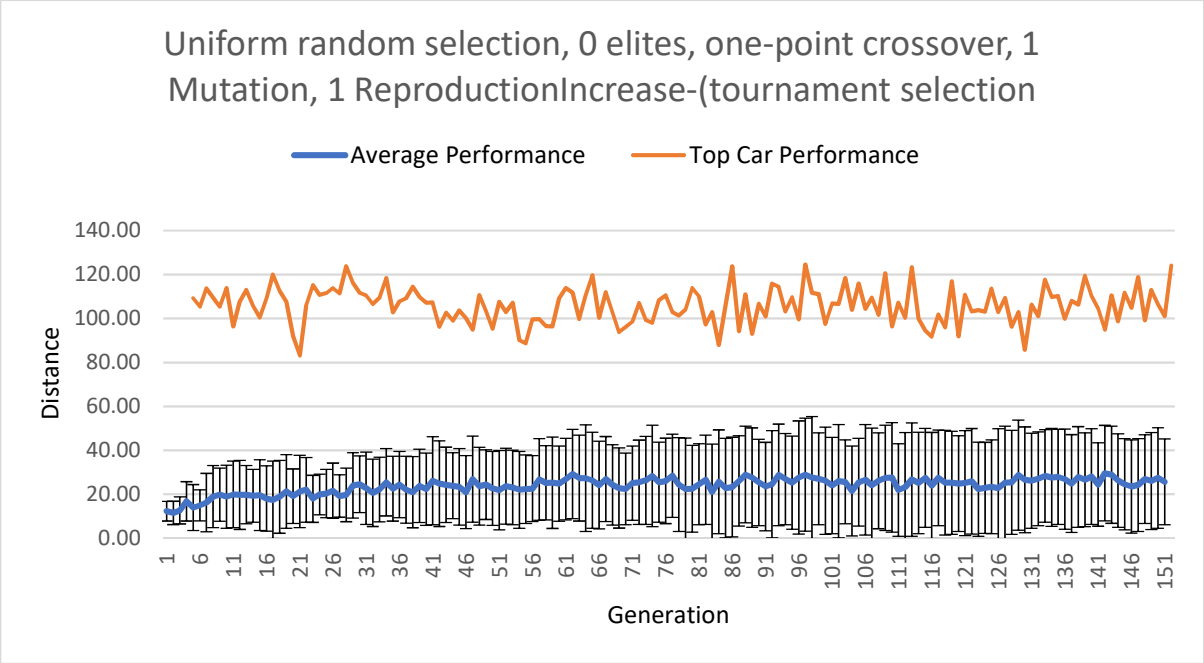


Figure 2

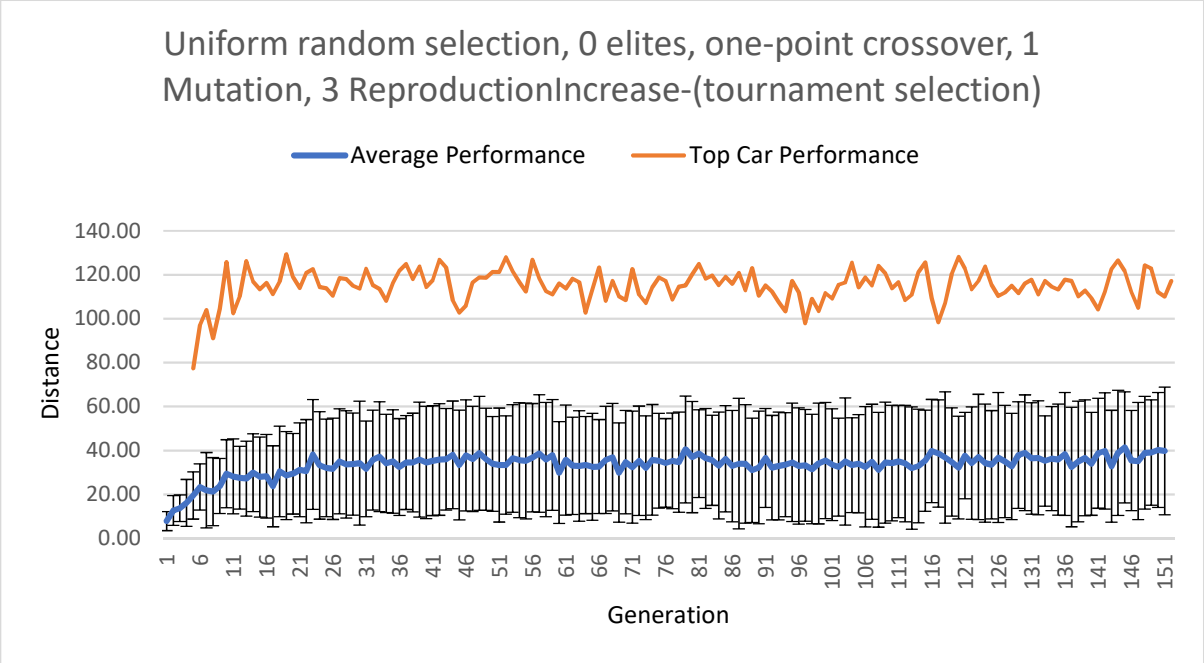
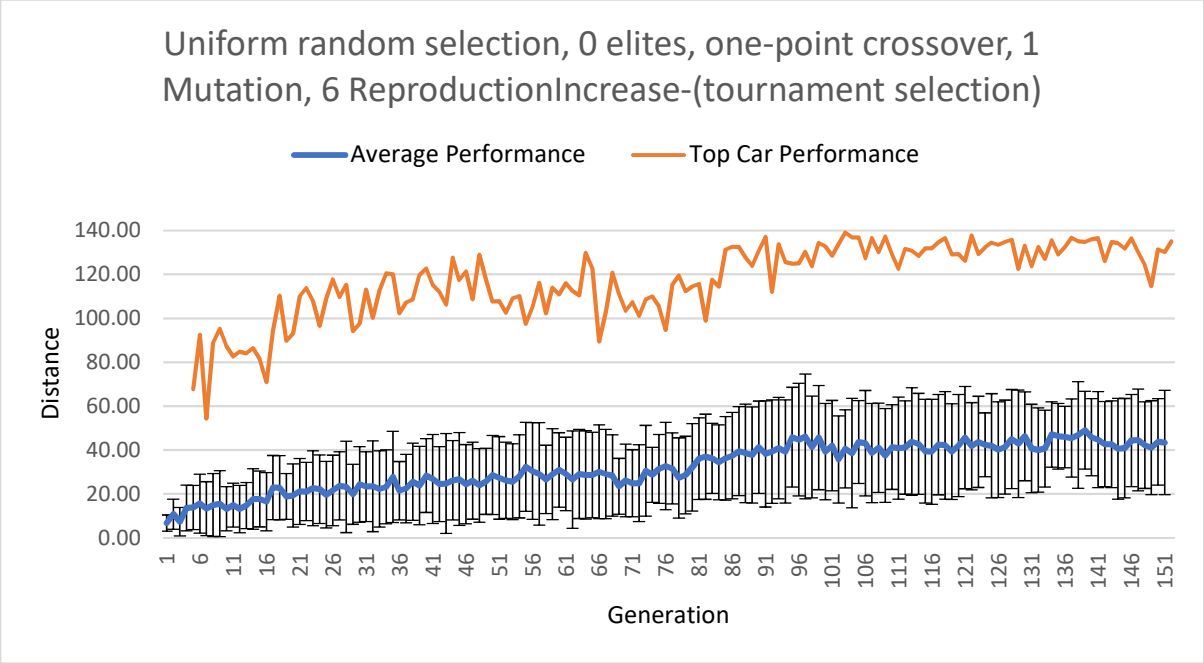


Figure 3


Uniform random selection, 0 elites, one-point crossover, 1 Mutation, 6 ReproductionIncrease-(tournament selection)
Average Performance
Top Car Performance
Distance
140.00
120.00
100.00
80.00
60.00
40.00
20.00
0.00
1
6
11
16
21
26
31
36
41
46
51
56
61
66
71
76
81
86
91
96
101
106
111
116
121
126
131
136
141
146
151
Generation

Appendix H

Figure 1

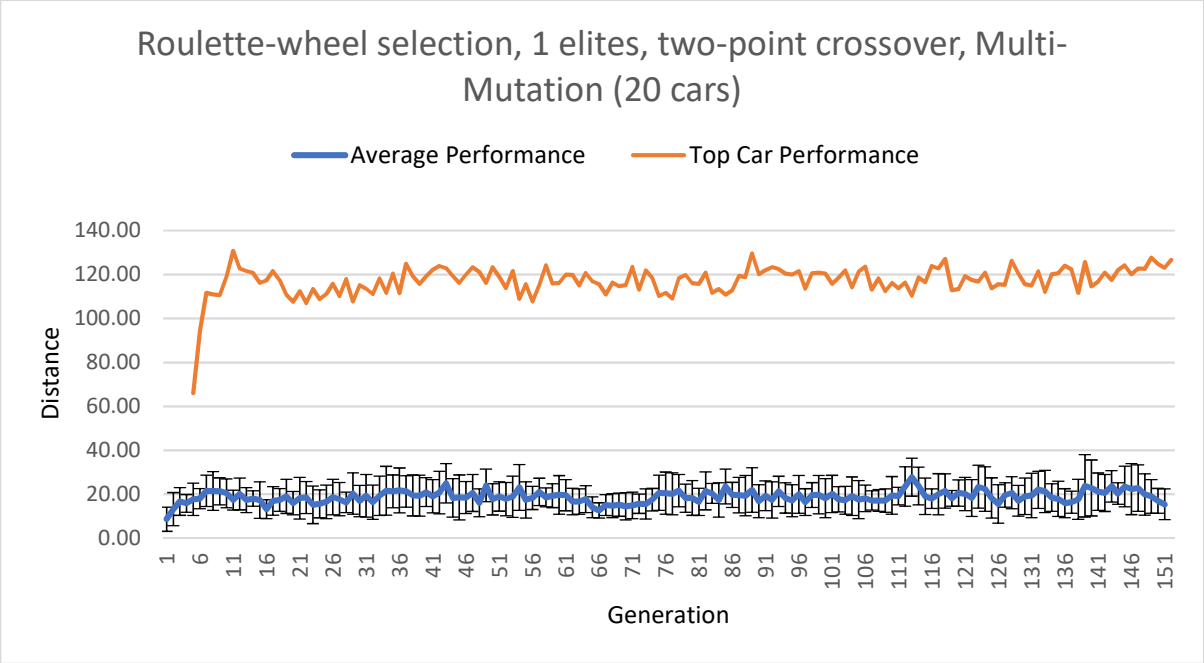


Figure 2

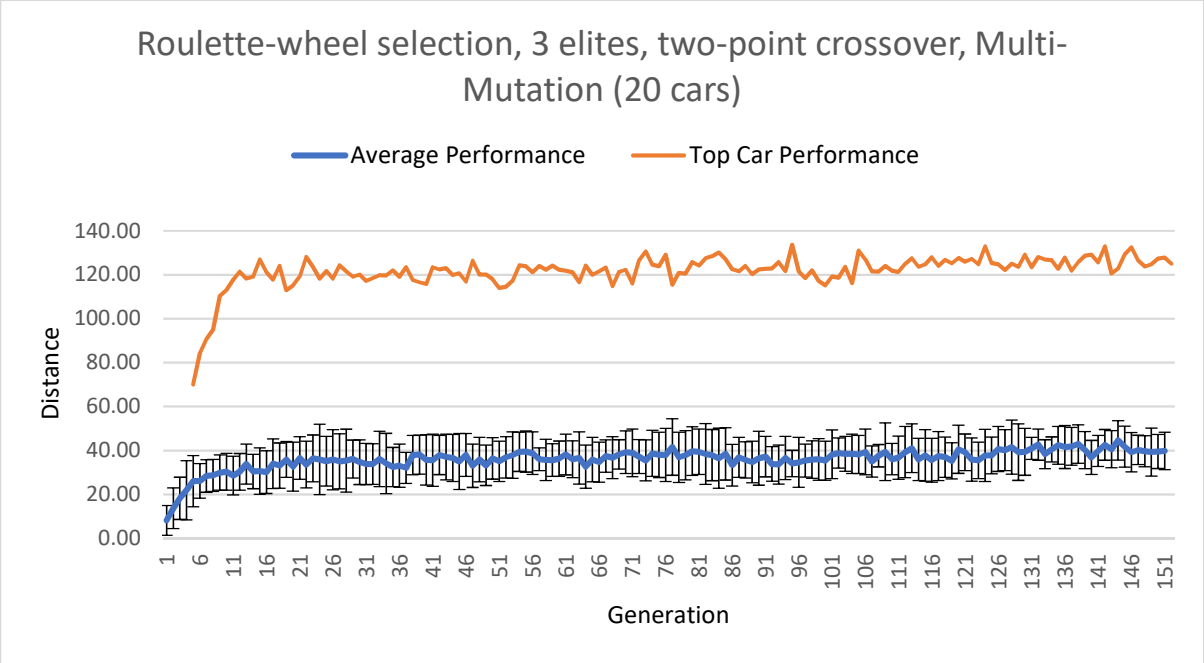
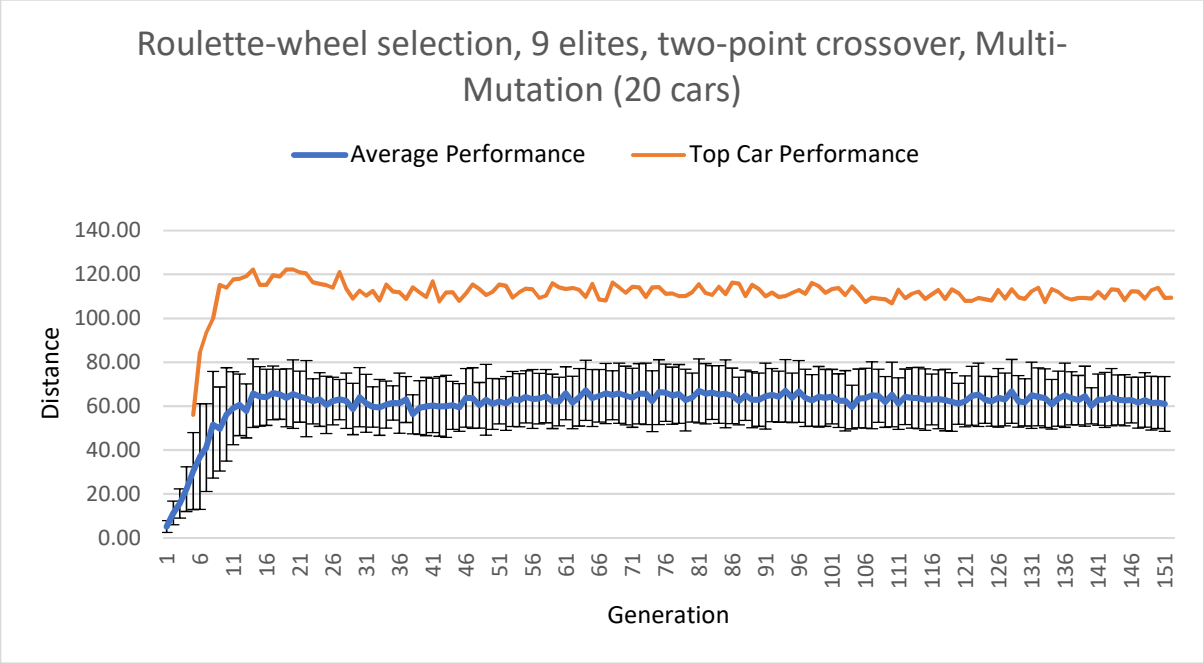


Figure 3


Roulette-wheel selection, 9 elites, two-point crossover, Multi-Mutation (20 cars)
Average Performance
Top Car Performance
Distance
140.00
120.00
100.00
80.00
60.00
40.00
20.00
0.00
1
6
11
16
21
26
31
36
41
46
51
56
61
66
71
76
81
86
91
96
101
106
111
116
121
126
131
136
141
146
151
Generation

Appendix I

Figure 1

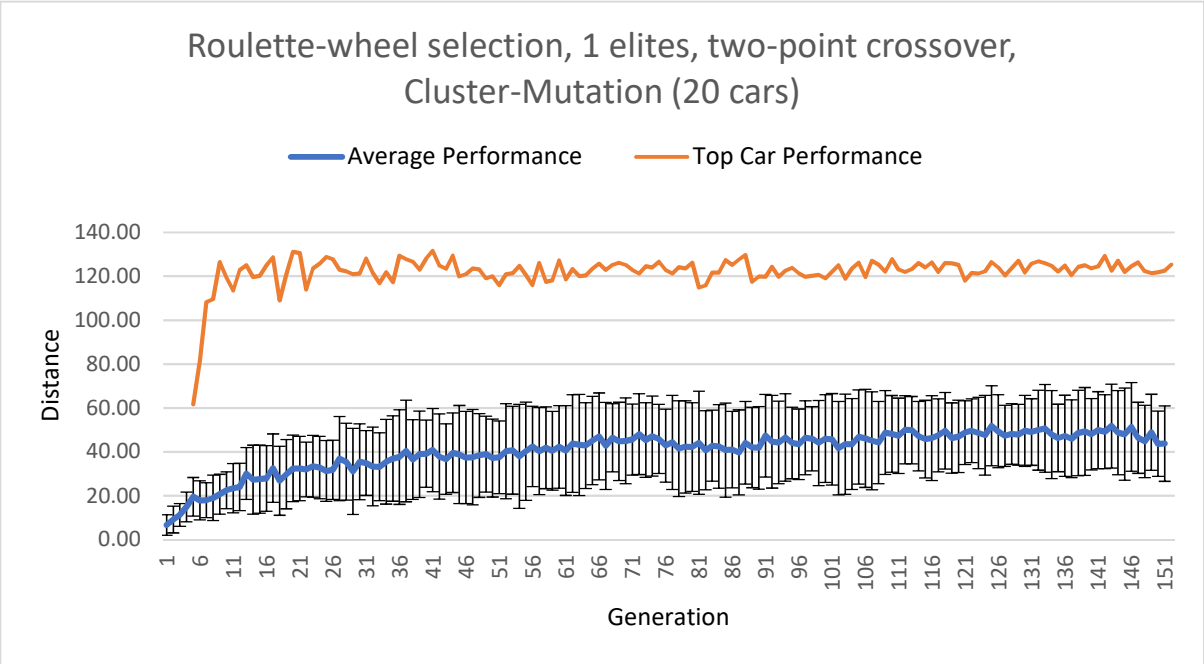


Figure 2

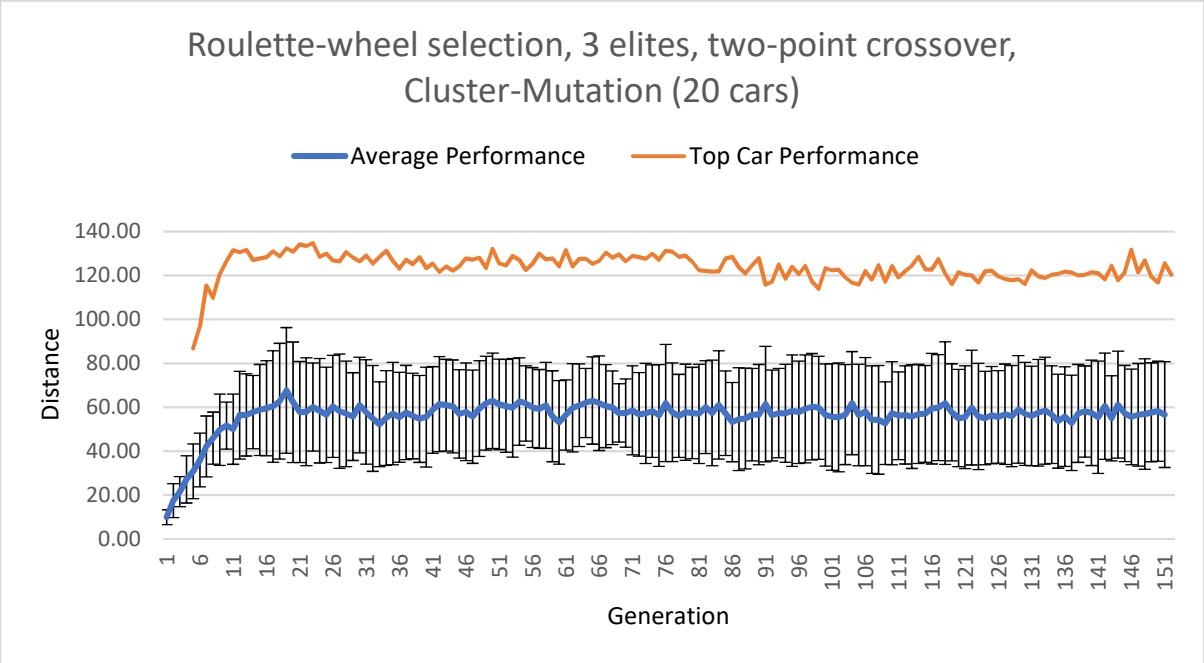
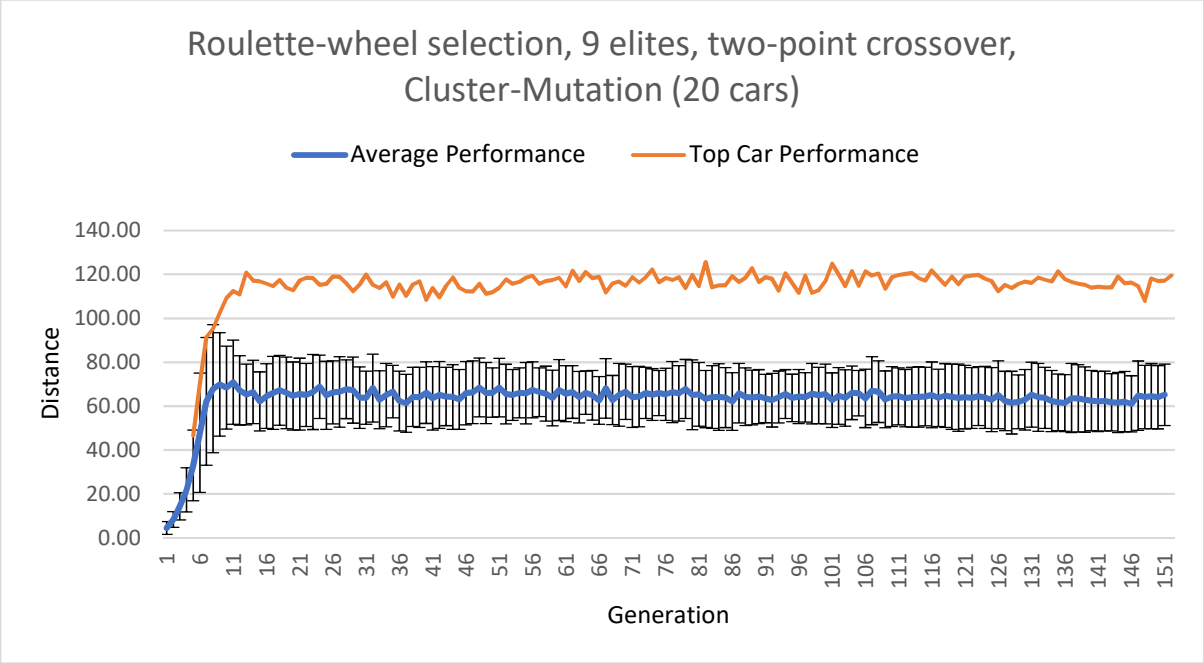


Figure 3


Roulette-wheel selection, 9 elites, two-point crossover, Cluster-Mutation (20 cars)
Average Performance
Top Car Performance
Distance
140.00
120.00
100.00
80.00
60.00
40.00
20.00
0.00
1
6
11
16
21
26
31
36
41
46
51
56
61
66
71
76
81
86
91
96
101
106
111
116
121
126
131
136
141
146
151
Generation

Appendix J

Figure 1

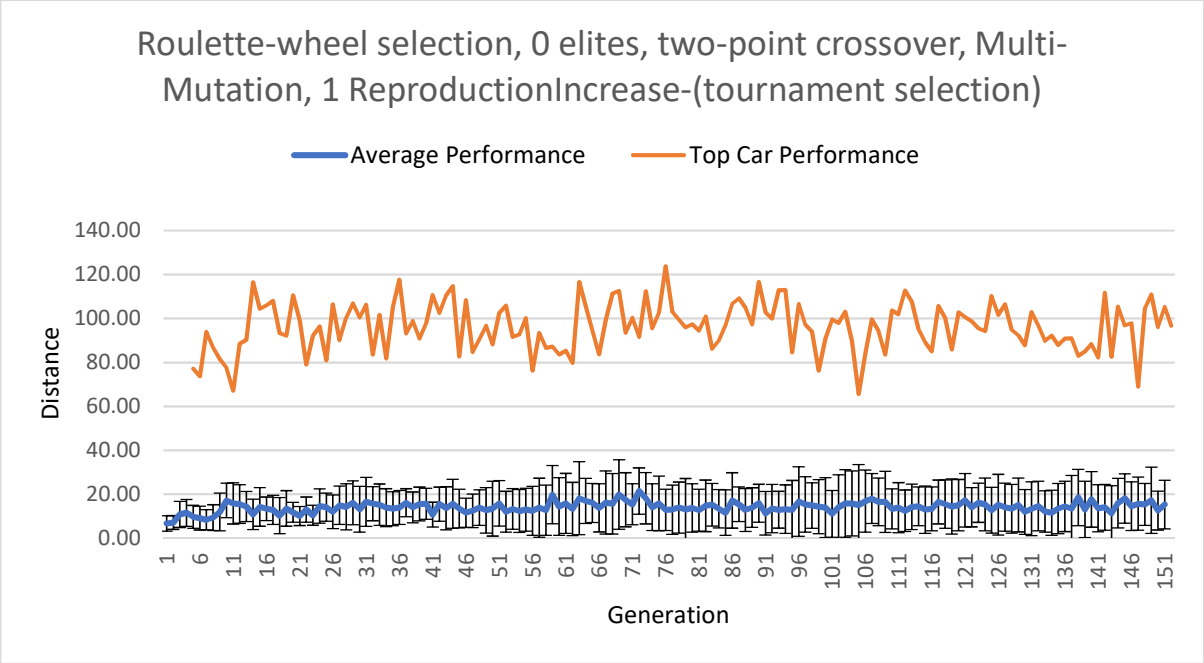


Figure 2

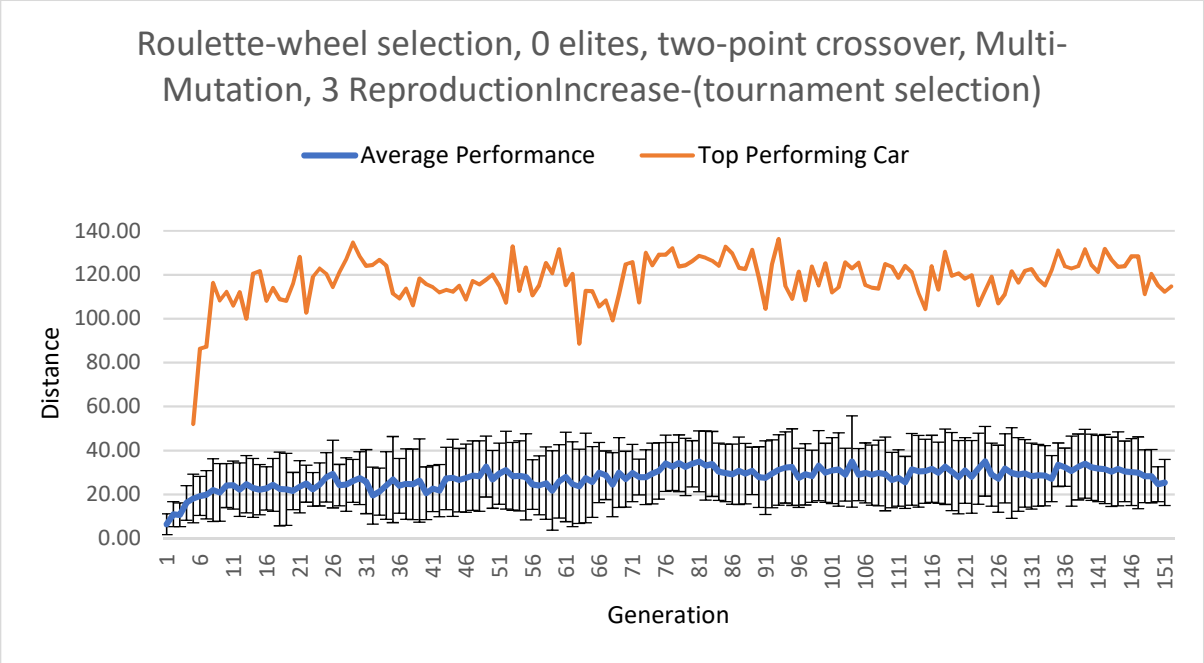
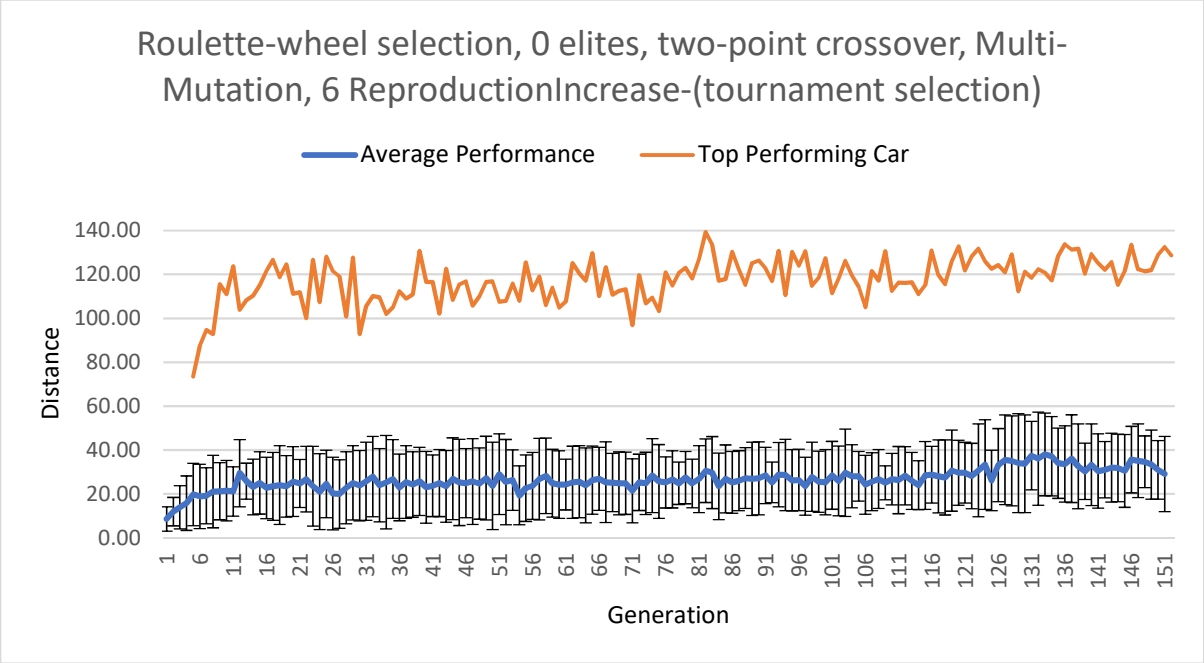


Figure 3


Roulette-wheel selection, 0 elites, two-point crossover, Multi-Mutation, 6 ReproductionIncrease-(tournament selection)
Average Performance
Top Performing Car
Distance
140.00
120.00
100.00
80.00
60.00
40.00
20.00
0.00
1
6
11
16
21
26
31
36
41
46
51
56
61
66
71
76
81
86
91
96
101
106
111
116
121
126
131
136
141
146
151
Generation

Appendix K

Figure 1

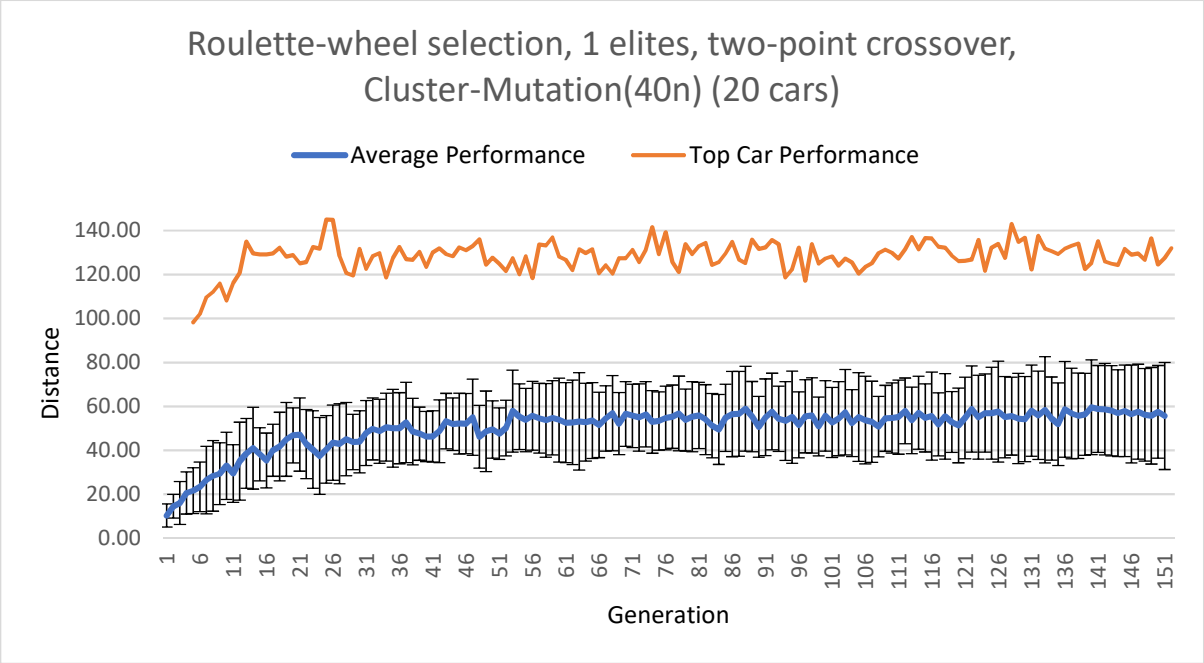


Figure 2

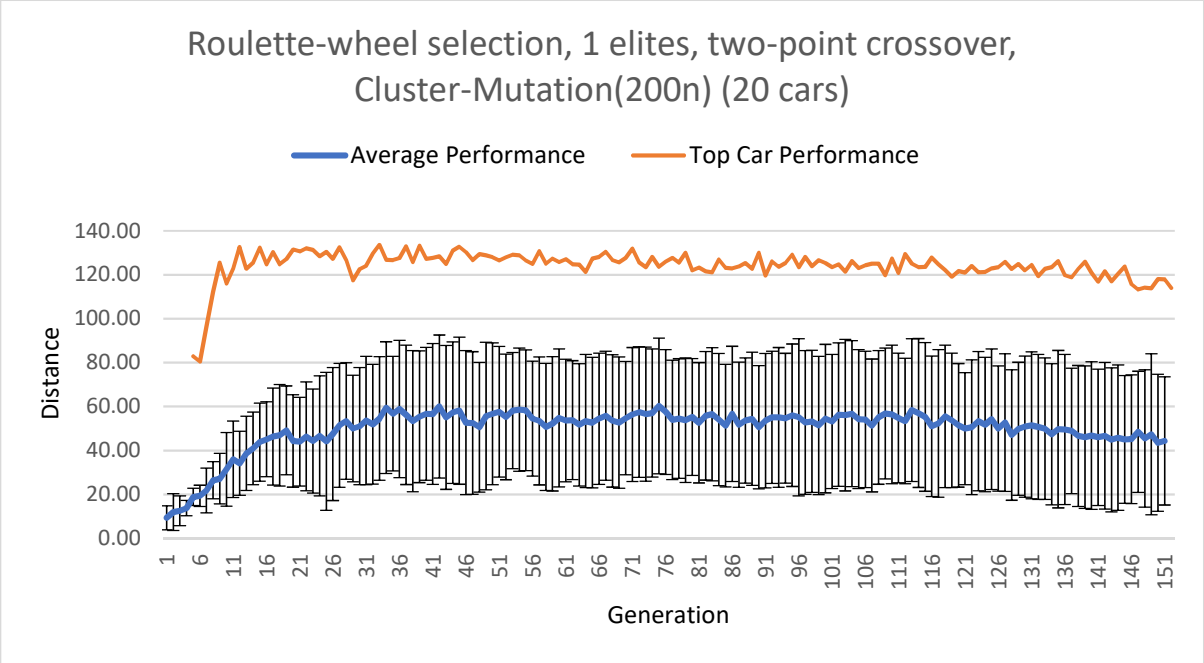
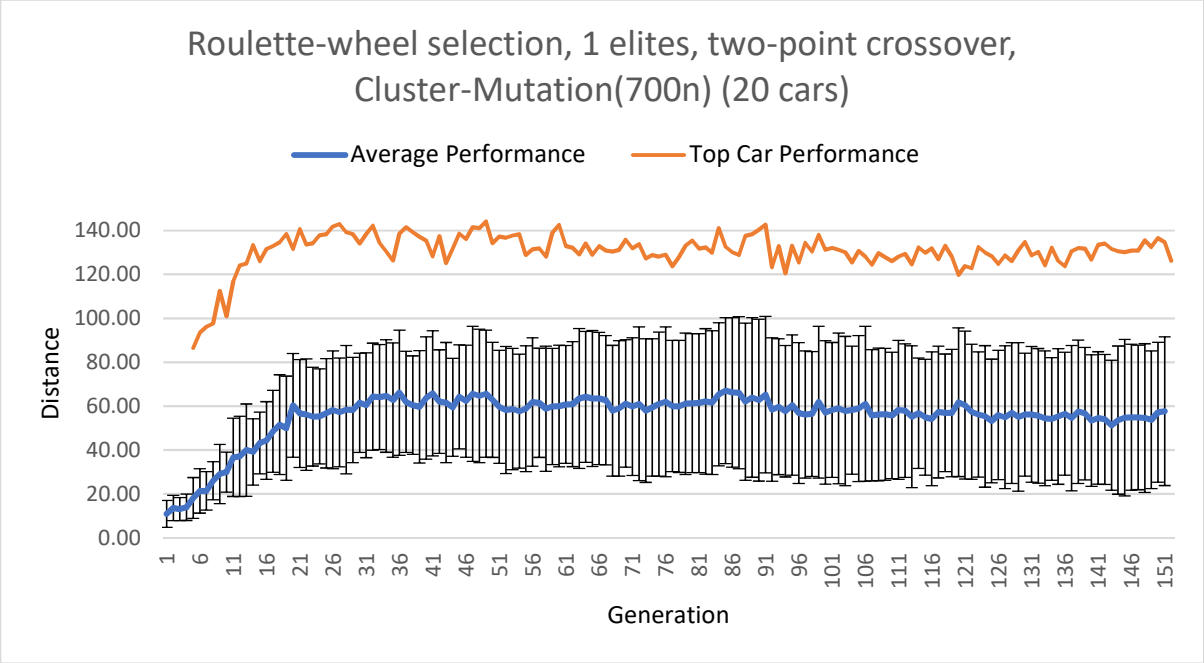


Figure 3


Roulette-wheel selection, 1 elites, two-point crossover, Cluster-Mutation(700n) (20 cars)
Average Performance
Top Car Performance
Distance
140.00
120.00
100.00
80.00
60.00
40.00
20.00
0.00
1
6
11
16
21
26
31
36
41
46
51
56
61
66
71
76
81
86
91
96
101
106
111
116
121
126
131
136
141
146
151
Generation

Appendix L

Figure 1

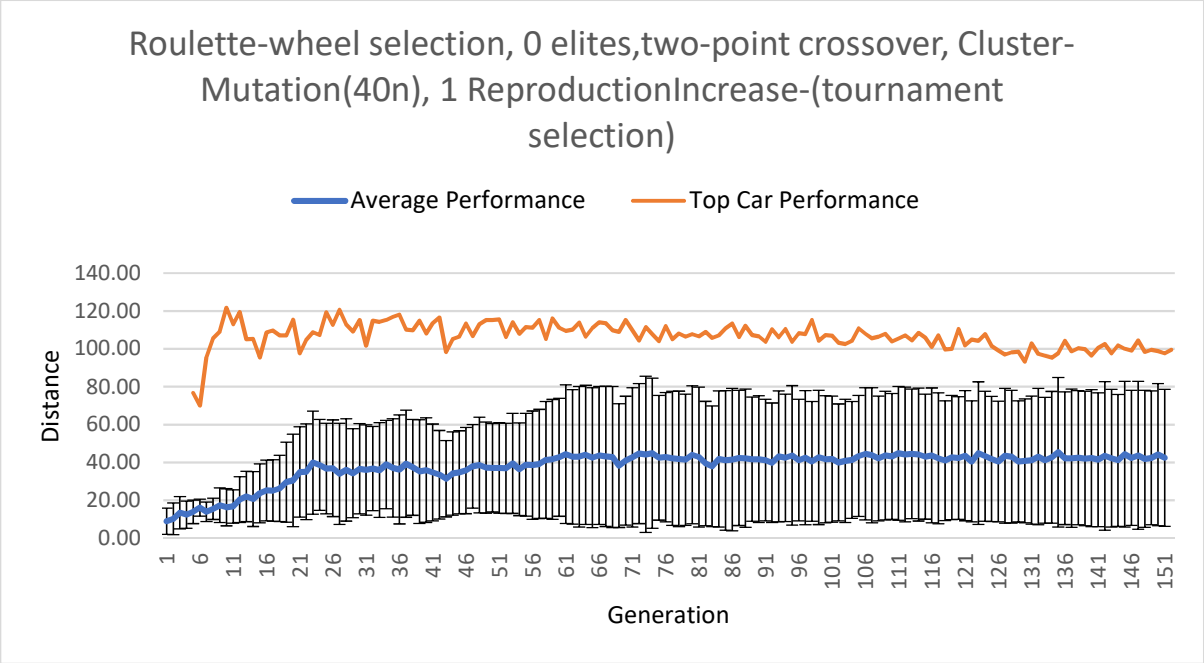


Figure 2

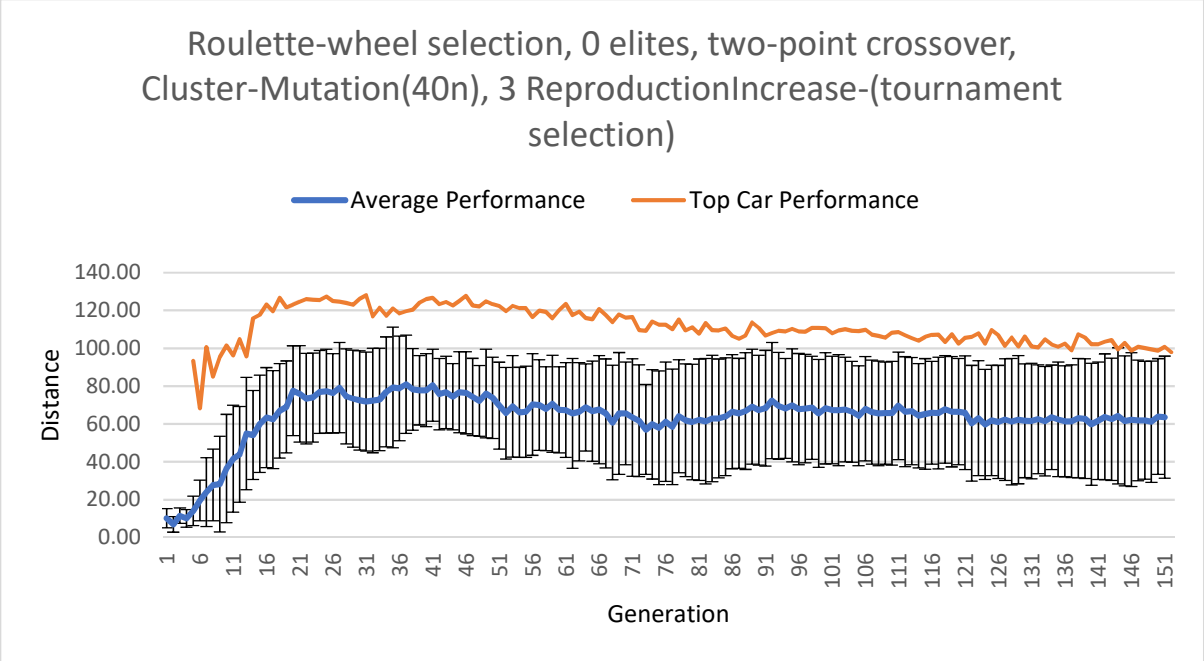
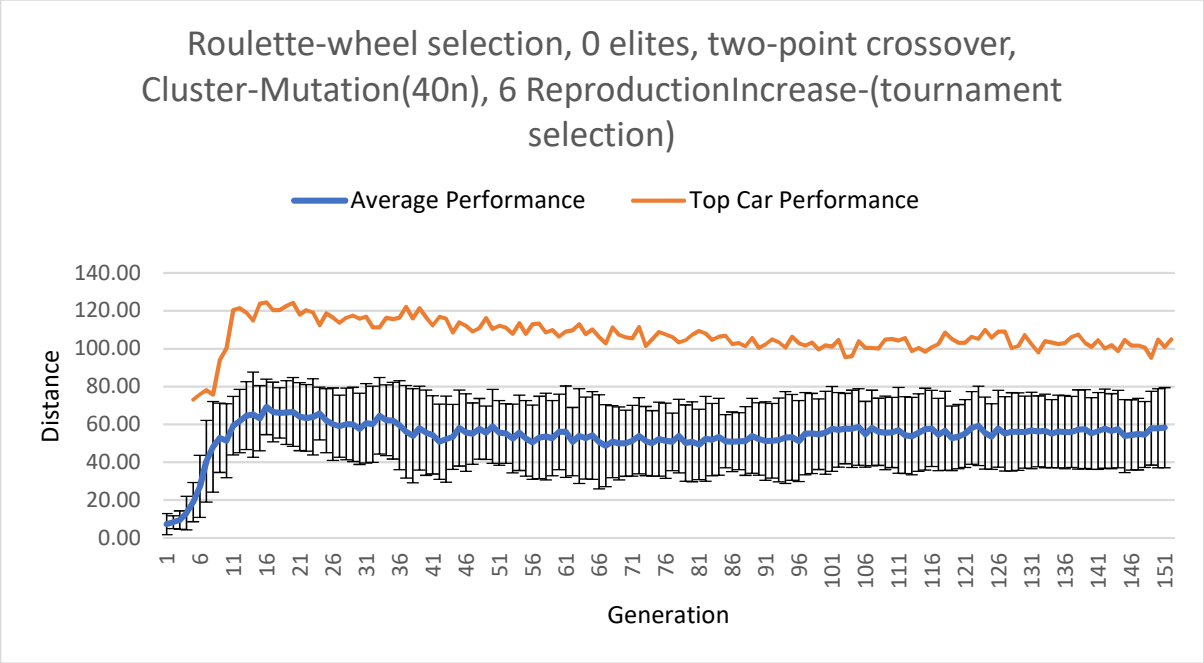


Figure 3


Roulette-wheel selection, 0 elites, two-point crossover, Cluster-Mutation(40n), 6 ReproductionIncrease-(tournament selection)
Average Performance
Top Car Performance
Distance
140.00
120.00
100.00
80.00
60.00
40.00
20.00
0.00
1
6
11
16
21
26
31
36
41
46
51
56
61
66
71
76
81
86
91
96
101
106
111
116
121
126
131
136
141
146
151
Generation

Appendix M – Ethics Submission

Ethics submission reference number: **12251**

Institute

IMPACS

Please provide a brief summary of your project (150 word max)

I'm creating an Evolutionary Algorithm for a Javascript framework which is a 2d car racing simulation where the car's evolve overtime improving their performance. I'm creating an algorithm which makes the cars evolve over time.

Where appropriate, do you have consent for the publication, reproduction or use of any unpublished material?

Not applicable

Will appropriate measures be put in place for the secure and confidential storage of data? Yes

Does the research pose more than minimal and predictable risk to the researcher? Not applicable

Will you be travelling, as a foreign national, in to any areas that the UK Foreign and Commonwealth Office advise against travel to?

No

Please include any further relevant information for this section here:

If you are to be working alone with vulnerable people or children, you may need a DBS (CRB) check. Tick to confirm that you will ensure you comply with this requirement should you identify that you require one.

Yes

Declaration: Please tick to confirm that you have completed this form to the best of your knowledge and that you will inform your department should the proposal significantly change.

Yes

Please include any further relevant information for this section here:

Appendix N – Third Party Code

The genetic cars simulation can be found at https://github.com/red42/HTML5_Genetic_Cars under a zlib licence authored by Rafael Matsunaga with Modifications made to some part of the code.

The final solution whhich is the Evolutionay algorithm consist of the following fles:

\HTML5_Genetic_Cars-master\src\machine-learning\genetic-algorithm\crossover.js - all the code in this file is my own written code

\HTML5_Genetic_Cars-master\src\machine-learning\genetic-algorithm\mutation.js - all the code in this file is my own written code

\HTML5_Genetic_Cars-master\src\machine-learning\genetic-algorithm\randomInt.js - all the code in this file is my own written code

\HTML5_Genetic_Cars-master\src\machine-learning\genetic-algorithm\selecton.js - all the code in this file is my own written code

\HTML5_Genetic_Cars-master\src\machine-learning\genetic-algorithm\clustering\cluster.js - all the code in this file is my own written code

\HTML5_Genetic_Cars-master\src\machine-learning\genetic-algorithm\clustering\clusterSetup.js - all the code in this file is my own written code

\HTML5_Genetic_Cars-master\src\machine-learning\genetic-algorithm\manage-round.js - the code that is mine within this program is between line 37-200

\HTML5_Genetic_Cars-master\src\index.js - the code that is mine is line 4 and line 405-414

\genetic_cars-testing-framework\HTML5_Genetic_Cars-master\spec\crossover.spec.js - the crossover operator unit tests

\genetic_cars-testing-framework\HTML5_Genetic_Cars-master\spec\selection.spec.js - the selection operator unit tests

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] Pencheva, Tania & Atanassov, K & Shannon, A. (2009). Modelling of a Roulette Wheel Selection Operator in Genetic Algorithms Using Generalized Nets. International Journal Bioautomation. 13.

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.

[10] Hassanat, Ahmad & Alkafaween, Esra'A & Alnawaiseh, Nedal & Abbadi, Mohammad & Alkasassbeh, Mouhammd & Alhasanat, Mahmoud. (2016). Enhancing Genetic Algorithms using Multi Mutations: Experimental Results on the Travelling Salesman Problem. International Journal of Computer Science and Information Security. 14. 785-801.

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

[11] A. Chehouri, R. Younes, J. Khoder, J. Perron and A. Ilinca, "A Selection Process for Genetic Algorithm Using Clustering Analysis", Algorithms, vol. 10, no. 4, p. 123, 2017. Available: 10.3390/a10040123.

Using clustering analysis to apply a change to the Car scores to optimize the selection process. The assumption being made is that a global pattern of good performance can be applied to the individual datapoints/chromosomes of a car, applying a score or rank to said datapoint to be added to a new summed score for the car. For example, a Local generation car is ranked in the middle of the population, applying the new score finds the datapoints of the car is rank in the top set of the population instead due to past existing cars being ranked at the top with similar datapoints [11].