Evolutionary Design Optimization for a Formula One Car and Track

by

Daniel Harper

(Under the Direction of Khaled Rasheed)

Abstract

In the last few years, a regulation change, development restrictions, and budget caps came to Formula One. Now, teams have to be intentional with their development resources while competing for fractions of seconds. I propose a set of evolutionary algorithms to allow a team to find car development areas to become up to 18% faster in as little as 20 minutes. All evolutionary algorithms are able to outperform a comparison algorithm. I also show that evolutionary algorithms can have an impact throughout motorsports with track evolution, which creates circuits in only a couple minutes. In the same years that the teams have changed their spending and testing, we've seen introduction of new tracks and new formats like the Sprint Race. Using evolutionary track generation would allow teams to better understand their car's weaknesses, track designers to not work from scratch, and governments to see possible circuits on their existing infrastructure.

Index words: EVOLUTIONARY COMPUTING, FORMULA ONE, MULTI-OBJECTIVE OPTIMIZATION, DESIGN OPTIMIZATION, MOTORSPORTS, AUTOMOTIVE, SEQUENTIAL QUADRATIC PROGRAMMING, GENE LINKAGE, ALGORITHM COMPARISON

Evolutionary Design Optimization for a Formula One Car and Track

by

Daniel Harper

B.S., University of Georgia, 2022

A Thesis Submitted to the Graduate Faculty of the
University of Georgia in Partial Fulfillment of the Requirements for the Degree.

Master of Science

Athens, Georgia

2024

Evolutionary Design Optimization for a Formula One Car and Track

by

Daniel Harper

Dedication

I would like to dedicate this to my family and friends who helped me get here. To my parents, thank you for fostering a learning environment for me as a child and for helping mold me into who I am today. To my sister, thank you for being my first and longest friend in this world. To my wife Abby, thank you for always being there, for bearing with me through all of this work, and for being my forever best friend. To my grandparents, thank you for all the love and kind words of encouragement along the way. To my roommates, friends, and team, thank you for the memories, for always calling me higher, and for appreciating me for me. I love you all deeply, and I owe this work to you.

Acknowledgments

I would like to thank my major professor, Dr. Khaled Rasheed, for creating a passion for evolutionary computing during his course, as well as his help advising me throughout this thesis. He has been available and helpful through many impromptu office visits. I would like to thank my advisory committee, Frederick Maier and Kimberly Van Orman. They have instructed and guided me through courses and this work, providing unique perspectives and encouraging me to do my best work. I would like to thank my other professors during my undergraduate studies and this graduate degree, you all inspired me to dive deeper and learn constantly. Lastly, I would like to thank Erik Panu for being an excellent mentor, and without whom, I may have not explored combining my passions for AI and motorsports. I would not be here without all of your support.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background of Applied Fields

The first car powered by an internal combustion engine was built and patented by Karl Benz in 1886, called the Benz Patent-Motorwagen (Mercedes-Benz, n.d.). Over the century and a half since this creation, cars have become commonplace in many of the world's societies, with a reported 1.446 billion cars registered across the world (Hedges & Company, 2021). But world changing innovation gives rise to global competition. Henry Ford, pioneer of the early automotive industry, once said, "Auto racing began five minutes after the second car was built," capturing the fiercely competitive nature of the automotive industry (Petrány, 2012).

This competition came to a new peak in 1950 when the Fédération Internationale de l'Automobile (FIA) formally defined the pinnacle of motorsports, the fastest and most advanced racing series in the world, Formula 1 (F1). It began with the British Grand Prix at Silverstone, a track where F1 continues to race to this day (Williamson, 2022). While the F1 cars of 1950 certainly could not compete with today's cars, they were able to complete a 70-lap race and run for over 2 hours ("Race Standings", 1950). Formula 1 has always been a proving ground for automotive giants, with early titans like Alfa Romeo paving the way for modern dynasties such as Mercedes. F1 technology has also had a significant impact on the consumer automotive market, introducing technology to cars such as carbon fiber, steering wheel buttons, paddle shifters, and hybrid powertrains (Duff, 2022). AI and Machine Learning have been making their way into F1 for some time now, with a recent article by Forbes noting some use cases (Marr, 2023). There is even an official partnership between Amazon Web Services and F1 for graphics and data analysis ("F1 Insights by AWS", n.d.).

As the sport has been expanding, the popularity of motorsports has only been growing in recent years, also. A 2014 article published by the USA Today cited the Harris Poll results of the same year, where Auto Racing placed 4th in most popular sports among adults in the United States (Schwartz & McGarry, 2014). More recently, in May of 2022, Harris Poll published that one in three adults in the United States are interested in Formula 1 ("Harris Poll", 2022).

1.2 Motivation of Work

While Formula 1 is growing and adapting to accommodate the rapidly expanding fan base, they are also introducing new technical regulations to increase competition, decrease spending, and limit under the table dealings between their teams. Forbes reports that Ferrari's F1 team spent over $400 million on research and development in 2016 (Sylt, 2018). Now, in 2023, it is the third season with a cost cap limiting what teams can spend on research and running costs. Therefore, teams are, more than ever, having to find ways to cut costs while fighting for extra performance over the rest of the teams. As they hunt for extra performance, they must also be careful to ensure that all parts of the car conform to the standards set by the FIA, the governing body of F1. The FIA write and approve a document every year that is just under 200 pages with deeply technical regulations for every part of an F1 car. This varies from broader and more obvious regulations like the dimensions of aerodynamic pieces like the rear wing, to much more detailed restrictions like the fiber weight of carbon fiber or the chemical makeup of the engine fuel. If any of these rules are broken, the team can face harsh penalties on and off the track.

While it is possible that an Evolutionary Algorithm may not outperform genius F1 aerodynamicists like Adrian Newey, it could serve two largely beneficial purposes. Firstly, it could aid any team, from a championship contender to a backmarker, in providing a head start of viable and non-viable solutions. This means that a team may not have to go through modeling, wind tunnel testing, and evaluation of certain methods because the evolutionary algorithm will have told them it is not fit to survive. Secondly, smaller teams which have a smaller budget and less experienced engineers could benefit from a boost in design processes for a lower cost. Rather than spending tens of millions of dollars on traditional engineering and research methods, they could utilize an evolutionary algorithm to help guide their design philosophies. This would be especially beneficial when F1 does a routine reset on the technical regulations since there is a severe lack of data, design principles, and time during these changes. In this research, I seek to create and evaluate different forms of evolutionary algorithms to show the benefits that this field of evolutionary computing could provide to competitive motorsports, as well as drawing conclusions about what forms of algorithms may be best.

In order to support any of my claims for the abilities and applications of evolutionary algorithms in motorsports, it will be necessary to compare the evolutionary algorithms to a non-evolutionary approach to lap time optimization. I have selected to use Simulated Quadratic Programming as my comparison algorithm. The background and inner workings of this algorithm, as well as my reasons for choosing it and its results will be discussed in later chapters.

1.3 Contributions of this work

I implement 6 forms of evolutionary algorithm for Formula 1 car development that are able to achieve up to 18% improvement in lap time simulation results in a span of 6 hours or 12% in only 20 minutes. These algorithms include 3 forms of Genetic Algorithm (one being multi-objective), 2 forms of Evolutionary Strategies, and a Cultural Algorithm. I compare all 6 of these algorithms against a non-evolutionary optimization algorithm, Sequential Quadratic Programming, to verify that they outperform a non-evolutionary solution. I utilize industry knowledge and prior research into gene linkage to tie algorithmic performance more closely to the real world relationships between car parameters. I propose an implementation of a Cultural Algorithm that allows a team to not only discover ways to improve their car in the future, but also have an understanding of how their opponents might look to improve. This gives a team the competitive edge to see where they will be vulnerable or where they may be able to rise above the competition. These algorithms are able to properly optimize in a very narrow search space, with every indication that they would perform under even more realistic and tighter constraints in a future implementation.

I believe that evolutionary algorithms have applications in racing beyond just car design, so I also implement 2 forms of evolutionary race track generation. This includes discussions about how track designers, regulation makers, and racing teams themselves could use my track generation algorithms to improve their current workflows. One of these algorithms is able to create a viable Formula One track in only 2 minutes. The second algorithm requires more human-in-the-loop interaction, but creates much more organic and flowing corner structure. Together I think these algorithms show potential for how work such as this could be used in the racing industry in the future.

Much of the prior research utilizing evolutionary algorithms for the motorsports industry is reliant on video games. This could be in use as the fitness function or the entire problem set and goal. While this is a very valid area to research and has shown promise in the world of procedural content generation, player engagement, as well as simulated car performance and others, this bears little to no value to a Formula 1 team that cannot rely on a video game designer's interpretation of car performance or track design. Rather, these teams have very secretive lap time simulations centered on real world physics that could more easily be applied to my work.

1.4 Structure of Thesis

This project is intended to show the viability, scope of use, and a comparison of algorithms for use of evolutionary computing in the world of competitive motorsports. Chapter 2 is a follow up to the background of this industry by reviewing important evolutionary computing, other programming, and mathematical concepts used in this research. Chapter 3 is a literature review divided by what portion of my work that paper applies to. It contains many papers read before this work began, that helped shape my goals for what the outcomes of this project would be.

Chapter 4 begins with a discussion of the data and the inputs used for my algorithms. This is most important for the car optimization algorithms, as the lap time simulator I have chosen has very particular values to represent a car. In this section, I will outline what these variables represent, the values that I chose for them, and the bounds that they were constrained to evolve within. The rest of chapter 4 discusses the software and hardware used in this project. The software includes a listing of the languages, packages, and applications used in my development. It also includes a description of the lap time simulator used as a fitness function and of my translation of it to a different language. The hardware section is included to give a reference to the run time measurements that are included in algorithm comparison.

Chapter 5 discusses all of the algorithms compared in evolution of a Formula 1 car. Firstly, I used a standard Genetic Algorithm. However, I was disappointed in its detachment from realism and physical constraints, so I introduced a concept I refer to as "codependence" or "linkage", where genes rely on the values of other genes as they evolve. After these Genetic Algorithms, I also wrote an Evolutionary Strategy algorithm to see if the Strategy class could improve my results significantly. After developing an ES for car optimization, I wrote a Multi-Objective GA with co-dependence as another attempt to improve the realism of F1 car development. To explore newer algorithms, I discuss the concept of a cultural algorithm, my attempt at developing one, and where I think it could be further applied. Lastly, I implement a Sequential Quadratic Programming (SQP) approach to compare the results of evolutionary algorithms to a non-evolutionary method. This section includes the experiments and discussion for these algorithms.

Chapter 6 discusses the two Genetic Algorithms for track evolution. The first was based on a few previous works in the field of "procedural content generation" (PCG) for video games. It is a complicated representation, relying on a mixture of natural evolution from a genetic algorithm with some heuristics for curve generation. This algorithm relies on a subjective fitness function, which I will discuss in this chapter as well. After this first algorithm similar to the PCG radial algorithms, I decided to write a GA for the track representation used in OpenLAP. Finally, Chapter 7 is conclusions, limitations, and recommended future work.

Chapter 2

Background of Concepts Used

This is a brief overview of some of the more complicated concepts that I relied on for this thesis. This is not intended as a full description and cited works or similar should be referred to for further explanation of these topics, many of which can be found in the literature review of the next chapter or in the captions of figures.

2.1 Computing Concepts

2.1.1 Components of Evolutionary Algorithms

The first piece of constructing an evolutionary computing algorithm is your **representation**. In evolutionary computing, inherited from biology, you have a genotype and a phenotype. Your phenotype is the tangible, physical manifestation of what you are working on, in my case, settings of a Formula 1 car or segments of a track. The genotype, or representation, is the encoding of these physical objects. All of the algorithms developed in this work use lists as the representation. I use a list of floats, a list of tuples, and a 2d list for my algorithms. These representations are constructed to create an **individual**, this is one object of what you are evolving, such as a car or a track. Then you create multiple individuals and call this set of individuals your first generation of your **population**.

Once you have created a population, you evaluate the **fitness** of each member of the population, akin to fitness in Darwinian Evolutionary theories. The fitness function is critical to the success of your algorithm and defines what you believe to be a good end result. You can perform two types of fitness evaluation: single objective and multi-objective. As is evident by the names of these types, single objective optimization modifies the individual to maximize (or minimize) the result of your fitness function, where multi-objective optimization tries to find a balance between multiple fitness values using pre-provided weights and domination calculations.

In Multi-Objective Optimization, a calculation that is usually done is to find the **Pareto Front**, or **Pareto Frontier**. This is a set of individuals from an evolutionary algorithm with multiple objectives. When you perform multi-objective optimization, the algorithm is finding a balance between the fitness measurements, as improving one category of fitness might worsen another. Therefore, instead of simply measuring by quantitative fitness as with single objective optimization, we find a set of individuals who are called "non-dominated". What this means is that these individuals have achieved a similar level of balance from one another, and there are no individuals in the population that beat them in every way. The Pareto Front is a way to retrieve multiple viable solutions to all of your fitness optimizations to avoid algorithmic bias and ensure that the desired outcome is achieved. After you have evaluated every individual in your population, you need to **select** the most fit individuals to serve as the parents for the next generation of your population. There are many different forms of selection functions, and you can always define your own. In all of my algorithms in this work I am using tournament selection. Tournament selection takes n individuals randomly from your population and selects the one with the highest fitness. This process repeats until you have the number of parents you desire selected.

Once you have selected your parents, you need to perform genetic **crossover** and **mutation**. As with selection, there are numerous existing functions to perform crossover and mutation. In this work, I used mostly pre-existing functions with modifications to fit my representation better. The specifics of these functions will be described in their corresponding algorithms. Crossover is a process intended to create children that are some kind of combination of the parents used to create it. Mutation is intended to promote diversity and avoid local optima by randomly or heuristically modifying some individuals.

The final component that I have used in my track evolution is a **repair** function. This is a function that occurs during fitness calculation to ensure that your individuals are not behaving and optimizing in a way that you do not want. In my case, I used it to ensure that there were no duplicated coordinates in track creation. If there was a duplicate coordinate, it was replaced by a new randomly generated coordinate.

2.1.2 Types of Algorithms

Genetic Algorithms (GA) are one of the most commonly used and most well known forms of Evolutionary Algorithms. A GA maintains a population and performs genetic operators as described above. It typically performs selection, crossover, and mutation to create future generations of a population from the existing population. GAs are more variable in the operators that they use, the representations of the individuals, and the types of problems that they can optimize.

Evolutionary Strategies (ES) were introduced to perform numerical optimizations. The original versions of Evolutionary Strategies did not use crossover and performed Gaussian mutation until children are good enough to supplant the parent individual (ES originally used one parent per child instead of 2). My ES does include crossover and utilizes a *Strategy* class which provides values for Gaussian mutation to be performed. I used a (μ, λ) ES to increase population size from selection to replacement.

Cultural Algorithms (CA) were introduced much later, and introduce concepts of belief spaces with cultures and societal and social interactions between individuals in a population. Rather than using crossover operators like many EAs, Cultural Algorithms perform selection for the purpose of belief space updating. In my CA, I used rank based selection to take the top n individuals whose gene values are used to shift the bounds from our belief space. Since new individuals are created only using the belief space, the selected individuals get to modify where future generations genes will be bounded, though these bounds can be broken by mutation of later individuals to ensure some diversity is possible. Where I see there being significant importance for CAs in motorsport would be in combining the CA with something like the Island Model from Evolutionary Computing, which I will discuss in the Future Work section of the final chapter.

Sequential Quadratic Programming (SQP) is a form of algorithm introduced by Wilson in 1963 (Wilson, 1963). SQP is a form of iterative optimization algorithm that relies on quadratic approximations in conjunction with bounds and constraints to optimize non-linear systems. You begin with an initial guess for your variable values. Provided an objective function, SQP seeks to create a quadratic sub-problem that can approximate the relationships between variables as found in the objective function itself. Then, it will solve this sub-problem and use a search method to find the direction to iterate values with. It ensures that upon iteration, all bounds and constraints are satisfied and checks whether there has been a significant change in objective function value. If it finds the change to be insignificant, it will terminate iteration, having converged on a solution. If it does have a significant enough change, it will continue and repeat the above steps.

2.2 Mathematical Concepts

Interpolation is a mathematical approximation of how to connect data points. It allows you to take a sampling of function values for a given step size, giving you the ability to reduce compute complexity for certain plots and formulas. In my translation of OpenLAP, which is discussed in further detail later on, I used *interp1d* to perform simple linear interpolation for car, track, and lap simulations. However, more critical to my research is Piecewise Cubic Hermite Interpolating Polynomial (PCHIP) interpolation. PCHIP Interpolation not only considers the data points you are interpolating between, but it considers the derivative at each of those points to ensure a higher level of smoothness between complex curves. I used this in my evolution of a track to get the x and y coordinates of each segment as they went along, and without it, the curves would have been much more jagged, and less feasible for a real world racing track.

In my first implementation of track evolution I use **B-Splines** made up of **Bezier Curves**, based on the research of (Togelius et al., 2007) and (Loiacono et al., 2011), as will be further explored in the literature review of Chapter 3. Bezier curves are a polynomial function defined by a number of control points. It is guaranteed to pass through the first and last point, and while it doesn't pass through the other control points, it is curved towards them. This enables you to create more variable and complex curves given only a few starting points. As you can see in the image, the Bezier curve passes through points P_0 and P_3, while only curving towards P_1 and P_2. By creating another Bezier curve beginning at point P_3, and repeating this curve creation from the previous endpoint, we create a B-Spline constructed of Bezier Curves.

A triangular distribution is between Gaussian and Uniform distributions. When I was performing track evolution, I wanted a distribution that allowed me to sample mostly on a normal distribution, but without a good sampling of tracks to inform my inputs for Gaussian random sampling, I opted for a Triangular distribution. This samples between points a and b, with an optional manual weighting towards c. I chose to leave c at its default value, which is the midpoint between a and b.

Chapter 3

Literature Review

There is not as much research in evolutionary computing relating to motorsports or the racing industry, as compared to some other fields of engineering. However, this chapter contains some examples of algorithms that have been used in applications for motorsport, or any similar published works using evolutionary computing for motorsports.

3.1 Related to Evolution of a Car

There were a few specific papers that inspired my algorithm design and experiment setup in this work. The first was Luque et al., 2020, which is where I drew inspiration initially to use EAs to optimize over a short race distance like a single lap. However, in this work, they chose to optimize an electric car to match the performance of an internal combustion engine vehicle. Instead, I wanted to optimize race car setups to be faster than the current standard rather than just matching them in a different way. Castellani and Franceschini, 2003 is the most direct inspiration for the application of algorithms here. They applied GAs to modify race car setup parameters for laptime. However, presently there are a few shortcomings of this work. Firstly and most simply, the data and information about Formula 1 cars is quite outdated. The car design changes every few years, and the cars they were optimizing are a couple of decades old by now. But further on the research side, they were very loose with their calculations of laptimes using "steady-state cornering" which does not involve nearly as many vehicle and track dynamics and simply calculates the speed that a car can carry through a turn of a given radius and then times it at that speed. By using the OpenLAP simulation, and hopefully future work with even more accurate simulations, I was able to account for more nuanced motion and performance through a corner as very few corners are taken as "steady-state" in real life. Also, they used algorithms to optimize from human designed setup groups, rather than operating values more independently. I felt that this introduced some human bias in the design, rather than allowing the algorithm to work freely. Similarly, Wloch and Bentley, 2004 takes a Formula 1 car from 20 years ago and optimizes setup parameters for a race car over a lap. However, they are using a video game as a fitness function, which I feel detracts from the applicability of the work for a race team. Racing video games are designed at least partially for player enjoyment and not always for accuracy, so they cannot be considered a reliable source of lap time simulations. As with Castellani and Franceschini, 2003, I felt that the lack of fully variable control for the algorithm in mutation, crossover, and fitness calculation was detrimental to the impact of this work.

A few other papers influenced my understanding of the inner workings of algorithms, and exploration of new algorithm types. First, Candelpergher et al., 2000 discussed the "G-G Envelope", which has come to hold several names in modern racing telemetry analysis. This is an earlier version of the "GGV Map" that is used in the OpenLAP representation of a vehicle. Understanding the longitudinal and latitudinal acceleration forces available given a cars current velocity is paramount to calculating performance through a corner, further pushing beyond the "steady-state" approach in Castellani and Franceschini, 2003. As I was working on the EAs development and theorizing future work, I came across an algorithm from Reynolds, 1994, the Cultural Algorithm. I began to implement the version that will be shown later in this work, but still believe that this algorithm may hold the most potential uses for a racing team in the future, out of those shown here. To better understand how cultural algorithms might work in a more modern application, I read Saad et al., 2022 for deeper background knowledge.

One of the weaknesses I noticed early in my testing was that values that are normally related to one another were able to evolve individually. I began to think about how to connect these values together when I found literature about gene linkage for evolutionary computing. Most of the literature used here was simply to see how linkage was used in other algorithms and applications to shape how I might apply this work here. Two books, *Linkage in Evolutionary Computation*, 2008 and *Exploitation of Linkage Learning in Evolutionary Algorithms*, 2010, contain a vast array of theories and applications related to gene linkage. Some of the basics of these are applied here with my "codependent" algorithms, while others like gene linkage for algorithms like the cultural algorithm are simply theorized and considered for future work. Another paper I read for applications of gene linkage in tightly constrained optimization is from Adair et al., 2016

3.2 Related to Sequential Quadratic Programming and Algorithm Comparison

When first advised and interested in applying an algorithmic comparison with SQP against my EAs, I read through its introductory paper (Wilson, 1963) and a paper documenting changes through a few decades since its introduction, Boggs and Tolle, 1995. These papers helped me more deeply understand SQP and why it fit as a competitor for the EAs. My first thoughts for performing algorithm comparison is from Hayward, 2007. He states the challenges associated with trying to perform an algorithm comparison in optimization for a Formula SAE vehicle, but I believe that the difficulties do not overshadow the importance of understanding how EAs perform in a given constraint space versus a non-evolutionary algorithm like SQP.

In searching for works using SQP for motorsports or similar engineering principles, I read Sheta and Turabieh, 2006. While focused on the energy industry rather than racing, it helped me see an example of

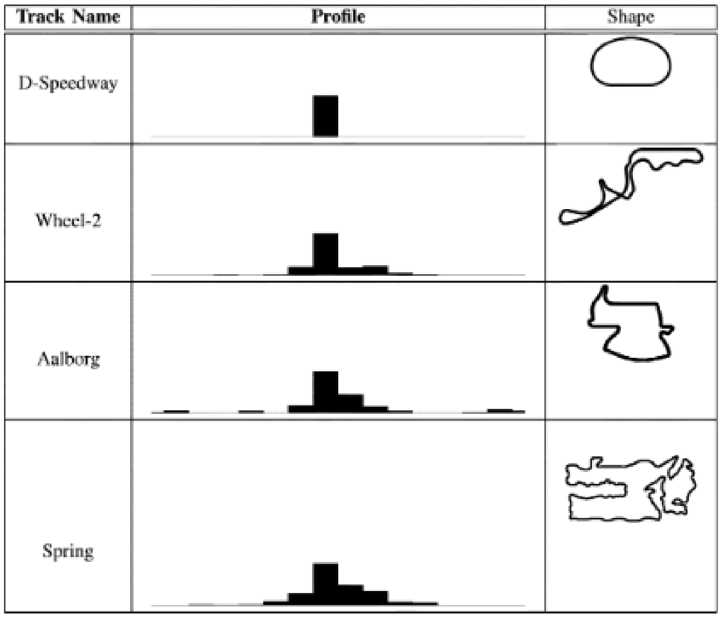
SQP being used for engineering design optimization. Similarly, as I was working on this research, Zhu and Borrelli, 2024 published a work creating a decision agent for autonomous racing vehicles using SQP.

3.3 Related to Evolution of a Track

The first paper I read performing evolution for track generation was Loiacono et al., 2011. In their paper, they use evolutionary computing to create fictional racing tracks. While their approach is largely focused on the industry of racing video games, I think that it is interesting to consider optimizing a car, testing a car, and developing a car, based on its simulations over a fictional circuit. While this may not be as directly applicable from a fictional track to F1 performance, I thought that this paper was an excellent example of a different way that evolutionary computing has been applied to the automotive and motorsports industries. I adapted much of my GA's design, as well as a significant portion of my fitness function from this paper. They defined one measurement of fitness to be the "curvature profile" of a track. The formula is as follows:

$$H(C) = -\sum_{i=1}^{16} c_i \log_2 c_i$$

This formula represents the entropy of the curvature profile, which is how they elected to measure the number of different types of turns in a track to encourage diversity of curvature. I adapted this formula slightly for my own fitness function in my Multi-Objective track generation algorithm. The results of their measurement on some example tracks from the video game "TORCS" can be seen in Figure 3.1.



Track Name	Profile	Shape
D-Speedway		
Wheel-2		
Aalborg		
Spring		

Figure 3.1: Curvature Profile Samples. Source - Loiacono et al., 2011

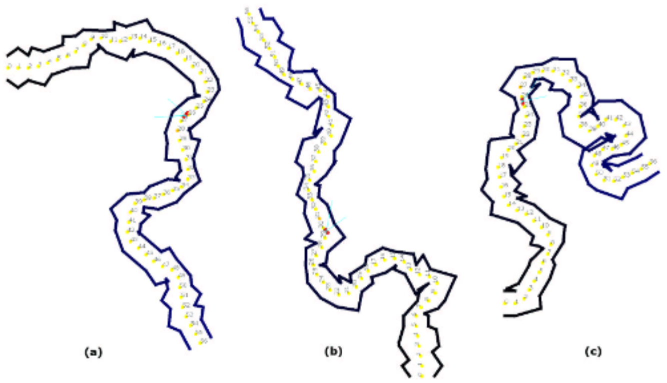


Figure 3.2: Sample Evolved Tracks for Increasing Experience. Source - Togelius et al., 2006

This paper cites Togelius et al., 2006 for its work in procedural content generation for video games. Specifically, they use evolutionary algorithms to optimize towards a track suited for different players. They would create a model of the play style and preferences of players of a racing game and then create a track that is believed to be more suited to their driving style and experience level. The track was generated segment by segment, which follows the field of PCG more closely than track design for a permanent or street circuit in F1. While the segments do not create the smoothest track, it is evident that across the three samples seen in Figure 3.2, that the track is getting more complex for drivers with more experience and who prefer a greater challenge.

These authors followed up their work with Togelius et al., 2007, improving their results specifically in track modelling and the representation used. They compared between three different track initialization styles to expand upon the segment method from their first work. Their new radial representation results in simpler tracks visually, but is still able to diversify between experienced and beginner drivers, an example result is shown in Figure 3.3. This radial representation in the 2007 paper served as inspiration for my first GA for track generation, while their 2006 work and the segment based method bears resemblance to my second GA involving OpenTRACK.

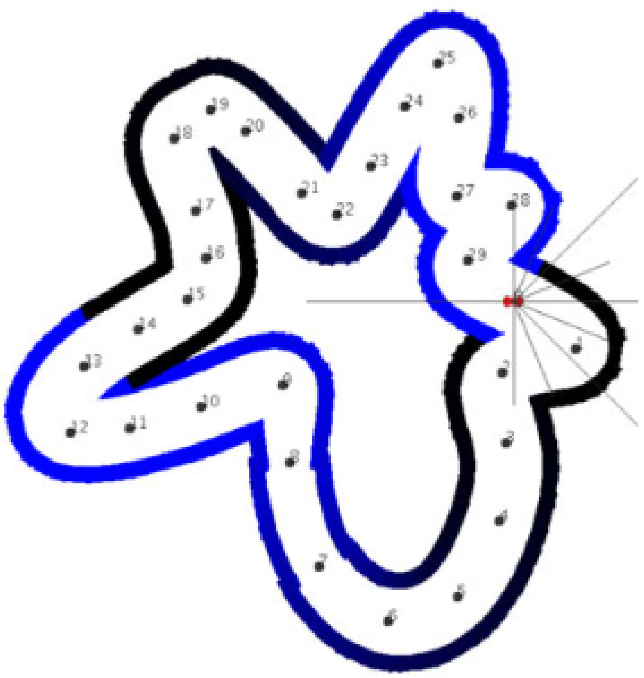


Figure 3.3: Radial PCG Evolved Track for Experienced Player. Source - Togelius et al., 2007

Chapter 4

Data, Software, and Hardware

4.1 Data

For initial population setup and constraints in evolution, I implemented everything with an array of bounds for each gene. The names of each value, its bounds, and its units can be seen in Table 4.1 below.

Table 4.1: Representation evolving variables with bounds and units

Variable Name	Bounds	Units
Mass *	908, 1015	kg
Front Mass Distribution *	44.5, 54	%
Wheelbase *	3460, 3600	mm
Lift Coefficient	-4.4, -2.8	-
Drag Coefficient	-1.1, -0.7	-
Front Aero Distribution	35, 55	%
Frontal Area **	0.9, 1.4	m^2
Disc Outer Diameter *	325, 330	mm
Pad Height **	52, 52.8	mm
Caliper Number of Pistons *	1, 6	-
Front Cornering Stiffness **	800, 1200	N/deg
Rear Cornering Stiffness **	800, 1200	N/deg
1st Gear Ratio **	2.21, 3	-
2nd Gear Ratio **	1.79, 2.2	-
3rd Gear Ratio **	1.51, 1.78	-
4th Gear Ratio **	1.31, 1.5	-
5th Gear Ratio **	1.15, 1.3	-
6th Gear Ratio **	1.05, 1.14	-
7th Gear Ratio **	0.9, 1.04	-
8th Gear Ratio **	0.7, 0.89	-

* - from FIA regulations, ** - from OpenLAP.
Others found elsewhere and discussed in this section.

The FIA's technical regulations provided many of the bounds for these values. Their minimum mass is listed at 796kg in Article 4 Section 1, while a full load of fuel is estimated to have a mass of around 110kg ("F1 Technical Regulations 2023", 2022). This mass was only recently lowered from 798kg, and all of my runs were performed with this 798+110kg lower bound. The upper bound was intended to allow room for evolution and for penalties in the Codependent GA. I came up with this bound on my own, with full knowledge that faster cars will have masses very near to the bottom bound. Section 4.2 gave the second bounds explicitly for the distribution of the mass from the center of the car. The upper bound for wheelbase is from Section 3.4.2 with a maximum of 3600mm, and I implemented the lower bound myself. Lift Coefficient, Drag Coefficient, and Frontal Area are all three interconnected through complicated physics, beyond that of the OpenLAP simulator. The formulas for these values are the following:

$$c_d = \frac{D}{\rho V^2 A/2},\ c_L = \frac{L}{\rho V^2 A/2}$$

In these formulas, c_d and c_L are the coefficients of drag and lift. D is drag force, L is lift force, ρ is mass density of fluid, V is velocity of object, and A is frontal area. Clearly, from these formulas, drag, lift, and frontal area are all linked together, which was my main inspiration for co-dependence in my GA. Based on various online posts like CarThrottle and the sample values from OpenLAP, I set the bounds for these values at -4.4 to -2.8 for drag coefficient, -1.1 to -0.7 for lift coefficient, and 0.9 to 1.4 square meters for frontal area (Fernie, 2016).The Front Aero Distribution was another value that was difficult to come by in the technical regulations, with most of the aerodynamic restrictions being dimensions and flexing of panels. However, there was an article posted to The Race, a prominent F1 news site, written by Gary Anderson, former technical director for F1 team Jordan in the 90's (Anderson, 2022). Anderson writes that he expects a 2022 F1 car to generate about 40% of the total downforce in the front half based on the new underfloor design. Based on this and given OpenLAP's sample value of 50%, I decided to set the bounds from 35 to 55 to give a 5% margin on both sides. The brake disc diameter is regulated by Section 11.3.3. This section has different values for the front and the rear brakes, but the OpenLAP simulator only takes one value for all four wheels. I chose to use the larger bound of 325 to 330 mm as it seemed a better representation of the overall brake discs of the car. The brake pad height was calculated based on the sample value from OpenLAP plus the 2023 regulations on brake discs. The OpenLAP sample F1 car is most likely based on a car from the 2009-2013 era of F1. This car's brake discs were listed at 250mm with pads at 40mm. I found the ratio between the sample car's discs and pads, applying this to the 2023 bounds to get the values of 52 and 52.8. For the number of pistons per brake caliper, the regulations in Section 11.2.4 state that there is a maximum of six pistons per caliper, so I set the bounds from 1 to 6. The technical regulations do not place a limit on the stiffness of the suspension, they are more focused on ride height and travel distance. The OpenLAP simulator recommends a value between 800-1000, which I decided to modify to 800-1200 to accommodate the possibility of stiffer suspension handling the greater mass of the new cars better. The rear was set to the same bounds. The gears of the gearbox are restricted in relative size to one another, but given that I had no physical examples to measure and convert to my needed format, I had to derive my bounds from the OpenLAP simulator. The sample car only had 7 gears, where the 2023 car must have 8 (Section 9.7.1). From here I adapted a general range of gear ratios from 3 at the upper end, reaching close to the value of 1 from the sample car for 7th gear, and going a bit below 1 for 8th gear. The margins from bottom to top bounds decreased for each gear, starting around 0.8 and ending around 0.2. These were of my own decision, but are more flexible in the codependent GA, which will be discussed later.

Table 4.1 represents the 20 values evolved in my setup, all to work with the OpenLAP simulator. There were another 32 values that I decided did not need to be evolved and were therefore hard coded. Some of these values were left as given by OpenLAP's default car file, while others were modified to be more representative of the newest generation of F1 car, these variables and their values can be seen in Table 4.2, as well as the torque curve of the engine in Table 4.3.

Of the 31 non-evolved values in the representation of a car, I modified 6, leaving the other 25 as recommended by OpenLAP. The unchanged values were values that did not appear to have any difference between 2013 and 2023, with many of them being coefficients or multipliers, or simply names and identifiers of the car. The first value that changed was the diameter of the brake caliper pistons. These were 40mm on the sample car, so I performed similar up-scaling to 52mm as with the brake pads. The same was done on the master cylinder piston, raising it from 25mm to 32.5mm. The tire radius has increased over the last 10 years, with Section 10.8.1 stating that the tires must be between 345 and 375mm in the front and 440mm and 470mm in the rear. The teams do not get to decide this, but the tire supplier, currently Pirelli, does. I opted to set the tire width at the average of the rear bounds, giving 455mm. The longitudinal and lateral friction load ratings were previously 250kg for a mass of 650kg. Therefore, I divided our minimum mass by 2.6 (the factor in the previous values) and rounded to 350kg. Lastly, the final gear reduction was increased from 7 to 8 to account for having an extra gear in the new cars. All other values were identical to the given and recommended values in OpenLAP's sample F1 car. The SQP algorithm used the exact same bounds as all of the evolutionary algorithms.

4.2 Software

4.2.1 OpenLAP Laptime Simulator

The OpenLAP Laptime Simulator was written by Michael Chalkiopoulos and released on his GitHub page in April of 2020 (Chalkiopoulos, 2020). He posted his code with a full public use license, as well as creating a YouTube series explaining how the simulator works and what different parts of the algorithm mean. This project contains three major parts: vehicles, tracks, and laps. Each of these has a dedicated script and they work closely together. To perform a lap simulation, you must first create a vehicle and track object using those two scripts. The vehicle script takes an input of 51 values in a Microsoft Excel spreadsheet, extracts the values from the spreadsheet, and uses these values to calculate more detailed representations of different parts of the car. It is saved as an "OpenLAP Vehicle" file , with output shown in Figure 4.1. A similar process is performed for the track. The tracks are represented by an Excel spreadsheet with coordinates and elevation data. These are extracted and converted to have a better understanding of

Table 4.2: Non-evolved values used by OpenLAP simulator

Value Name	Value	Units
Name	'Formula One'	-
Type	'Open Wheel'	-
Steering Rack Ratio	10	-
Lift Coefficient Scale Multiplier	1	-
Drag Coefficient Scale Multiplier	1	-
Air Density	1.225	kg/m^3
Pad Friction Coefficient	0.45	-
Caliper Piston Diameter	52	mm
Master Cylinder Piston Diameter	32.5	mm
Pedal Ratio	4	-
Grip Factor Multiplier	1	-
Tire Radius	457	mm
Rolling Resistance	-0.001	-
Longitudinal Friction Coefficient	2	-
Longitudinal Friction Load Rating	300	kg
Longitudinal Friction Sensitivity	0.0001	1/N
Lateral Friction Coefficient	2	-
Lateral Friction Load Rating	300	kg
Lateral Friction Sensitivity	0.0001	1/N
Power Factor Multiplier	1	-
Thermal Efficiency	0.35	-
Fuel Lower Heating Value	47200000	J/kg
Drive Type	'RWD'	-
Gear Shift Time	0.01	s
Primary Gear Efficiency	1	-
Final Gear Efficiency	0.92	-
Gearbox Efficiency	0.98	-
Primary Gear Reduction	1	-
Final Gear Reduction	8	-
9th Gear Ratio	N/A	-
10th Gear Ratio	N/A	-

Table 4.3: Torque Curve of F1 Engine

Engine Speed (rpm)	Torque (Nm)
1000	529
2000	540
3000	564
4000	598
5000	690
6000	702
7000	782
8000	817
9000	776
10000	776
11000	750
12000	736
13000	529
14000	460

the track, with the output as shown in Figure 4.3. Finally, the "OpenLAP VEHICLE" and "OpenLAP TRACK" files are input to the OpenLAP script, which calculates how fast the car could be going on the limit of its grip at every point on the track, resulting in an output lap time, plots shown in Figure 4.5. One of the most beneficial parts of this project is that it includes detailed plots to give a better understanding of everything going on in the algorithm. These plots are all included here. The OpenVEHICLE plot includes 4 separate subplots. The first, in the top left, is the Torque and Horsepower curve plotted over the rpm of the engine. The plot on the middle left is gearbox information, with two y-axes representing rpm and gear number as you accelerate to top speed. The bottom left plot displays information about different forms of traction (engine, aerodynamic, tires, etc) at different speeds. The larger plot on the right is the GGV Map, sometimes also called the Acceleration Envelope or Performance Envelope. It represents the levels of lateral and longitudinal acceleration possible for a car at varying acceleration and steering values, which is the same as representing the "G-G envelope" from Candelpergher et al., 2000 across different possible velocities. The OpenTRACK output includes 6 plots. The first on the left is a map of the track, with color coding for the 3 sectors it contains. The other 5 plots are all plotted over distance, representing how the track changes over its length, and they represent curvature, elevation, incline, banking, and grip factor. In the case of the tracks used here, banking and grip factor are constants of 0 and 1 respectively. The OpenLAP output shown in Figure 4.5 contains 7 plots. The first plot represents speed over the length of the track, while the second is a duplicate of the elevation and curvature from the track. The third plot represents the longitudinal and lateral accelerations of the car over the course of the lap, and the fourth plot represents the amount of throttle and brake pedal input at each point. The fifth plot contains the angle of the steering wheel. The sixth plot on the bottom left represents a transformed perspective of the GGV plot from the vehicle, with dots added for each point on the track and where the car was in its longitudinal and lateral accelerations based on the acceleration forces over the whole lap. The last plot in the bottom right uses a heat map line of the track to represent the speed of the car similar to the first plot.

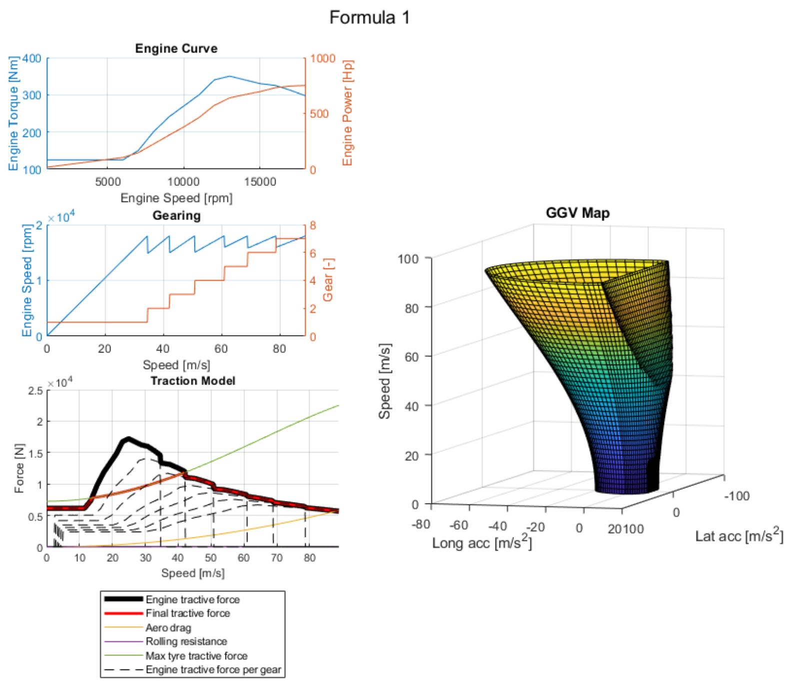


Figure 4.1: OpenVEHICLE Output Plots. Source - (Chalkiopoulos, 2020)

4.2.2 OpenLAP-Python Laptime Simulator

I decided to translate the OpenLAP simulator from MATLAB to Python to improve connectivity of this program with my evolutionary algorithms and to improve run time. I have included here the plots that my Python translated version outputs, in order to show the close similarity to the original program. As the original repository was open source, I have left my translation to Python as an open source program as well, available on my GitHub (Harper, 2023). The translation of the lap simulator from MATLAB to Python was very incremental, with a general syntax implementation followed by line by line debugging and variable value checking. The algorithm was calibrated to be as close as possible with the sample Formula 1 vehicle over the provided track file for Spa-Francorchamps. The Python version has a lap time of 99.23702 and the MATLAB (original) version has a lap time of 99.238.

I am pleased with the accuracy between my OpenVEHICLE and OpenTRACK plots seen in Figures 4.2 and 4.4, respectively. The only major, distinguishable differences between my plots and the originals being in the scaling and colors between MATLAB and Matplotlib in Python. With the OpenLAP plots in Figure 4.6, there are a couple of differences, however. The heat map key is missing from the track speed map, and the points over the GGV map are improperly distributed. However these are only graphical differences, and all internal values have been checked, with the final testing lap time being within 0.0001% of the MATLAB version over the sample track Spa-Francorchamps.

4.2.3 Development Software, Languages, and Packages

Given the scope and purpose of this thesis, I used a variety of software applications, tools, and packages to create these algorithms. For the writing and running of the code itself, I used the IDE Visual Studio Code for its flexibility, ease of use, and my existing comfort with the platform. I used GitHub for all version control and syncing between different devices. All of my code is written in Python version 3.10.6, but the OpenLAP open-source lap time simulation which I adapted for my use here was originally written in MATLAB with the Add-ons of DSP System Toolbox, Phased Array System Toolbox, and Signal Processing Toolbox. I performed the translation of MATLAB code from OpenLAP to Python code myself for ease of use and to be able to further use this code in the future, as discussed in the last section. I used version R2022b of MATLAB to test and translate the OpenLAP repository, which I downloaded from GitHub and have cited in the references at the end of this paper (Chalkiopoulos, 2020). The full list of packages I used in Python are as follows:

- array
- copy
- datetime
- DEAP
- functools
- itertools
- math
- matplotlib
- MPL toolkits
- numpy
- os
- pandas
- pickle
- random
- scipy
- shutil
- tabulate
- time
- tqdm
- warnings
- xlsxwriter

4.3 Hardware

The majority of the developing and all of the measured runs of these algorithms were done on a computer with the following specifications:

- Operating System: Windows 11
- CPU: Ryzen 9 3900X (12-cores)
- RAM: 32GB (DDR4)
- GPU: NVIDIA GeForce RTX 3070 (8 GB of VRAM)

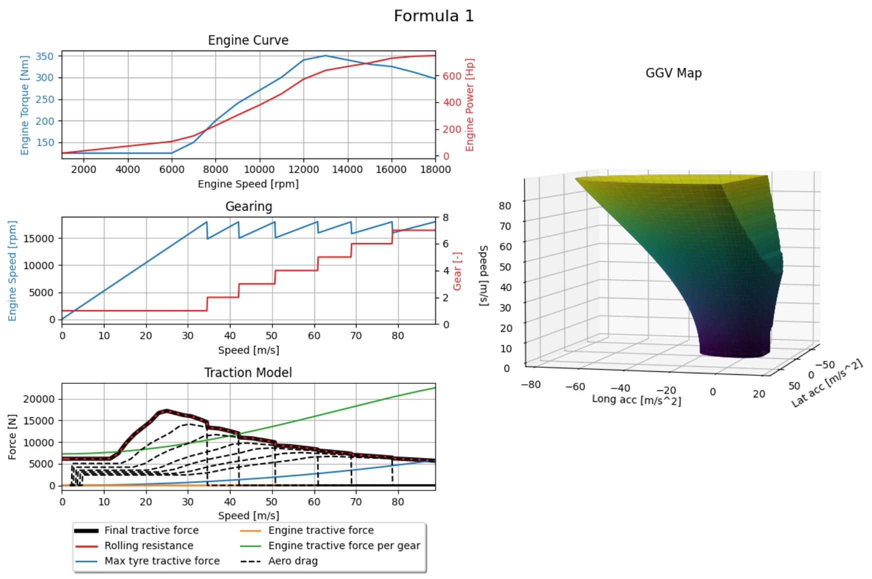


Figure 4.2: OpenVEHICLE-Python Output Plots

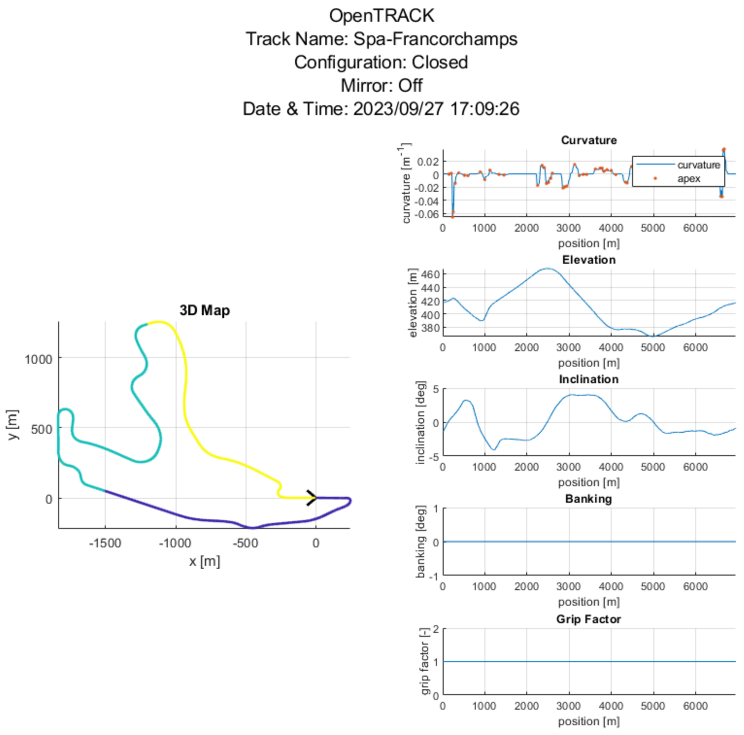


Figure 4.3: OpenTRACK Output Plots. Source - (Chalkiopoulos, 2020)

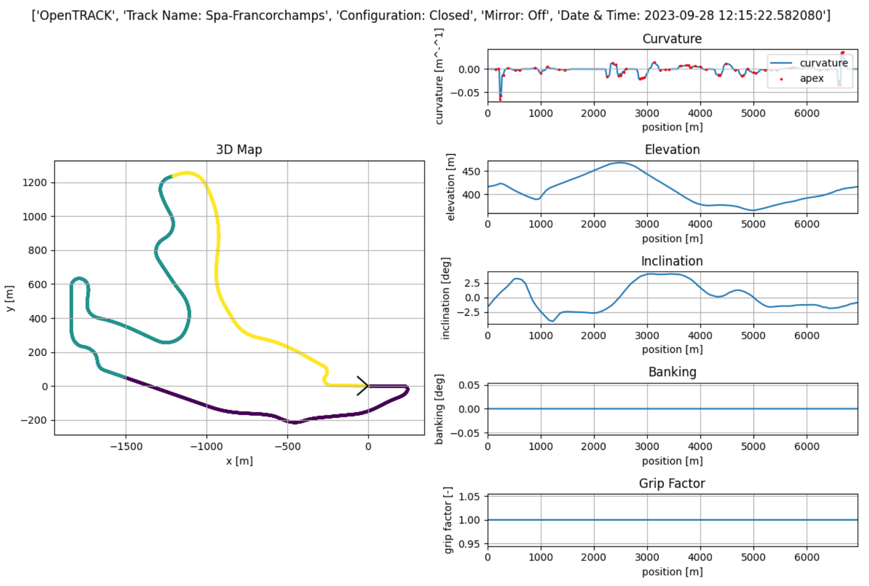


Figure 4.4: OpenTRACK-Python Output Plots

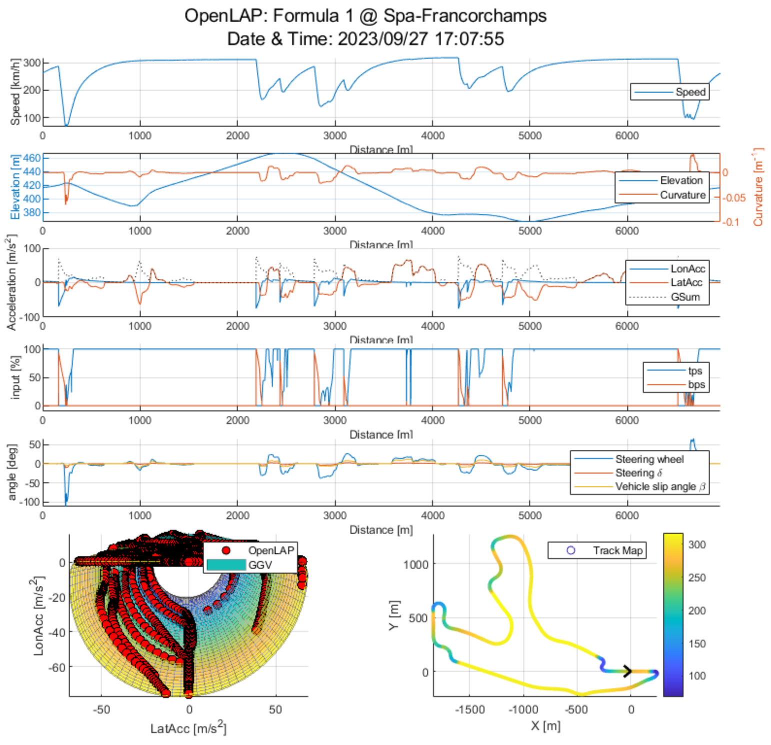


Figure 4.5: OpenLAP Output Plots. Source - (Chalkiopoulos, 2020)

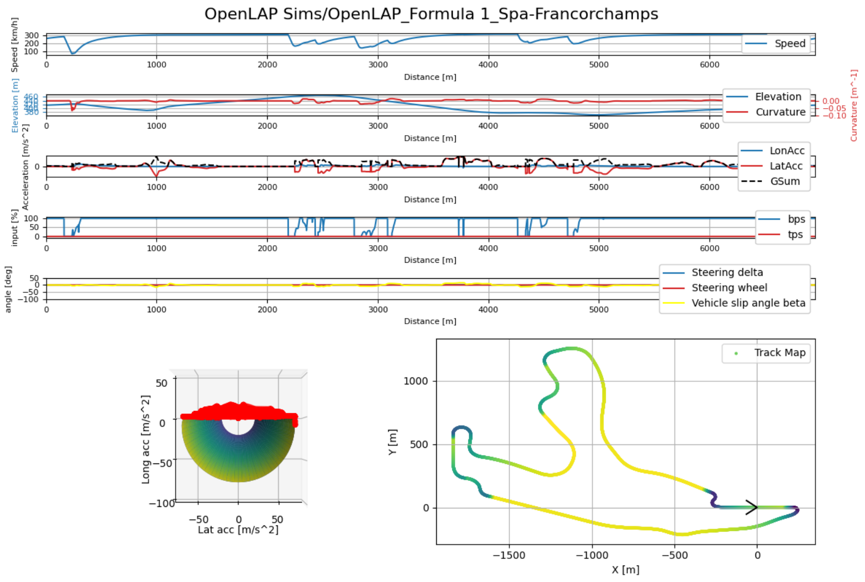


Figure 4.6: OpenLAP-Python Output Plots

Chapter 5

Evolution of an F1 Car - Experiments and Discussion

5.1 Experiments

The data shown for experiments and drawing conclusions about algorithm comparison include run time for computational efficiency, algorithm parameters, and output results. This chapter outlines the logged data from sample runs of each of the algorithms. 10 runs are included from each of the first 4 algorithms and 5 runs of the CA and 3 runs of Multi-Objective are also logged here. This is far from every run that was performed throughout this research, and all 48 of these were performed when each of the other algorithms were completed to ensure that no optimization changes were made during testing based on the outcomes of runs. The logged data from the main 4 algorithms was total run time using the 'datetime' package of python, the crossover and mutation probability hyper-parameters, and the lap time of the best individual using DEAP's hall of fame. There will also be some charts showing how fitness changed over generations for some discussion about how each algorithm tends to be optimizing. The first section will be mostly raw data as taken from the code, and the second section will be further discussion on interpretations of this data. In the discussion of these, I will include details on how each algorithm converged towards its optima, where it was strong, where it was weak, and what can be drawn from it to create something better in the future.

Hyper-parameters were chosen by me based on findings during development and testing. As discussed in the algorithm details, GA operator probability is independently between 0 and 1, where ES probabilities are summed between 0 and 1. Therefore, not all 10 of these runs were performed with the same probabilities. Both GAs used the same for all 10, as with ES also, and 5 of the 10 runs match up between the GA and ES algorithms. All of these runs were performed with a population size of 5 and a generation count of 50. For the CA, there is no concept of crossover and mutation probabilities. Utilizing full replacement and rank based selection, I included a varying population size, generation count, and number of individuals to be accepted by our selection function.

In terms of logged output, I included the total run time of the algorithm and the lap time of the best individual as recorded by DEAP's hall of fame function. All run times are in minutes and seconds unless there is a third number, which would represent hours, minutes, and seconds. As with input, the CA is a little different. Since I manually coded all of this algorithm, rather than using DEAP, best lap time is stored by using the situational belief space update. Each of these rows represents a single run of one of these algorithms, not an averaging of them. It would have been too complicated and take too long to try and perform multiple runs for each parameter setting. Since we aren't making a serious decision on algorithm comparison, merely understanding strengths and weaknesses, I think that single-run performance over different parameters suffices.

After testing and comparing algorithms, I selected one of them to convert into a multi-objective optimization function to optimize the car for all of the provided tracks from OpenLAP. I hoped to do this for each of the algorithms, but the complexity of multi-objective optimization using OpenLAP is quite high, as will be evident in the second track algorithm in the next chapter. As just discussed, this is not intended to suggest that this algorithm is the best choice, but simply to give an example of an expansion of this research into a wider use case for a competitive motorsports team.

5.1.1 Genetic Algorithm

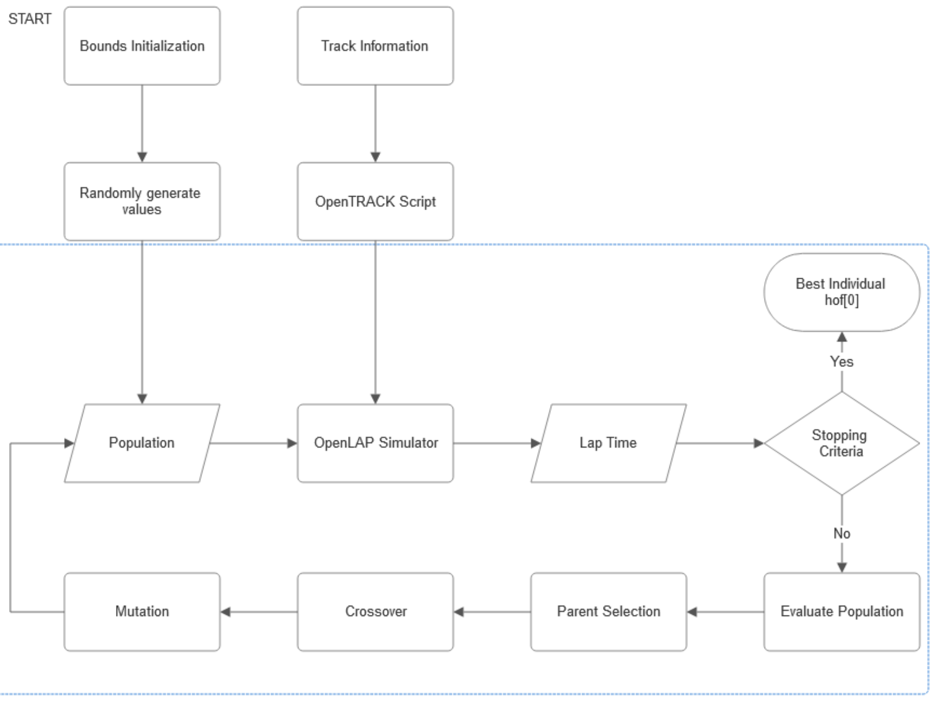


Figure 5.1: Genetic Algorithm Flowchart

The first algorithm that I implemented was a standard Genetic Algorithm. This was written using the 'eaSimple' function from DEAP. This algorithm used a population size of 5 over 50 generations. A custom mutation function titled 'mutationpower' was used with this algorithm with an 'indpb' parameter of 0.3. This mutation function comes from a previous project that I did and was influenced by descriptions in Luque et al.'s paper and MathWorks documentation for MATLAB (Luque et al., 2020), ("Genetic Algorithm Options - MATLAB", 2023). I used the Intermediate crossover function using an array of the bounds that I discussed in chapter 4 and a ratio of 0.8. I also used tournament selection with a tournament size of 3 for this GA. A flowchart of this algorithm is seen in Figure 5.1.

5.1.2 Codependent GA

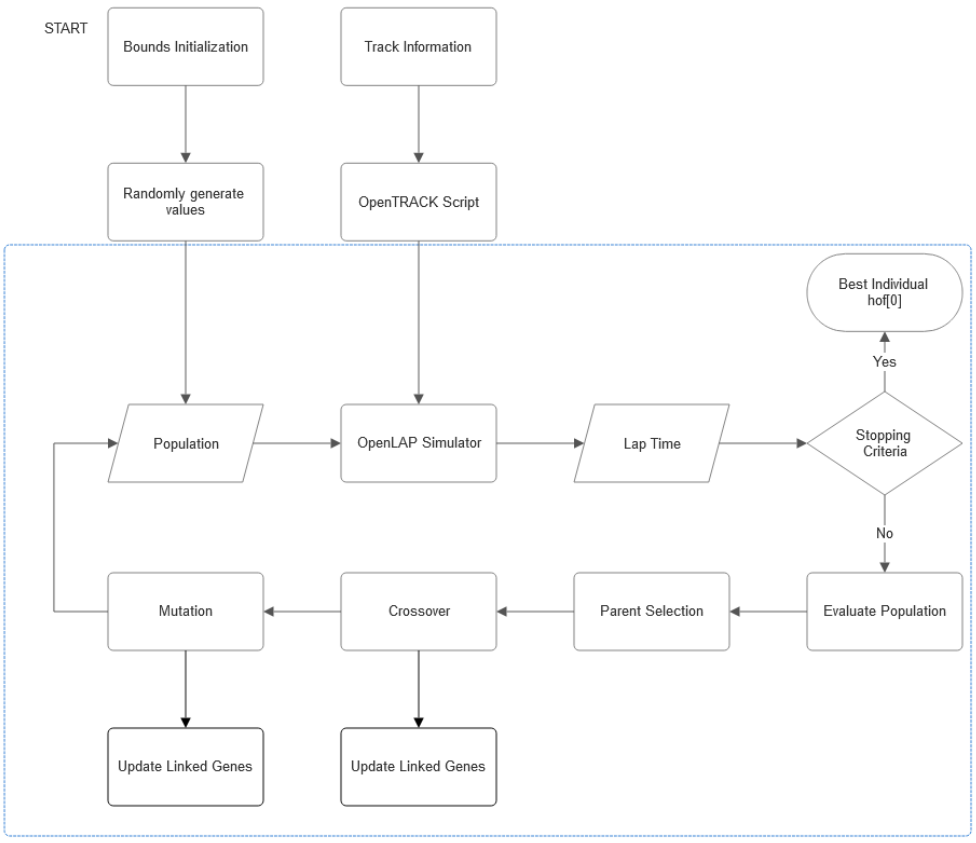


Figure 5.2: Codependent Genetic Algorithm Flowchart

The second algorithm I implemented was based on my own knowledge of Formula 1 to solve a weakness of evolutionary algorithms, my GA in particular. The issue that arises with algorithms like mine is that many of the values are interconnected, proportional to one another, or simply bounded by physics. An easy example of this arises with mass and frontal drag area. No matter what happens with the other values of our setup, a lighter car should almost always be faster than a heavier car, at least since it has a realistic bottom bound of mass. However, in the real world, the implementation of more complex aerodynamic features may make a car faster through the corners but come with a penalty of extra weight. Therefore, I wanted to implement a version of my GA that allowed for some values to be tied together. I made 4 of these values become what I am referring to as ‘codependent’ or what may be understood as ‘linkage’: mass, drag coefficient, lift coefficient, frontal area, and gear ratios. Relevant literature to these concepts can be found in Chapter 3.

Vehicle mass is dependent on other values such as the wheelbase of the car to ensure that as the car gets physically larger or denser, the mass is reflecting this change as well. The drag and lift coefficients are closely connected values in physics, and I decided to select a drag coefficient over a random distribution and make the lift coefficient a multiple of that drag coefficient value. Lastly, the frontal area is given from a formula based on the drag coefficient. The coefficient of drag is closely related to the “slipperiness” of an object through air, and therefore, a car with more frontal area of drag should also have a higher coefficient of drag. This algorithm used a custom crossover and custom mutation function to allow for gene linkage to occur properly. However, all of the computation for crossing and mutating are the same, only adding necessary code to update linked genes when one of them is changed through crossover or mutation, as seen in the Codependent GA flowchart of Figure 5.2.

5.1.3 Evolutionary Strategies

The third algorithm is Evolutionary Strategies. This algorithm is implemented with DEAP’s ‘eaMuCommaLambda’ function with μ of 20 and λ of 100. There is a function which attaches a "Strategy" to each individual. This is created from the initial bounds as described in the previous chapter and can be updated during mutation with a Gaussian distribution. On initialization, however, it uses a uniform distribution between bounds. This algorithm uses ‘ESBlend’ crossover and ‘mutESLogNormal’ mutation. Both of these functions come from DEAP, though I had to make one small modification to the mutation function to avoid an issue it was causing in OpenLAP, likely with gear ratios not being in descending order after some mutations. Worth noting on a major difference between GA and ES is that ‘eaSimple’ takes crossover and mutation probabilities independently, where each is between 0 and 1, but ‘eaMuCommaLambda’ requires the sum of crossover and mutation to be between 0 and 1, both bounds inclusive. This algorithm also utilizes toolbox decorators from DEAP to ensure that strategies remain valid and in the provided bounds, shown in Figure 5.3.

5.1.4 Codependent ES

The changes from GA to Codependent GA are nearly identical to the changes made to the ES to make it Codependent. This was essentially a few extra lines upon individual initialization, crossover, or mutation that ensured that linked gene values changed with each other. This algorithm still used ‘cxESBlend’ and ‘mutESLogNormal’ for crossover and mutation. It also used ‘eaMuCommaLambda’ with a μ value of 20 and λ value of 100. It utilizes an identical strategy checking function, as shown in Figure 5.4.

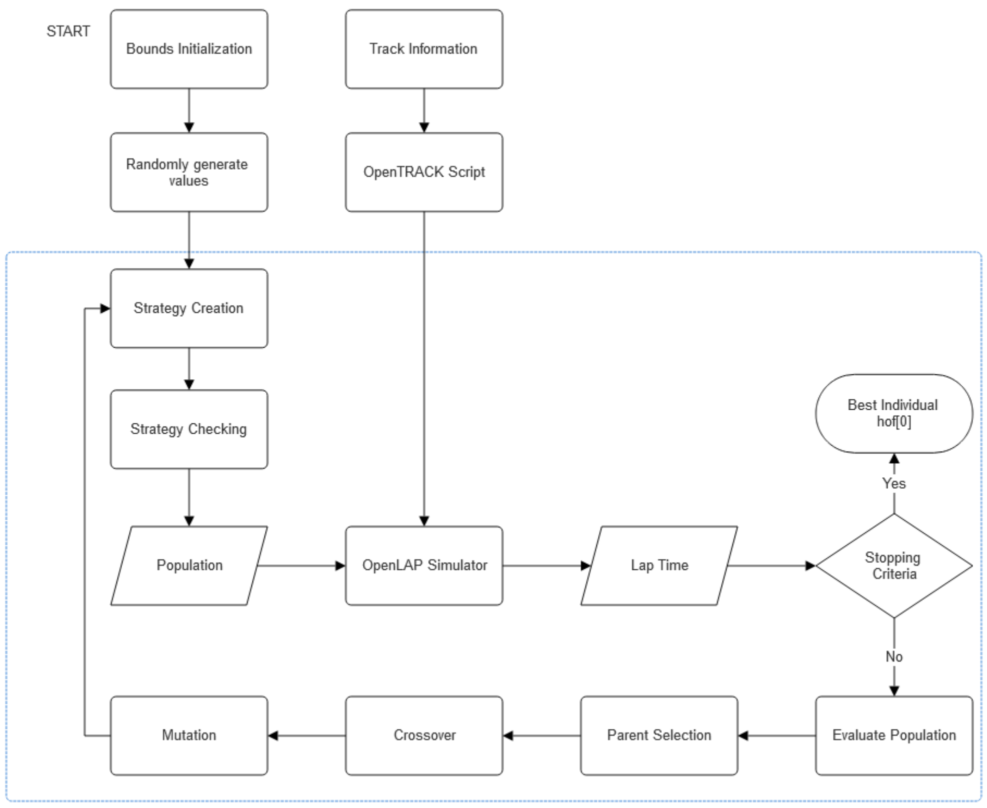


Figure 5.3: Evolutionary Strategies Flowchart

5.1.5 Cultural Algorithm

The last evolutionary algorithm that I wanted to implement was more recently developed than the others I used, and less common in research, a Cultural Algorithm (Reynolds, 1994). My implementation specifically borrows from a GitHub repository by Keith Cissell (Cissell, 2017). This repository was posted as public and did not have a license file included. I wrote nearly all of the code to suit my problem, but used this repository to help construct my understanding of the process and flow of a CA. As you can see from Figure 5.5, the CA is somewhat similar to the GA and ES with the main difference being the focus on belief space evolution rather than direct evolution of individuals. The beginnings are the same, using our given bounds as outlined in the previous chapter to create an individual. This initial population is passed through the OpenLAP simulator which gives each individual its fitness value. We take the best individual from this population to make a situational belief space update. This situational update is mostly used for tracking our best individual, similar to the hall of fame in DEAP. Then we go into the actual generational loop itself by creating our first internal generation for our population. This is using our Normative Belief space, which begins as equivalent to our bounds as discussed in chapter 4. We evaluate fitness on these

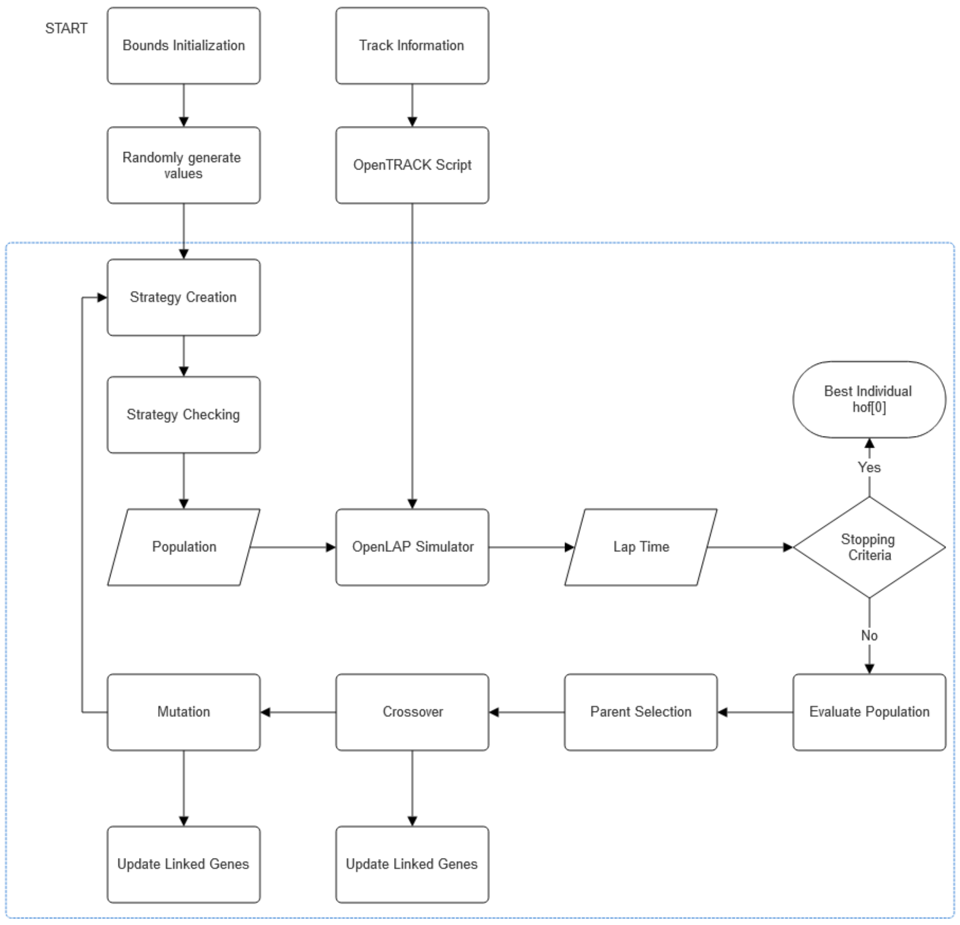


Figure 5.4: Codependent Evolutionary Strategies Flowchart

individuals and use our hyper-parameter of number accepted to perform rank based selection. The best individual updates situational belief space and all of the selected parents are used for normative belief space updating. Then the cycle continues until we reach our stopping criteria. You can see that this algorithm utilizes full replacement, and rather than using crossover of existing individuals to create children, we use selected individuals to perform updates to our population wide belief space, optimizing where future individuals genes may be bounded.

5.1.6 Sequential Quadratic Programming

For my non-evolutionary comparison algorithm, I used the SQP algorithm. It was selected for its wide usage, numerous previous works comparing with evolutionary algorithms, as well as its overall ability to optimize non-linear problems. The details of this algorithm are discussed further in 2.1.2. The overall process of the algorithm is not so dissimilar to the evolutionary algorithms used here. The bounds are identical to all of the EAs, as outlined in the last chapter, and same as the EAs we must begin with a random guess, though we only make one guess rather than an entire population. I created the initial guess by filling each of the 20 modifiable values with a uniformly distributed random value between its upper and lower bounds. I used the **minimize** function from *scipy*, passing in my optimization function, the initial guess, bounds, and the method of choice. Based on my research, I found most people using and recommending the *SLSQP* optimization method for this algorithm, so that is what I used. The objective function is identical to the EAs. One further modification that I made for this algorithm was the use of a callback function to be able to force a certain number of iterations before the algorithm stopped. To implement this, I simply introduced a counter for the number of times that the objective function was calculated, in order to force some equality between the EAs and SQP. Once the number of objective calculations was greater than the limit I set, the algorithm would break upon the completion of the current iteration. I allowed it to finish the current iteration to ensure that I was reaching the algorithm's final decision on the current sub problem and solutions, rather than an intermediate step in the process.

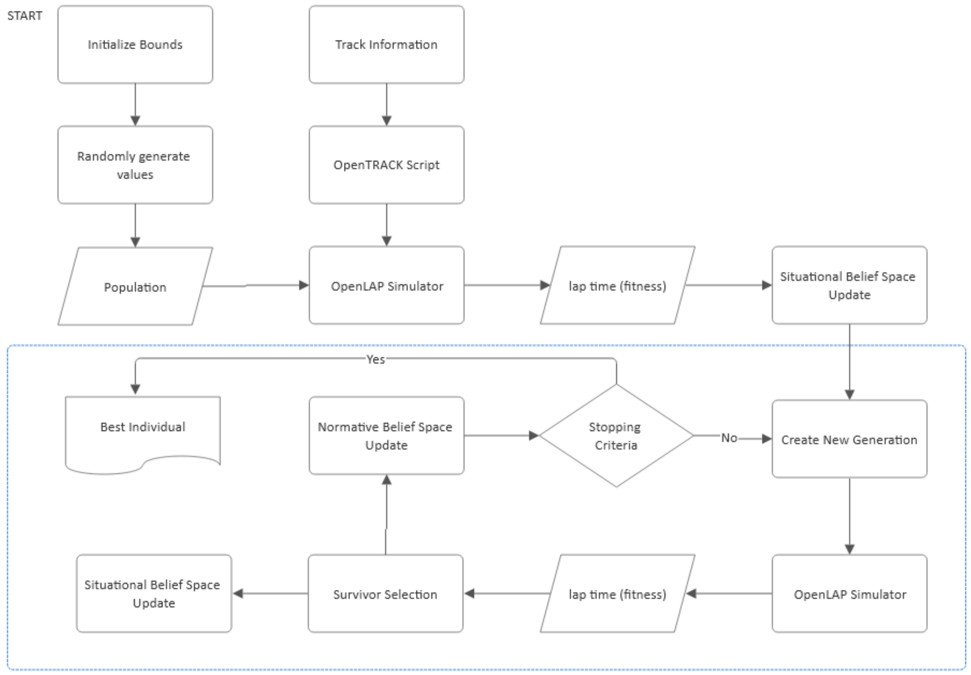


Figure 5.5: Cultural Algorithm Flowchart

5.2 Discussion

For the first 4 car evolution algorithms I am using the DEAP package in Python (De Rainville et al., 2014), as well as the MOO algorithm. The CA does not use any EA focused package. Much of my

Table 5.1: Sample recorded values from 10 GA runs

Run Number	Run Time	Crossover Prob.	Mutation Prob.	Lap Time
1	25:50	1	0.3	98.5324
2	23:27	0.9	0.3	95.2027
3	22:41	0.8	0.3	99.4389
4	22:23	0.9	0.3	94.8207
5	25:51	1	0.3	95.2062
6	24:41	1	0.3	94.8331
7	19:58	0.7	0.3	95.1666
8	18:32	0.7	0.3	94.7850
9	20:45	0.7	0.3	98.1501
10	21:33	0.9	0.3	98.2792
AVG	22:35	0.86	0.22	96.44149

Table 5.2: Sample recorded values from 10 Codependent GA runs

Run Number	Run Time	Crossover Prob.	Mutation Prob.	Lap Time
1	26:05	1	0.3	97.7364
2	23:02	0.9	0.3	103.6574
3	18:32	0.8	0.3	98.1754
4	21:50	0.9	0.3	106.4179
5	21:57	1	0.3	97.5301
6	22:49	1	0.3	98.0036
7	17:16	0.7	0.3	97.5523
8	19:28	0.7	0.3	103.5796
9	15:31	0.7	0.3	97.8338
10	18:24	0.9	0.3	97.3151
AVG	20:29	0.86	0.22	99.78008

implementation was constrained by having to work with the OpenLAP lap time simulator that I am using for my fitness function.

As Hayward discussed in his future work, making a fair algorithmic comparison will be difficult in this domain (Hayward, 2007). So much relies on accuracy of fitness calculations and differences in algorithms themselves make it hard to compare directly. He also mentions the "No free lunch theorem" as further reasoning that a fair comparison may not be possible (Wolpert & Macready, 1995). This is an interesting point to make, but I don't think a comparison needs to be made for the sole purpose of declaring one algorithm type as the definitive best solution for a problem. Rather, I think that by analyzing the advantages, disadvantages, and nuances of different algorithm types, you get a deeper understanding of where certain approaches fall short or outperform others. By learning in this way, you are able to make judgements about what might currently be best, but more importantly, discuss creating something more uniquely constructed to solve a specific problem space in a better way later on.

Table 5.3: Sample recorded values from 10 ES runs

Run Number	Run Time	Crossover Prob.	Mutation Prob.	Lap Time
1	9:12:07	0.7	0.3	87.2375
2	8:34:59	0.7	0.2	89.8709
3	5:54:31	0.8	0.2	88.2099
4	9:39:19	0.9	0.1	91.2522
5	8:32:46	0.6	0.4	89.6609
6	6:07:50	0.5	0.5	88.4108
7	7:44:27	0.7	0.1	91.1924
8	6:17:37	0.8	0.1	90.8786
9	6:32:28	0.6	0.3	86.9102
10	7:25:56	0.6	0.2	90.5024
AVG	7:36:12	0.69	0.24	89.4187

Table 5.4: Sample recorded values from 10 Codependent ES runs

Run Number	Run Time	Crossover Prob.	Mutation Prob.	Lap Time
1	9:20:36	0.7	0.3	92.3808
2	8:03:49	0.7	0.2	94.0906
3	8:36:07	0.8	0.2	97.7497
4	8:38:32	0.9	0.1	100.3189
5	8:43:05	0.6	0.4	96.4791
6	8:43:47	0.5	0.5	101.3509
7	6:38:35	0.7	0.1	98.7306
8	8:06:55	0.8	0.1	102.1619
9	8:37:31	0.6	0.3	98.9442
10	7:32:12	0.6	0.2	93.4354
AVG	8:18:07	0.69	0.24	97.5642

I do believe that I show, as other researchers have, that evolutionary algorithms have potential for significant impact in the automotive and competitive motorsports industries. While Hayward's work was rooted in reality, most or all of the other cited works using EAs for this industry are tied to video games in some capacity. While I understand that this is due to a higher accessibility level for the researchers and others looking to apply their work, I think that it can also detract from the professional usability of that research for a racing team. The OpenLAP simulator has some natural inaccuracy. But the advantage for using this over a video game is that I am not going through an extra interpretation of someone's physics engine. Having also played many racing video games myself, there are very few which are considered to be accurate to the real world behaviour of these vehicles at all. By using a lap simulator rooted in real world physics formulas and calculations, I believe I am able to make a more accurate conclusion about the strengths and weaknesses of these algorithms than one applied to video games. This is not intended to take away from the incredible findings of previous authors, merely to suggest that there needs to be a strong

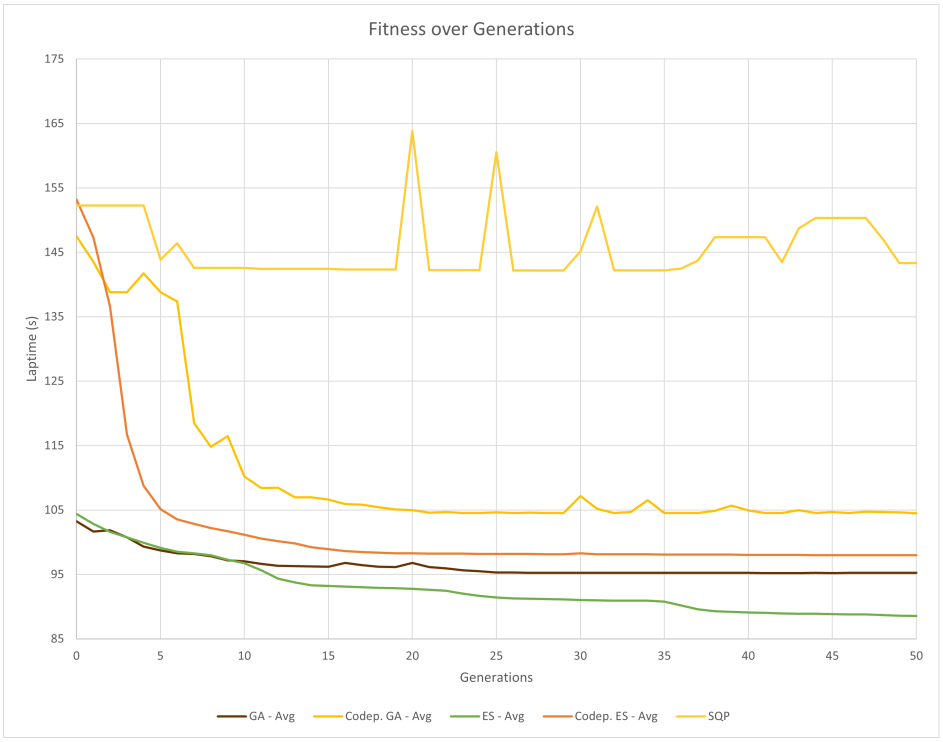


Figure 5.6: Average Fitness of Population over 50 generations
For SQP, sample run with 250 function calculations is used (equivalent to number of evaluations for both GAs).
Since SQP has no population, this is just a single sample run.

differentiation between applications for gaming and applications for real world racing. Racing teams use computer simulations and driver simulations somewhat similar to video games often in competitive motorsports. However, the simulators that these teams are using are far more complex than any video game that is commercially available. That is why I have opted to integrate the lap time simulation into my algorithms, so that this research could be more easily applied to a racing team's simulator for design optimization for their class of racing.

5.2.1 Evolutionary Algorithms Performance

The first thing to observe is our overall lap time gains. As seen in Table 5.3 our ES had one run which performed best with a laptime of 86.971 seconds, which converts to 1:26.971. The fastest time at this years Belgian GP run at the same track was a 1:46.168. We were able to gain an astonishing 20 seconds over a real life F1 car with this algorithm. Obviously, it isn't this simple. Our lap simulator cannot accurately calculate all of the minute details, and the representation of the track doesn't include grip factors or banking or things that naturally occur in small amounts in the real world. While it is easy to bring these things up and throw out conclusions from my lap time improvements here, I still thing it is impressive and worth

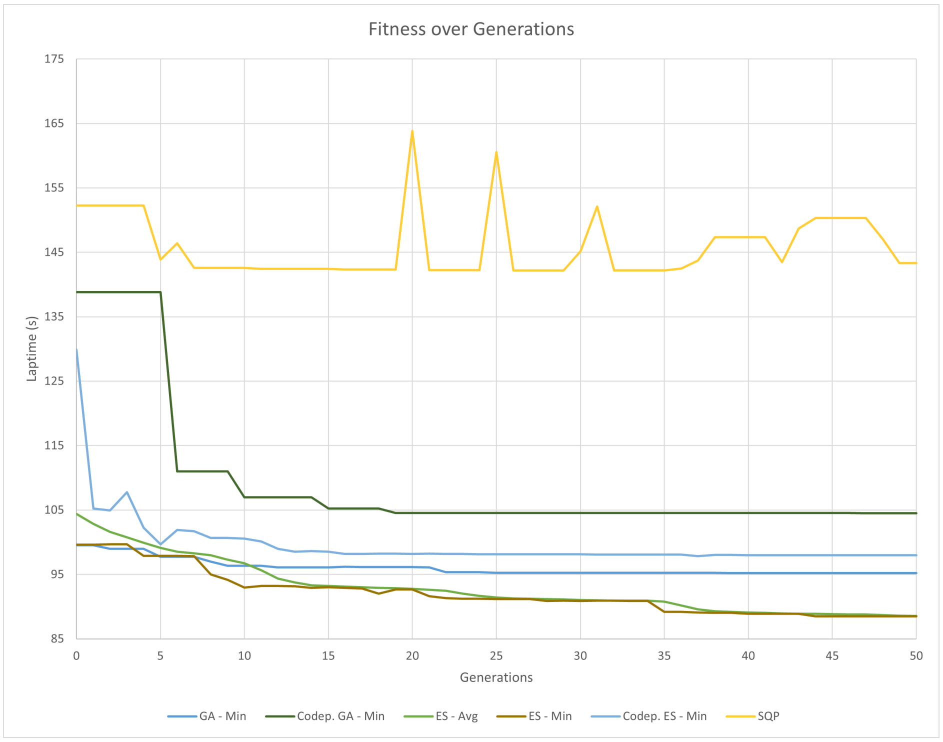


Figure 5.7: Minimum Fitness of Population over 50 generations
For SQP, sample run with 250 function calculations is used (equivalent to number of evaluations for both GAs). Since SQP has no population, this is just a single sample run.

paying attention to a 20 second time savings. For the line plots that are shown here, these runs were performed after and independent from the 10 runs measured for the tables. I took the parameters that had the fastest lap time for each of the 4 algorithms and ran them one time each, recording the best fitness recorded every generation for the same 50 generations as before. Figure 5.6 showcases the average fitness of the population at a given generation count. This allows to see how the population as a whole optimizes over time, where bumps can represent randomization from mutation or crossover to further the coverage of the search space. This is not indicative of the minimum fitness at this point, which is seen instead in Figure 5.7. This showcases the fitness of the best individual in the population at each generation count. The cause of the bumps here are most likely due to mutation changing the best individual, or removing them by losing selection. These two graphs together help to give an understanding of how each algorithm optimizes over a full run. While the comparison to SQP will be discussed more fully in the next section, it is worth noting that the line for SQP is not representative of the mean or minimum fitness at a given point in time. Rather, this is forcing SQP to report current fitness calculation every 5 calculations over 250 total calculations.

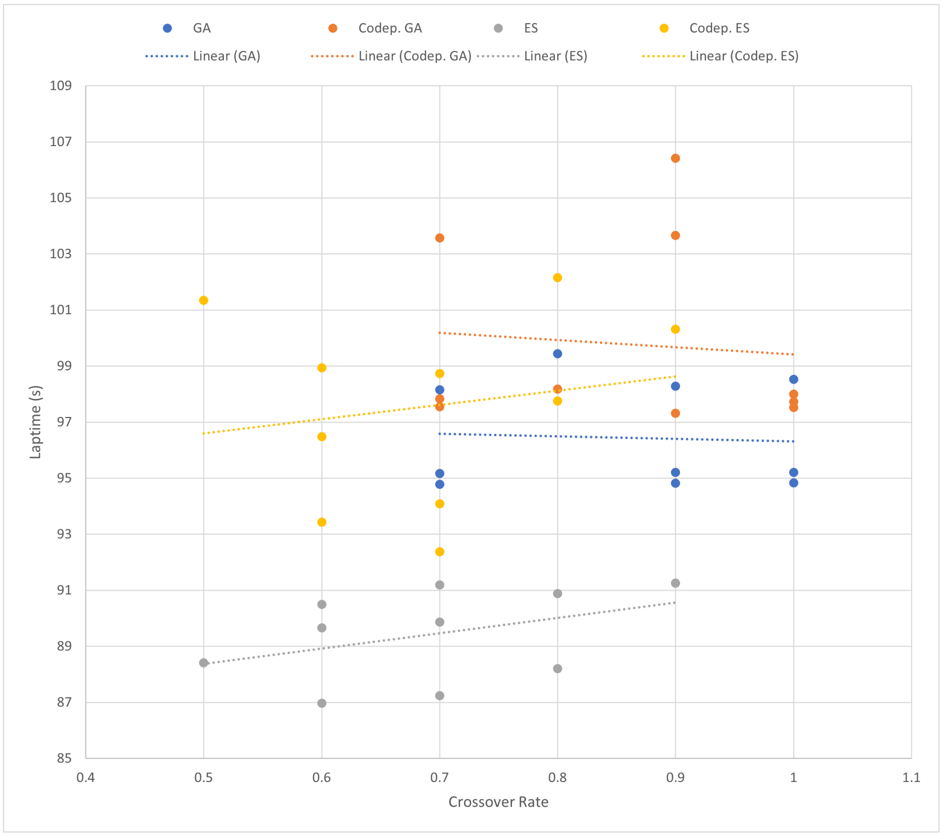


Figure 5.8: Best Fit Line to show the effect of crossover probability on lap time

You can see multiple important features from these graphs. Firstly, you see that it takes longer for the codependent algorithms to optimize, while they start significantly slower and finish around 9 seconds slower than their non-linked counterparts. This indicates that my gene linkage is performing as I hoped by adding initial complexity and decreasing overall speed by staying more true to real world physics. This is also indicated to me by the average fitness of the codependent GA over 50 generations. It continues to explore outside of current individuals through mutation, but the fitness gets worse before returning to where it was before. This shows to me that the algorithm is forming an association between genes like mass hampering performance, desiring to lower it, but finding that there are other factors tied into mass also, requiring it to strike a balance. The second finding is that the GA is faster than the ES for linked and non-linked to converge to its optima, while also having faster end result lap times also. Of the sampled 10 runs, the GAs had average best lap times of 96.44 and 99.78, seen in Tables 5.1 and 5.2. The ES algorithms had average best lap times of 99.22 and 106.98, as shown in Tables 5.3 and 5.4 respectively. Due to using replacement based on fitness rather than the crossover and mutation full replacement, ES never worsens its average fitness, excluding from generation 0 to generation 1 in the codependent algorithm.

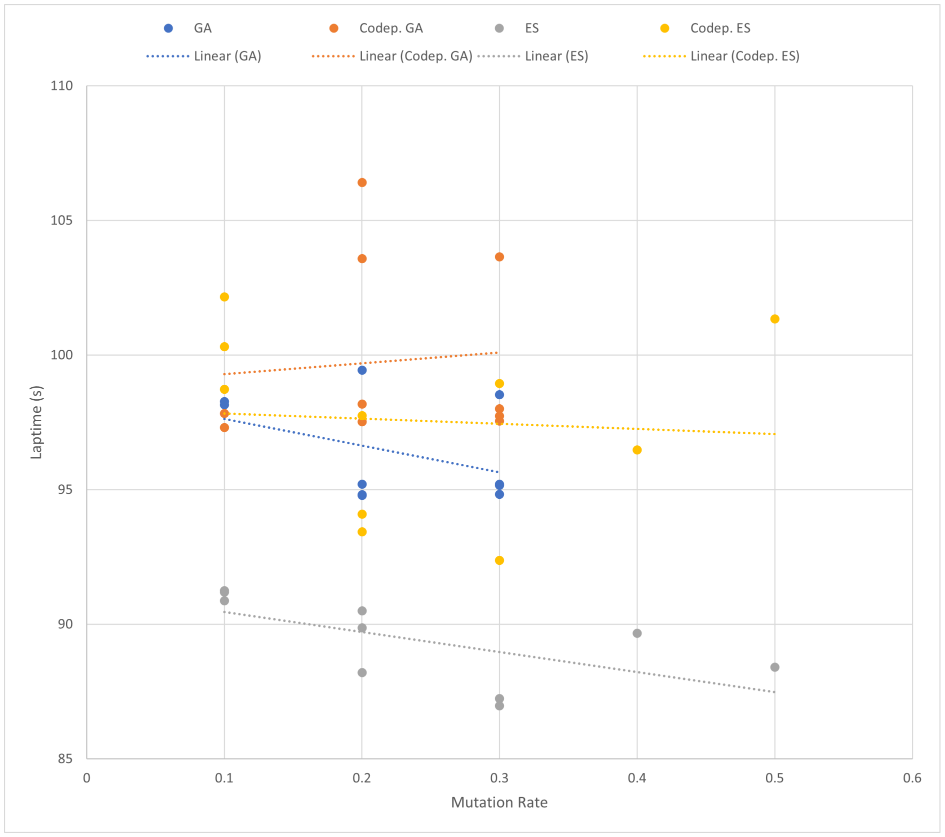


Figure 5.9: Best Fit Line to show the effect of mutation probability on lap time

Moving to the scatter plots, these are intended to look for correlations between hyper-parameters like crossover and mutation probability with the resulting lap time, as well as seeing if there is a relationship between runs that took longer and resulting lap times. The points are results of the runs shown in the preceding tables, where the dashed line is the line of best fit for each algorithm. This line of best fit is intended to show whether there is a strong effect between any hyperparameters and the resulting laptimes.

In the first plot of this kind, Figure 5.8, there does not seem to be a very strong connection between crossover and performance, but it is interesting that the slopes for GA and ES have opposing signs. The GA slightly prefers higher crossover where the ES slightly prefers crossover closer to 0.5. I am also encouraged that the slopes between the two GAs and the two ESs are very near to parallel. While this is a minor thing to check, it adds further evidence that my gene linkage is not inhibiting the characteristics of the algorithm, merely adding some extra realism.

The mutation lines in Figure 5.9 are less consistent, and while they also doesn't seem to have a linear relationship to lap time, they seem to have a slightly stronger effect than crossover. As with crossover, the ES algorithms prefer mutation probabilities closer to 0.5. The GAs on the other hand have nearly opposite best fit lines. The standard GA prefers higher mutation probability where the codependent GA prefers

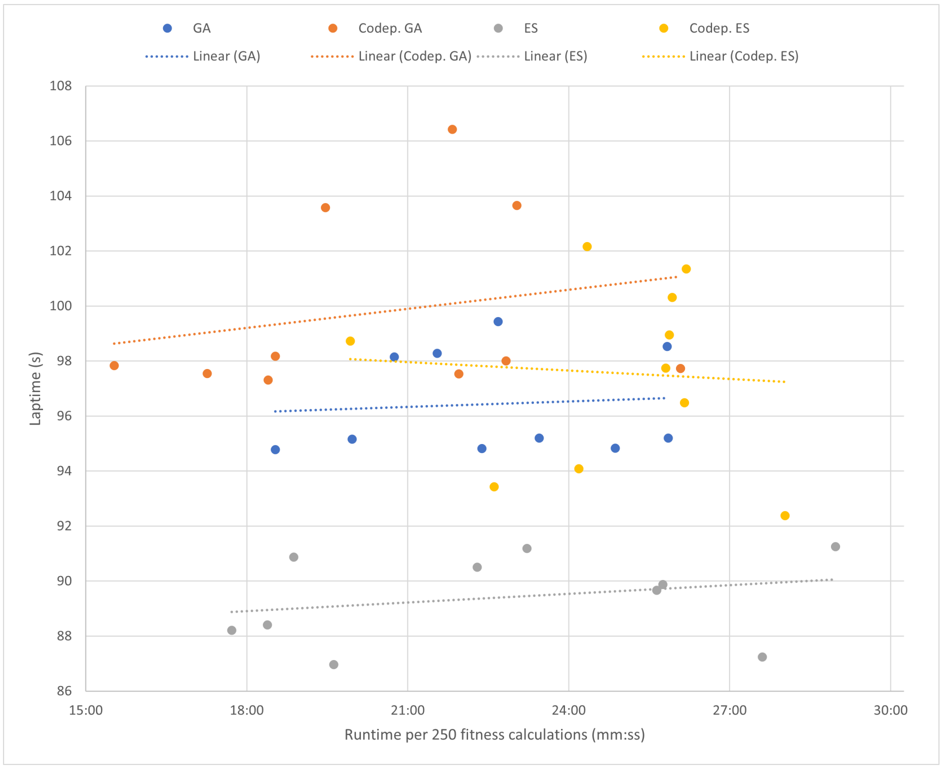


Figure 5.10: Best Fit Line to show the effect of run time on lap time
For both ESs, the runtime is divided by 20 since they have 20 times the number of function evaluations. True runtimes are shown in 5.3 and 5.4

lower mutation rates. It is interesting that they have the same relationship with crossover but not with mutation. I believe this is due to the nature of the gene linkage. If a gene is changed during mutation it will have a stronger change to linked genes than if that value is changed during crossover. This could be due to built in designs of these functions, but I also think it is due to the roles of these operators. Crossover is taking selected parents and combining them to make children, where mutation is simply changing a gene at a time based on some bounds. In this way, genes between selected parents are likely to be more similar than gene values before and after mutation. Therefore, I do not think that this plot weakens my linkage, but may further strengthen it by showing that the algorithm is finding linkage to be an inhibiting factor on achieving the fastest lap times.

With the last plot in Figure 5.10, I wanted to see if there was a correlation between algorithms requiring more compute complexity (therefore having a longer run time) and having faster resulting lap times. What I found was nearly the opposite. The best performing runs, though only slightly, were the faster running versions in 3 out of 4. In the codependent ES, there is a very strong relationship between run time and lap time. Based on the best fit line, for every extra 3 minutes that the algorithm takes, it is approximately 7.5 seconds faster around the track. I don't believe however that this means that better laptimes are achieved

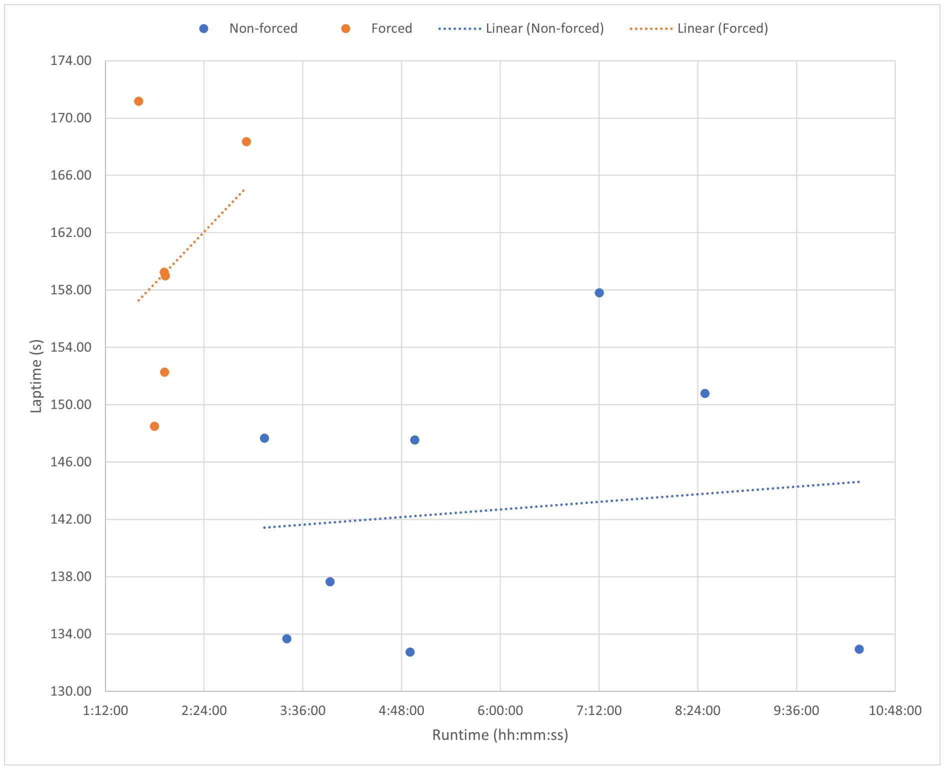


Figure 5.11: Best Fit Line to show the effect of run time on lap time for the SQP Algorithm

through forcing shorter runs. Rather, I believe this to be a reflection of luck. If an algorithm's original random guess population is closer to an optima, it will simply converge faster than originally slower and possibly more diverse populations, rather than anything indicative of algorithmic performance traits. There is also a plot showing the effect of runtime on the SQP algorithm in Figure 5.11, which shows similar correlations to the EAs. This is likely indicative of luck on initialization again, similar to the EAs.

Based on the testing runs recorded in the first four tables, as well as previous testing, I elected to use the codependent GA as the chosen algorithm for multi-objective optimization (MOO) across different lap times. Over these final 40 runs, the ES did achieve faster lap times than both GAs on average, it is significantly slower, though this is due to number of fitness calculations and population size. In testing a GA up to 5000 calculations (the same as the ESs), it did not manage to match the performance of the ES with laptimes in the 94 second range at best still. Similarly though, the ES run at smaller population and larger with generations modified to be around 250 calculations (same as the GAs), it achieved lap times in the upper 90s or low 100s, while being about 10 minutes slower than the GA. Therefore, it seems that for a start of season testing to understand the car and regulations, an ES would outperform the GA in the long run. But, if needing to make setup changes between practice sessions or before a race with tighter time constraints, the GAs will perform better. I elected to use the Codependent GA, therefore, because

Table 5.5: Sample recorded values from 5 CA runs

Run Number	Run Time	Pop. Size	Num. Accepted	Generations	Lap Time
1	2:28:20	75	15	15	97.2255
2	2:37:10	50	10	25	97.5412
3	0:30:34	10	2	25	100.9161
4	0:46:35	15	3	25	100.8440
5	2:33:44	15	3	75	98.3117
AVG	1:47:16	33	6.6	33	98.9677

Table 5.6: Sample recorded values from 5 MOO runs

Run Num.	Run Time	Pop. Size	Gens	Cx Prob.	Mut. Prob.	Best Spa Lap	Best Monza Lap
1	3:55	5	5	0.9	0.1	109.5033	80.5471
2	41:21	5	50	0.7	0.3	104.5153	76.8704
3	18:05:15	5	25	0.8	0.2	104.1582	76.4013
AVG	22:38	5	26.66	0.8	0.2	106.0589	77.939

I wanted the MOO to be more efficient for smaller runs since it required frequent modification of the weights and other factors in development.

While all of the previous testing runs were recording over the provided OpenTRACK file for Spa-Francorchamps, the MOO algorithm will be run over two tracks with equal weights for all of their lap times. These two tracks are: Circuit de Spa-Francorchamps and Autodromo Nazionale Monza. Both of these track files are provided in the OpenLAP repository (Chalkiopoulos, 2020). Everything else about the algorithm has remained exactly the same. My first run for MOO was run with crossover probability of 0.9 and mutation probability of 0.1 because these were the hyper-parameters that the single objective codependent GA performed best on. My first run for the MOO algorithm also used a population size of 5 and 50 generations as with the original runs. After that I did two other test runs to get a little bit of an understanding how the hyper-parameters would affect our results. In the end, we were able to achieve a best result of a 104.1582 at Spa-Francorchamps and 76.4013 at Monza, shown in Table 5.6. These are very competitive times compared to my testing with the OpenLAP simulator and the other algorithms here. I think this certainly proves the viability of expanding any of my algorithms into MOO over multiple track simulations.

For the cultural algorithm, it did not perform the way that I hoped, results can be found in Figures 5.12 through 5.16 and Table 5.5. However, I did learn about this style of algorithm enough to still believe in some future viability. In the code that I adapted for this problem, the algorithm used a population size of 50 and a generation count of 50. During testing and development, I quickly realized that these parameters would not be a good fit for my algorithm. The largest reason is compute complexity, with a secondary reason being more intricate problems of search space exploration. Compared to the initial four algorithms, the cultural algorithm took around 4 times as long to run on average. The standard GA averaged around 25 minutes for a run, where the CA averaged almost 1 hour and 50 minutes, also including 2 runs with much smaller populations and generations to test faster runs. This combined with an average lap time of 98.96, with a best lap time of 97.2255, simply doesn't make the CA seem worth running for so much longer. The search space exploration is another difficulty for this algorithm. I think the reason that bigger populations perform better is because it nearly guarantees bigger bounds to evolve within. The number accepted is scaled based on population size, so with fewer individuals, we have less information to update our bounds. This leads to a narrower search space and greater difficulty to converge to global optima. Also evident from the fitness over generations of all 5 runs, there is not much optimization actually occurring compared to the codependent algorithms at least. In Figure 5.15 you can see that only one better individual was even found. This run was so uneventful that the axes of the graph were even messed up, deciding to show the decimal point second improvement from the first couple of generations to every other, though the resulting lap time was still over 100 seconds. I do not, however, consider developing this algorithm fruitless labor. If anything, seeing these trouble spots further solidified my opinions expressed earlier that a fully fledged CA has the potential to be even more advanced and beneficial to a real life racing team than any of the other algorithms here. This will be discussed more fully in the future work section of the final chapter.

While these results seem very promising, it is important to ensure that they, and my opinions of their success, are not occurring in a vacuum. In order to test this, we will need a different form of algorithm to compare to. As mentioned a few times earlier, I am using the SQP algorithm to compare to the evolutionary algorithms due to its iterative nature, handling of constraints, and existing literature also using it as a benchmark for EAs.

5.2.2 Comparison to SQP

The factors which I will be comparing between these algorithm types are lap times, runtime and efficiency, as well as future possibilities and other considerations that arose during my development and testing time.

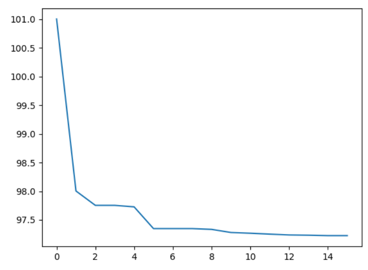


Figure 5.12: CA run 1 fitness over generations

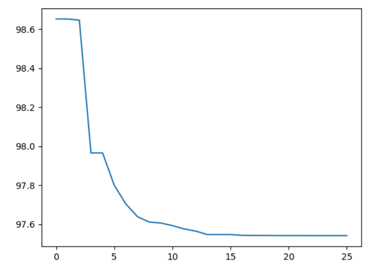


Figure 5.13: CA run 2 fitness over generations

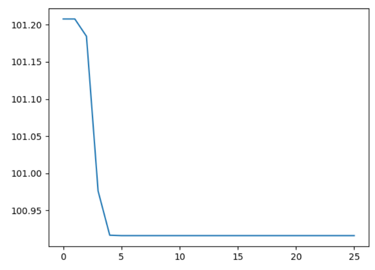


Figure 5.14: CA run 3 fitness over generations

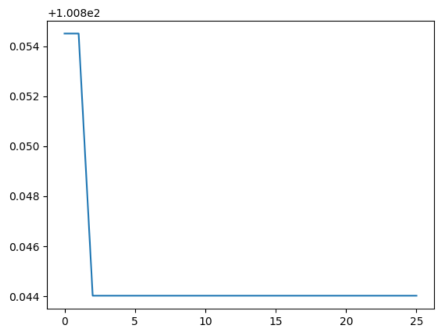


Figure 5.15: CA run 4 fitness over generations

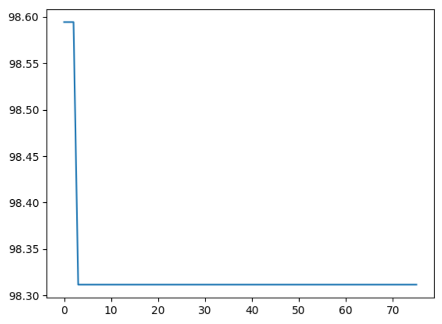


Figure 5.16: CA run 5 fitness over generations

Table 5.7: SQP run data for runs with no forced function evaluation count

Run Num.	Run Time	Func. Evals	Laptime	Reason for Stopping
1	4:57:41	697	147.53	Optimization successful
2	4:54:15	673	132.76	Optimization successful
3	3:08:04	413	147.66	Optimization successful
4	10:21:41	1528	132.93	Constraints Incompatible
5	8:29:10	1166	150.8	Constraints Incompatible
6	3:24:23	506	133.68	Constraints Incompatible
7	7:12:13	1104	157.8	Constraints Incompatible
8	3:55:51	536	137.64	Optimization successful
AVG	5:47:55	827.875	142.60125	

First, we consider the lap time results, the overall success of the optimization for our objective function. The best lap time achieved by SQP was a 132.76 on non-forced run 4 (seen in Table 5.7, with the best forced iteration run that reached its limit was 138.85 on run 10 (seen in Table 5.8. The non-forced runs had an average lap time of 142.6, forced had an average of 151.74, giving an overall average of 147.68. For comparison, all of the evolutionary algorithms (both GAs, both ESs, CA, and MOO) combined had an average of 96.77, approximately 35% faster than SQP.

Next we consider the runtime and efficiency where again, the EAs perform better, though less perhaps clearly than on runtime. Beginning with GAs, we have an average run time of 0:21:32, with ESs averaging 7:57:09. CA and MOO average 1:47:17 and 0:21:19 respectively. The SQP algorithms, forced and non-forced iterations, combine for an average of 6:14:57. However, the full forced 5000 calculations run is a large outlier at 39:08:09, which when removed brings the SQP average to 4:18:52. For a closer comparison, consider only runs with similar counts of objective functions. Both GAs have 250 objective function calculations and average 0:21:32. The forced stop SQP runs with a limit of 250 have an average run time

Table 5.8: SQP run data for runs with forced function evaluation count

Run Num.	Run Time	Func. Evals	Laptime	Reason for Stopping
1	1:47:47	271	148.49	Limit Exceeded - 250
2	1:36:10	258	171.17	Limit Exceeded - 250
3	1:55:39	270	158.98	Limit Exceeded - 250
4	1:54:59	255	159.25	Limit Exceeded - 250
5	2:54:59	269	168.36	Limit Exceeded - 250
6	1:55:20	266	152.27	Limit Exceeded - 250
7	4:13:43	580	133.35	Constraints Incompatible
8	3:30:59	524	137.47	Constraints - Incompatible
9	7:07:57	1026	149.2	Terminated early
10	39:08:08	5013	138.85	Limit Exceeded - 5000
AVG	6:36:34	873.2	151.7397	

of 2:00:49, around 82% longer than GA. Similarly, the ESs average 7:57:09 with 5000 fitness function calculations, where the forced SQP run with a limit of 5000 has a runtime of 39:08:09, around 80% slower. SQP is longer to run on average than both CA and MOO as well, though they have varying population and generation counts between runs which leads to much higher variance on run time. All of this shows that EAs are more successful in less time than SQP. In real world terms, this means that if a race team were trying to find a more optimal setup between two practice sessions, a change in weather, or some other time constraint, an EA would allow for significantly larger runs with more fitness calculations and a much faster lap time than SQP.

There are a few considerations to make still to understand this situation better. Firstly, the self-reported success of the algorithm. An EA will always converge to a solution and give the final found solution as the "hall of fame", or best, result. SQP, on the other hand, does not guarantee a solution by default. In 5.7, you see that 4 of the 8 runs optimized successfully, where the other half quit optimization for constraint issues. I forced the algorithm to give me the latest solution upon termination, where as standard it would not have given me results if it did not consider itself to have found a converged, optimal solution. While this doesn't necessarily change what you might do with the results, as it still reports a lap time and the parameter values in my version, I didn't find it very confidence inspiring to use results that even the algorithm itself was not satisfied with. EAs, on the other hand, will always converge to a maxima or minima, in my case dealing with minimization. This is not guaranteed to find a global optima, but it is designed at least to maintain similarity to its best individuals with the designs of these algorithms. Again, framing this into the real world, I can't imagine a racing team getting results that an algorithm self-reports as not optimized, yet implementing them on a car that could cost millions of dollars, being driven by a real person on the limit for multiple hours. Again, nothing about its results are inherently unsafe, but I would not have much confidence in results that are openly considered to be "incompatible" with the constraints.

This does possibly raise some questions about the constraints, which is the second important consideration to make in this algorithm comparison. As shown in chapter 4, some of the bounds for vehicle parameters are very small, notably brake pad height, which varies from 52.0 to 52.8 mm, as well as a few other very restrictive bounds. In my development of the SQP algorithm for this paper, I started with wider bounds to ensure that the algorithm converged easily so that I could learn output formats and other factors before the final version. I made two changes to the bounds for these tests, I increased brake pad height bounds to 50-55, and I made gear ratios ten times higher, for example 22 to 30 for gear 1 instead of 2.21 to 3. I would then divide the gear ratios back down before fitness calculation. In one testing run with these modified bounds, SQP successfully optimized in 49 minutes with 504 function evaluations and a resulting lap time of 93.58. SQP is a very widely studied and used type of algorithm, and this is not a discussion about whether EAs are better than SQP for all problem sets, only if EAs are better for this specific problem set. It might be easy to say that the bounds are too restrictive and that they should be widened to allow the algorithms to converge more easily. However, that is what makes this problem space so difficult, and that is what makes the EAs performance even more impressive than SQP. The EAs worked under the same bounds as a real life engineer and outperformed SQP. Widening the bounds for an algorithm cannot be considered a viable solution, as the bounds are not widened for real racing engineers. Though, I did think this was an important inclusion as to another reason why EAs are better suited for this space. SQP did not optimize well under the very restrictive bounds, and EAs did. This applies even more so when considering the future applications of work like this.

As will be discussed more in the final chapter of this paper, there are many ways to improve this work in the future. Nearly every improvement to make is focused on adding more realism to the lap simulator and results of the algorithms. Ensuring that the algorithms have a very strong correlation to the real world is critical to any teams using them for their real setups. As you approach reality, there will be even more bounds and even tighter bounds than those already seen here. The FIA's documentation about the rules for F1 and many other racing series cars are very specific about every single part of the car. Moving forward, I don't know that there is much more to be done to use SQP in this way. It is possible you could use it to optimize very specific parts of the car in smaller batches or similar, but it doesn't seem viable when it isn't guaranteed to optimize to a solution it doesn't like, especially as bounds tighten moving forward.

Chapter 6

Evolution of an F1 Track - Experiments and Discussion

6.1 Experiments

As an exploration of further opportunities to apply Evolutionary Computing to the world of motorsports, I generated a Formula 1 race track using a Genetic Algorithm. This process has been applied to the world of procedural content generation for video games (Togelius et al., 2006), (Loiacono et al., 2011), as was discussed in Chapter 3. However, I have not seen any research about using GA's for track generation in the real world. I see many potential uses for this technology in this domain. There are two types of racetracks: permanent and street circuits. Permanent tracks are what you typically picture for a racetrack. It is in a more isolated place with a dedicated track layout and permanent grandstands and other features. A street circuit, on the other hand, is a race track that is designed to work with existing roads and structures. These can be found all over the world in Australia, Miami, Mexico City, and with the newest addition to Formula 1, the Las Vegas Grand Prix.

Each type of track comes with its own unique set of challenges. For a permanent track, you are only constrained by your creativity, along with some basic guidelines set by the FIA for a Formula 1 track. Track designers like Hermann Tilke have amassed a reputation for creating tracks that are enjoyable to both drivers and fans. But with this freedom comes increased challenge of forming something from nothing and ensuring that it reaches a standard to be a spectacle of racing and an economic bolster to Formula 1 and its growing fan base. For a street circuit, the challenge is how to create an exciting track with what you already have. This often requires massive construction projects, repaving of roads, and developing a temporary grandstand and amenities for attendees of the event.

As it is with designing a Formula 1 car, it is possible that the best track designer is still an experienced engineer with a pencil. However, I believe that there is merit to my application, as well as those used for video games, to be able to solve the challenges posed by both types of racing tracks. In designing a permanent circuit, you can design a fitness function specifically enough that the algorithm creates your subjectively "perfect" track. For street circuits, I think it could be possible to create an algorithm that runs this same evolution over possible track designs, but with the constraints that it must use the existing roads that you have. Perhaps the algorithm is good enough on its own, or perhaps it is a financial and time saver for the engineering team. Either way, I think there is enough evidence of potential uses in this field.

6.1.1 TSP Bezier Spline Track GA

The first track optimization algorithm is the one largely based on Togelius et al., 2006 and Loiacono et al., 2011 as mentioned before. It uses a set of randomly generated points and a travelling salesman problem (TSP) brute-force solver to create outer control points with heuristically calculated internal control points to construct segments of Bezier curves. By combining these segments at overlapping control points, we create our B-spline representation as seen in the plots of the sample tracks generated in the 10 test runs. The algorithm is not optimizing anything to do with the segments and the appearance of the curves themselves directly, as it is in the second algorithm. Instead, all the algorithm is doing is evolving a list of coordinates. Every time a coordinate is changed, the TSP must be recalculated to avoid the track not closing or major track intersections. This algorithm uses tournament selection with a tournament size of 2. For crossover and mutation it uses One Point Crossover and Bounded Tuple mutation. The fitness function of this algorithm is odd, but is intended to contrast the high compute complexity of the second algorithm. This algorithm checks against a number of features that the FIA outlines as requirements for a track to host a Formula 1 event, which can be found in Appendix O of "International Sporting Regulations - FIA", 2023. The ones I have included in this fitness calculation are as follows: the maximum track length is less than 7 km and no segment defined as a 'straight' is longer than 2 km. The other calculations used are for avoiding control points being too close together, turns being too sharp for an F1 car, and the curvature profile similar to that defined in Loiacono et al., 2011. Since this algorithm is less rooted on characteristics of the segments and more on layouts of control points, I had to use semi-arbitrary definitions for these fitness values. I used a conversion of the euclidean distance to represent track length and straight length. I also chose to define a turn of 30 degrees or sharper to be too tight for an F1 car based on some estimation from track maps of current F1 tracks. I then added or subtracted the values from these calculations to a single value for fitness. In this way it is similar to Multi-Objective Optimization as used in the second algorithm, I even have ways of weighting each value differently. However, this is part of what gives this algorithm much faster run times than the MOO OpenTRACK GA. As with the car algorithms, all run time measurements are in minutes and seconds or hours, minutes, and seconds in both of these track algorithms. Results of this algorithm can be seen in Figures 6.1 through 6.10

6.1.2 OpenTRACK Multi-Objective Track GA

The OpenLAP Laptime Simulator used in the last chapter for car evolution was also my second implementation for track evolution. This project contains a file called OpenTRACK that was discussed earlier and was the basis for this genetic algorithm. The OpenTRACK script takes a list of segments containing three data points. The first is the type of segment, with the options 'Straight', 'Right', and 'Left'. The second data point for a segment is the segment length in meters. Finally, the third point is the radius of curvature of that segment also in meters. This list of segments was what I used as my representation for my multi-objective genetic algorithm.

Each individual is created through a mostly random process to ensure lack of bias and algorithm control. First, we set a segment counter for each individual as a random integer between 50 and 100. These bounds were found through testing for best results. A segment, in this case, is a group of pieces that all have the same curve type. Each segment contains a random number of pieces from 6 to 10, these bounds also found from testing and comparing to the sample tracks.

This algorithm uses tournament selection with a tournament size of 2. It uses one point crossover to try and keep segments together as well as bounded tuple mutation to encourage diversity with bounds of -150 to 150, inclusive. I used varying hyper-parameters in the testing runs of this algorithm, changing crossover probability, mutation probability, population size, and number of generations. I logged all of these values, the run time for each run, the number of individuals in the Pareto frontier, and the fitness values of my subjectively preferred track. The images of all of these tracks are included throughout this chapter as well, Figures 6.11 through 6.20.

Table 6.1: Logged Parameters and Output from B-Spline GA runs

Run Number	Run Time	Pop. Size	Generations	Crossover Prob.	Mutation Prob.	Fitness
1	0:14	5	20	0.7	0.2	37.7447
2	0:17	5	20	0.7	0.2	49.5565
3	0:32	5	50	0.7	0.2	50.3139
4	1:52	15	50	0.7	0.2	58.1281
5	2:01	10	75	0.7	0.2	58.9482
6	2:02	15	50	0.9	0.1	48.2629
7	2:11	15	50	0.8	0.2	58.1929
8	2:12	15	50	0.6	0.4	56.1777
9	4:20	30	50	0.6	0.4	57.4368
10	4:28	30	50	0.7	0.3	58.8602
AVG	2:01	14.5	46.5	0.71	0.25	53.3622

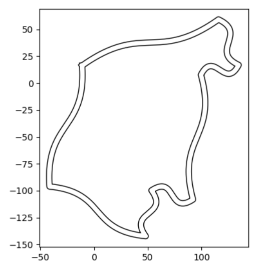


Figure 6.1: B-Spline GA Run 1

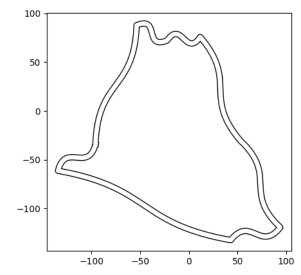


Figure 6.2: B-Spline GA Run 2

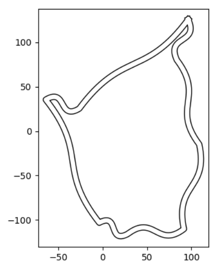


Figure 6.3: B-Spline GA Run 3

Table 6.2: Logged Parameters and Output from OpenTRACK GA runs

Run Number	Run Time	Pop. Size	Generations	Crossover Prob.	Mutation Prob.	Num. Pareto
1	3:55	5	20	0.7	0.2	4
2	5:00	5	20	0.7	0.2	3
3	9:08	5	50	0.7	0.2	4
4	17:38	15	50	0.7	0.2	12
5	11:51	10	75	0.7	0.2	1
6	13:46	15	50	0.9	0.1	1
7	15:09	15	50	0.8	0.2	1
8	25:23	15	50	0.6	0.4	6
9	29:20	30	50	0.6	0.4	2
10	24:58	30	50	0.7	0.3	1
AVG	15:37	11.5	46.5	0.7	0.26	3.5

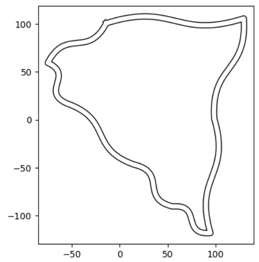


Figure 6.4: B-Spline GA Run 4

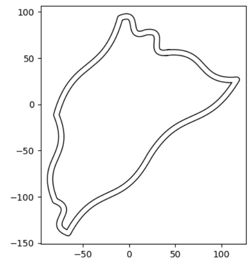


Figure 6.5: B-Spline GA Run 5

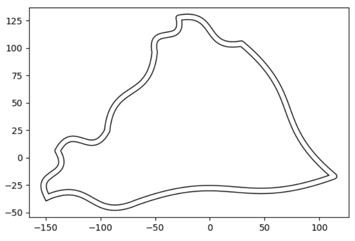


Figure 6.6: B-Spline GA Run 6

6.2 Discussion

Comparing between these 2, it seems fairly obvious that the B-Spline algorithm has better outputs in its current configuration, its results shown in Table 6.1. The algorithm was not able to converge to what I was hoping in the OpenTRACK GA, but I think this is a design error of mine somewhere. I am very pleased with B-Spline outputs, and I actually think that some of them would make very good F1 tracks exactly as they are. However tracks like Run 7 are evidence of some remaining issues. But tracks 1 through 5 or maybe even 6 show serious promise. As it was with the car evolution algorithms, I think the biggest advantage and distinction of my work over others prior is in the usability for the real world. Since this algorithm relies entirely on coordinates, I think there are very critical applications that could be created from this. The most impactful of these will be mentioned again in the future work of the last chapter, but it would be in a city providing a map of their streets and using an algorithm like mine to find the best possible circuit between them. While my algorithm as it is wouldn't work for this, it does not seem

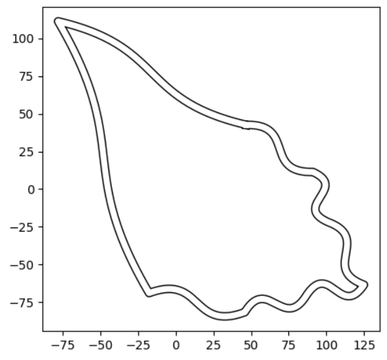


Figure 6.7: B-Spline GA Run 7

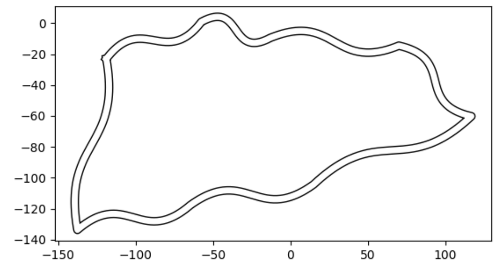


Figure 6.8: B-Spline GA Run 8

very far off if you set some different heuristics for control points than I used here. For permanent circuits, this algorithm could work exactly as it is. Apply a simple scaling calculation to make the track and each segment the length that you desire by multiplying the coordinates and then work to add different track features like gravel traps, run off areas, etc. Regardless, I think these examples show that my algorithm is able to take a basic list of coordinates and create a suitable F1 track in only a couple of minutes. Example tracks from this algorithm are seen in Figures 6.1 through 6.10

For the OpenTRACK GA however, the results are not what I was hoping for, though the data in Table 6.2 does not immediately seem concerning. In its current state, this is the best it has to offer. Given the noticeably longer run times compared to the other algorithm and the fact that this algorithm isn't able to close the track, and it is clearly inferior to the first algorithm, as seen in Figures 6.11 through 6.20. However the main reason I wanted to attempt developing this algorithm is that I see a possible intersection between a car generation and track generation algorithm. You develop your car for a sample of existing tracks and discover what parts or features of a track it struggles with aerodynamically. Then, you could go out and generate tracks that have a higher concentration of these types of features to improve your simulation. In this way, you can combine the two types of algorithms I developed for this research and end up with an even faster car and perhaps a more interesting track design along the way. However, the results are not there at all yet. The representation of this track is much more connected but also more abstract. The algorithm is having a harder time learning to make incremental, minute changes to segments to make them a section of sweeping corners or a tight hairpin turn. Rather, they make a semi-straight curve that goes off into the distance, sometimes crossing over itself, and with less defined turn structures than the B-Spline tracks. Track 4 of the OpenTRACK outputs is the only one that I would say is the beginnings of a track. Even still, it has maybe 5 defined turns, the road is never straight, and the turns it does have aren't very interesting. I think there is plenty of room for improvement here. I think the biggest limitation

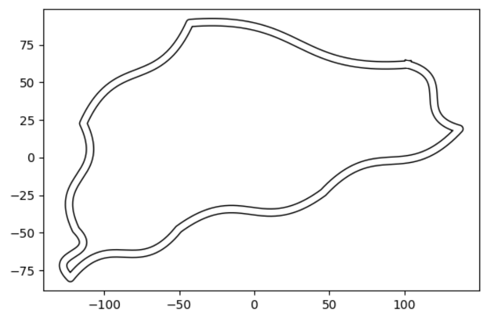


Figure 6.9: B-Spline GA Run 9

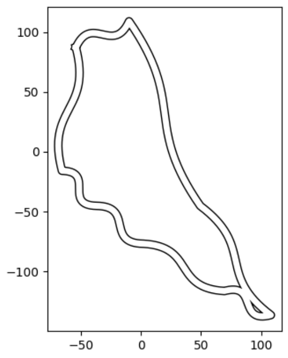


Figure 6.10: B-Spline GA Run 10

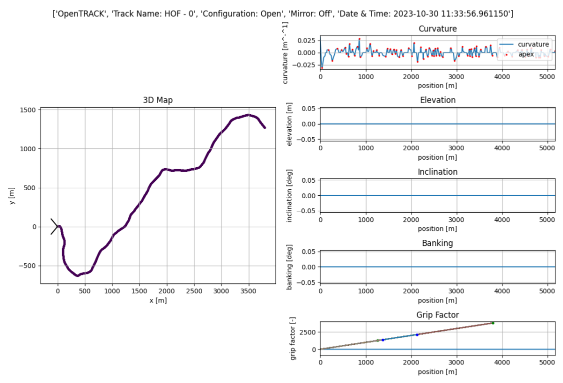


Figure 6.11: OpenTRACK GA Run 1

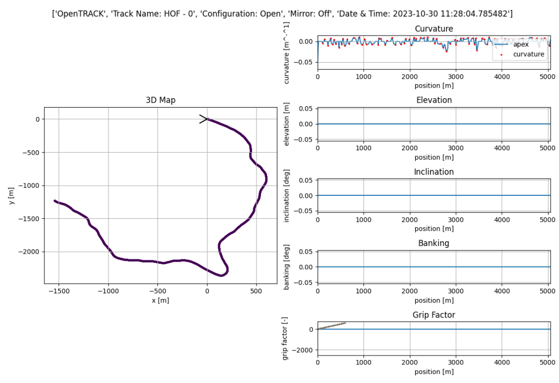


Figure 6.12: OpenTRACK GA Run 2

is in the representation of each snippet of the track being a gene rather than a list of pieces making up a segment. I tried to fix this through different individual initialization, but I think it would require an almost top down rewrite of the algorithm to fix this. I do think it is possible to achieve similar results with OpenTRACK as the B-Spline. However, even if you could, it may not be worth the 10 times slower computation just to achieve similar or worse results. There would need to be significant reason for using a track designer from a simulator as I mentioned with co-development of a car earlier.

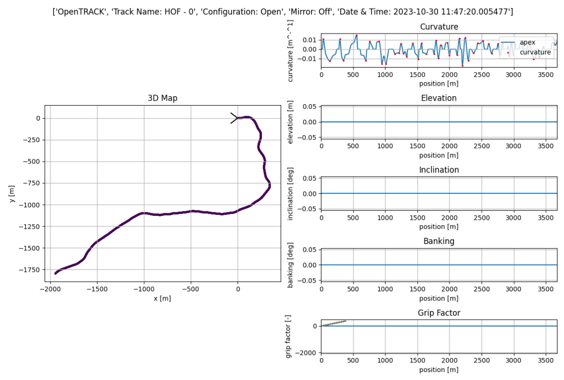


Figure 6.13: OpenTRACK GA Run 3

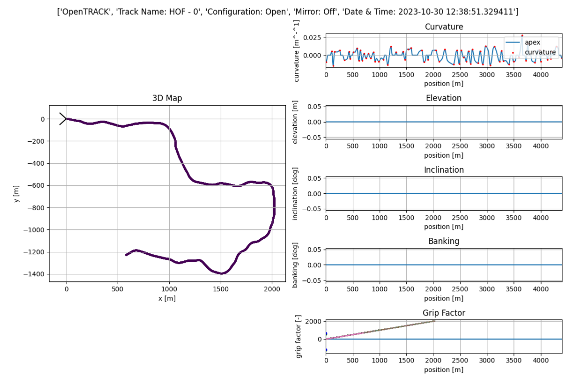


Figure 6.14: OpenTRACK GA Run 4

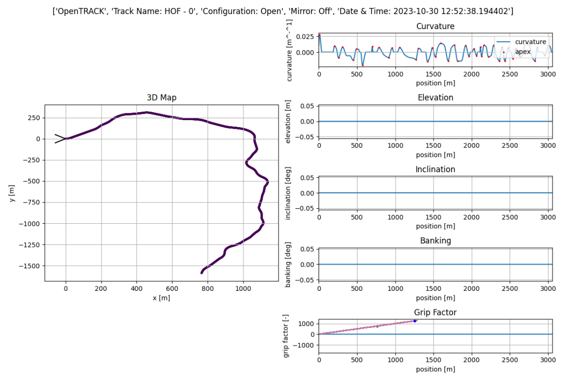


Figure 6.15: OpenTRACK GA Run 5

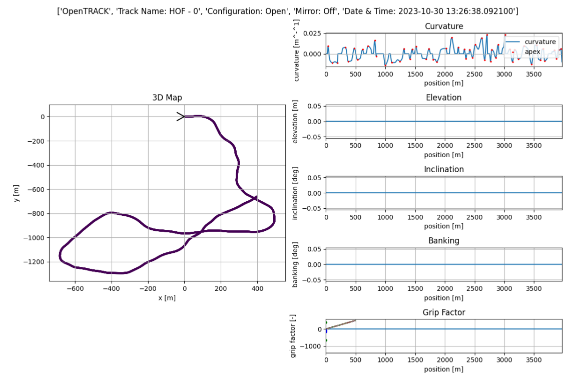


Figure 6.16: OpenTRACK GA Run 6

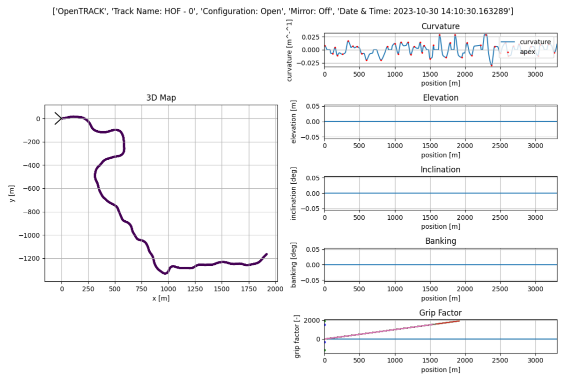


Figure 6.17: OpenTRACK GA Run 7

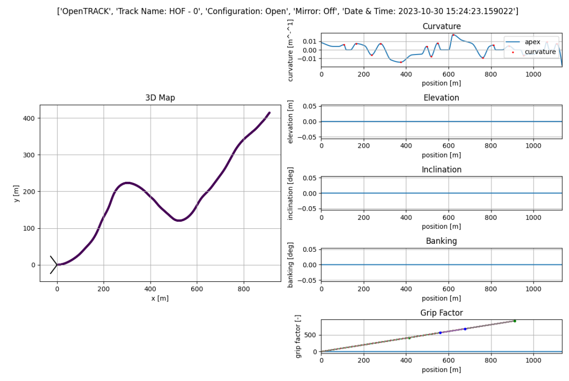


Figure 6.18: OpenTRACK GA Run 8

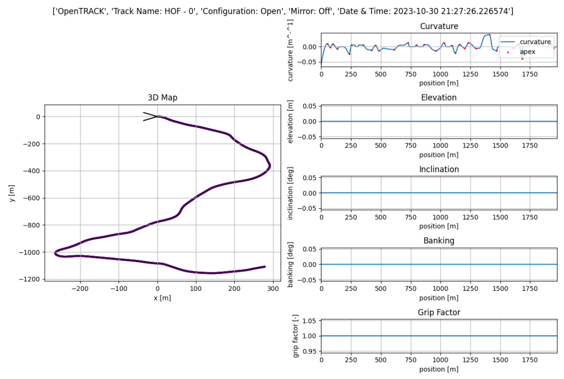


Figure 6.19: OpenTRACK GA Run 9

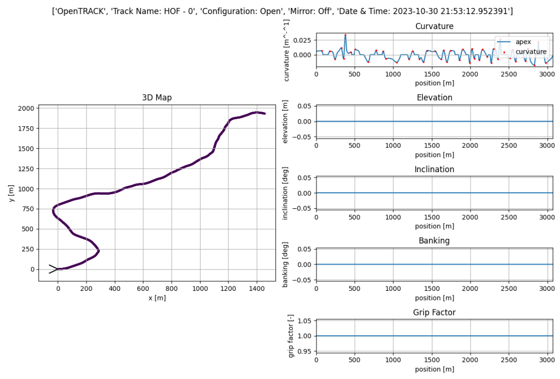


Figure 6.20: OpenTRACK GA Run 10

Chapter 7

Conclusions

7.1 Conclusion

As I showed early on in this thesis, F1 is a constantly changing sport utilizing the latest technologies to fight for marginal gains over competitors every single day. Since it is a championship sport with points given every race to decide the winner, every single lap and every single turn matters. A single point can mean tens of millions of dollars to a driver or team, and they will do anything to reach the top. With the ever restrictive rules coming from the FIA, especially during the current cost cap era, this also brings another dynamic to F1 development: loopholes and shortcuts. Teams have been finding loopholes and shortcuts around the rules since F1 was first created many decades ago. While in many sports these actions would be marked as cheating and result in harsh punishments, as is also the case sometimes in F1, some of these achievements are actually lauded as clever and playing the system. I would certainly not propose any method of cheating or even a loophole as my research here. But I do think algorithms like mine fill the second style of shortcuts.

In the existing forms of these algorithms, EAs achieve a best result of 86.971 and worst result of 106.4179, also having an overall average between our 4 main algorithms of 96.772. Compared to the best time in the real world this means that out of 40 runs, our single worst performance was 3 tenths of a second slower. Worth noting, this one time slower than the real world would still have resulted in a second place finish. Every other run of the 40 was faster, even up to 18% faster, than real life. With an average runtime of 4:09:21 across all our algorithms on consumer grade hardware, it also evident that these algorithms are quite efficient, at least when compared to traditional research and development. With these results and future improvement, using an evolutionary algorithm for car development could save a team several tens of millions of dollars in engineering, wind tunnel testing time, carbon fiber molding, and many other areas. In times with a cost cap, this could allow them to find time elsewhere in their development process.

For track algorithms there are two areas of application. The first is very straightforward: designing an entertaining and challenging track for Formula 1 or other competitive motorsports. This could be for a permanent track with room to work, or for a street circuit looking to use the roads and surfaces they already have. The second application is as discussed in the end of last chapter, boosting car development further.

By developing tracks that push the weaknesses of your car, you might be able to make further discoveries in the simulator about where you could gain tenths of a second with aerodynamic or setup changes. Both of these could result in another several tens of millions saved for a racing team or track designer. I believe that the comparison to SQP shows that these algorithms can provide a much more feasible, efficient, and accurate solution to problems in motorsport and other fields where other non-evolutionary algorithms may fall short. This comparison proved that EAs achieved much faster lap times in shorter run times than the comparable SQP runs. It also showed that EAs are more robust to tight constraints and bounds that will serve more realistic use cases in the future.

Many researchers before me have made critical strides in advancing car or track development using evolutionary algorithms. But, as discussed several times, they are largely rooted in the world of video games. These still provide serious benefit for this industry, and my algorithms would not work as well in that case, though they could be adapted. Rather, my work expands on these and seeks to provide a touch more of realism to show the vast possibilities of using algorithms like these in the automotive and motorsports industries.

7.2 Contributions

In this paper, I introduce 6 evolutionary algorithms. Genetic Algorithm, Codependent GA, Evolutionary Strategies, Codependent ES, Cultural Algorithm, and a Multi-Objective Codependent GA. I also implemented the SQP algorithm for optimization over a laptime for EA comparison. All 6 EAs outperformed the SQP algorithm for every run, while typically having faster run times as well. I was able to achieve 18% improvement on laptime in several hours or 12% laptime improvement in under half an hour. Through this I believe I have shown that EAs are viable and scalable to be utilized with more intricate constraints and lap time simulation in order to be used in a motorsports engineering department.

While I think that EAs are most currently equipped for car setup optimization, I showed that EAs can be utilized in other ways in the motorsports industry as well, creating 2 types of track optimization algorithms. These algorithms, while in need of fine tuning and improvement moving forward, show promise for track designers and teams alike to benefit from the creation of several viable tracks in a span of 2 to 20 minutes. These track algorithms would encourage tracks to continue to make layout improvements for the sake of racing and enjoyment, while also being useful for the teams and exploring the strengths and weaknesses of their cars more fully than existing tracks may be able to.

With both the car and track optimization algorithms, I remained as closely tied to real world properties and performance as possible. This was a limitation in some ways, as available to consumer lap simulations and data about Formula 1 can be hard to come by. However, I do not think this is reason enough to resort to utilizing less reliable means such as video games in order to perform optimization. I propose further research into the realism of physics based lap time simulation that allows these algorithms to perform even better and more accurately. I believe that the algorithms implemented here are capable of working with a Formula 1 team's private lap simulator and working to improve their research and development processes immediately. Until more is accessible to the public however, more work is needed.

7.3 Limitations

There are two factors that are the largest limitations on this work by a significant margin. First is the availability and accuracy of lap time simulators. Using an imperfect representation for my fitness function makes it difficult to make serious claims about tenths of a second gained over a lap or anything quantitative like that. However, I believe that my work with this lap simulator shows the viability that these algorithms have if I were to be provided with a more advanced and detailed simulation. The second major limitation is the availability and accuracy of data about Formula 1. For evolving a car, many of my values and bounds had to be taken from very unreliable sources like online forums or even video games. Also, this is still an incredibly limited scope compared to the millions of data points that an actual F1 team collects in simulators or over a race weekend. It is estimated that a F1 collects 160 TB of data per race weekend (Miller, 2021), where I am working with kilobytes at most. Having access to more data, as well as data over more parameters of the car, would allow me to see how generalizable this research is. Of course, SQP is limited by the same constraints as the EAs are, largely up to the objective function's realism and detail. Though I think this limitation furthers the strength of the EAs over SQP, rather than them struggling equally. In my first versions of SQP during testing, I used much wider bounds for a couple of the more restrictive parameters like brake pad height, to ensure that the algorithm did not consider my bounds infeasible to solve. With only a couple of parameters expanded (though expanded greatly), the SQP algorithm was able to converge much more quickly, and reach a solution in the low 100s, much closer to the EAs. I think this goes to show that this is a very particular problem to solve and that algorithms like my GAs which are able to navigate the tight bounds to learn optimal solutions can provide a serious advantage to design optimization.

7.4 Future Work

While I would love the opportunity to do a project like this with a better lap simulator or real data as discussed in chapter 5, I am going to focus on where this work can go with existing systems and data. The first work I would like to expand upon is in the Cultural Algorithm. I believe that setting up a culture and knowledge base for each of the F1 teams and allowing them to evolve to see the uniqueness of each team's engineering tendencies come through could be very insightful. Similarly, you could include a mechanism for a team to buy knowledge from another team, as teams like Haas F1 purchase parts and research from Ferrari, as well as many other customer relationships in F1. I don't have enough insight into the inner workings of F1 to form a system like this, but I believe this information is out there. You could use established theories like the Island Model to allow for information sharing through purchasing knowledge or for when teams discover things about each other through data and picture collection at the track. By writing this kind of algorithm, you are not only able to evolve where your car could be faster, but also discover where your opponents might gain speed and be able to prepare for those developments, or beat them to it. In one of the books referenced in Chapter 3, (*Linkage in Evolutionary Computation*, 2008), there is a chapter dedicated to gene linkage with the island model (Skolicki, 2008). I think this research could be applied also to include the realism from my codependent algorithms with the cultural algorithm.

The second future area I see advantages of using these algorithms in motorsport would be in evolving 3D representations, rather than just setup parameters. You could evolve a cloud of 3-dimensional coordinates to represent a particular part of a car, or perhaps the whole car. The bounds are still available from the FIA so design would be bounded, and you could use varying fitness functions from reduction of drag on a rear wing, increase of downforce on the sidepod, or many others. In doing so, you could very accurately simulate different car concepts and evaluate viable parts and setups to research with your limited time and budget. This is quite far removed from my work here, but I do think this would be possible.

The third form of improvement I see is in co-development of a car like I discussed in chapter 6. By creating tracks to further explore the strengths and weaknesses of your car, removing the limitation of the currently run 23 tracks, you might be able to gain an even greater understanding of your car for future development and improvement. The third opportunity for expansion upon this work would be in the representation of the track. While the solution I implemented is functional, it does not feel like the cleanest solution for drawing a track between points in a coordinate plane. I think an easy point to make about this is that technically none of my B-Spline tracks have truly straight sections. While not all 'straights' in F1 are perfectly straight, it feels like an omission for it not to be at all possible here. With this, I also think that path optimization functions like the one from (Zhu & Borrelli, 2024) could become useful in bolstering the relationship between generated tracks, lap simulators, and eventually creating insights that can be interpreted by a strategy team and drivers. By understanding more clearly the path travelled over a track, you can more easily understand where your car might be struggling the most. Racing teams spend dozens of hours poring over their telemetry data for this exact reason, and using simulated methods like these may enable them to understand things in the real world more fully.

The last area for future work that I see is with track generation from a given area. Many new circuits in Formula 1 are "street circuits" meaning that they are run on existing roads in a city, like the new Las Vegas Grand Prix for the 2023 season. I think it would be interesting if an algorithm could take an input of existing roads and use them to come up with the most entertaining circuit layouts on the existing road system.

References

Adair, J., Brownlee, A., & Ochoa, G. (2016). Evolutionary algorithms with linkage information for feature selection in brain computer interfaces. *Advances in intelligent systems and computing*, 287–307. https://doi.org/https://doi.org/10.1007/978-3-319-46562-3_19

Anderson, G. (2022, January). Gary anderson: What 2022-style f1 ground effect looks like. Retrieved October 13, 2023, from https://www.the-race.com/formula-1/gary-anderson-what-2022-style-f1-ground-effect-looks-like/

Boggs, P. T., & Tolle, J. W. (1995). Sequential quadratic programming. *Acta Numerica*, *4*, 1. https://doi.org/https://doi.org/10.1017/s0962492900002518

Candelpergher, A., Gadola, M., & Vetturi, D. (2000). Developments of a method for lap time simulation, 2000-01–3562. https://doi.org/10.4271/2000-01-3562

Castellani, F., & Franceschini, G. (2003). Use of genetic algorithms as an innovative tool for race car design, 2003-01–1327. https://doi.org/10.4271/2003-01-1327

Chalkiopoulos, M. (2020, April). Openlap lap time simulator. Retrieved October 7, 2022, from https://github.com/mc12027/OpenLAP-Lap-Time-Simulator

Cissell, K. (2017, July). Cultural-algorithm. Retrieved October 30, 2023, from https://github.com/KeithCissell/Cultural-Algorithm

De Rainville, F.-M., Fortin, F.-A., Gardner, M.-A., Parizeau, M., & Gagné, C. (2014). Deap. *ACM SIGEVOlution*, *6*(2), 17–26. https://doi.org/https://doi.org/10.1145/2597453.2597455

Duff, C. (2022, April). F1 innovations that made their way into road cars | racv. Retrieved September 26, 2023, from https://www.racv.com.au/royalauto/transport/cars/f1-innovations-in-road-cars.html

Exploitation of linkage learning in evolutionary algorithms. (2010, January). Springer Nature. https://doi.org/https://doi.org/10.1007/978-3-642-12834-9

F1 insights by aws. (n.d.). Retrieved October 12, 2023, from https://aws.amazon.com/sports/f1/

F1 technical regulations 2023. (2022, December). https://www.fia.com/sites/default/files/fia_2023_formula_1_technical_regulations_-_issue_4_-_2022-12-07.pdf

Fernie, M. (2016, December). Drag coefficients explained: Which kind of car is slippiest? | news | carthrottle. Retrieved October 13, 2023, from https://www.carthrottle.com/news/drag-coefficients-explained-which-kind-car-slippiest

Genetic algorithm options - matlab. (2023). Retrieved October 12, 2023, from https://www.mathworks.com/help/gads/genetic-algorithm-options.html

Harper, D. (2023, June). Openlap-python. https://github.com/dghjunior/OpenLAP-Python

Harris poll. (2022, May). Retrieved September 26, 2023, from https://theharrispoll.com/briefs/formula-1-fandom/

Hayward, K. (2007, January). *Application of evolutionary algorithms to engineering design* [Doctoral dissertation].

Hedges & Company. (2021, June). How many cars are there in the world in 2021? stats by country. Retrieved September 26, 2023, from https://hedgescompany.com/blog/2021/06/how-many-cars-are-there-in-the-world/

International sporting regulations - fia. (2023, October). Retrieved October 24, 2023, from https://www.fia.com/regulation/category/123

Linkage in evolutionary computation. (2008, January). Springer Nature. https://doi.org/https://doi.org/10.1007/978-3-540-85068-7

Loiacono, D., Cardamone, L., & Lanzi, P. L. (2011). Automatic track generation for high-end racing games using evolutionary computation. *IEEE Transactions on Computational Intelligence and AI in Games*, *3*(3), 245–259. https://doi.org/10.1109/TCIAIG.2011.2163692

Luque, P., Mántaras, D. A., Maradona, Á., Roces, J., Sánchez, L., Castejón, L., & Malón, H. (2020). Multi-objective evolutionary design of an electric vehicle chassis. *Sensors*, *20*(13), 3633. https://doi.org/10.3390/s20133633

Marr, B. (2023, June). How artificial intelligence, data and analytics are transforming formula one in 2023. Retrieved September 26, 2023, from https://www.forbes.com/sites/bernardmarr/2023/07/10/how-artificial-intelligence-data-and-analytics-are-transforming-formula-one-in-2023/?sh=2c3accd21c6a

Mercedes-Benz. (n.d.). Benz patent motor car: The first automobile (1885–1886): Mercedes-benz group. Retrieved September 26, 2023, from https://group.mercedes-benz.com/company/tradition/company-history/1885-1886.html

Miller, R. (2021, October). How cloud data-crunching power accelerates the f1 racing experience. Retrieved October 12, 2023, from https://www.datacenterfrontier.com/cloud/article/11427867/how-cloud-data-crunching-power-accelerates-the-f1-racing-experience

Petrány, M. (2012, December). The first 40 years of motor racing. Retrieved September 26, 2023, from https://jalopnik.com/the-first-40-years-of-motor-racing-5969690

Race standings. (1950, May). Retrieved September 26, 2023, from https://www.formula1.com/en/results.html/1950/races/94/great-britain/race-result.html

Reynolds, R. G. (1994). An introduction to cultural algorithms.

Saad, S., Palazzolo, T., Zhang, C., Reynolds, R. G., Lemke, A., Shea, J. O., & O'Shea, C. (2022). Learning to evolve procedural content in games using cultural algorithms, 106–115. https://doi.org/10.1109/TransAI54797.2022.00026

Schwartz, N., & McGarry, T. (2014, January). The nfl is the most popular sport in america for the 30th year running. Retrieved September 26, 2023, from https://ftw.usatoday.com/2014/01/nfl-most-popular-sport-poll

Sheta, A., & Turabieh, H. (2006). A comparison between genetic algorithms and sequential quadratic programming in solving constrained optimization problems. *ICGST International Journal on Artificial Intelligence and Machine Learning (AIML)*, *6*(1), 67–74.

Skolicki, Z. (2008). Linkage in island models. *Studies in computational intelligence*, *157*, 41–60. https://doi.org/https://doi.org/10.1007/978-3-540-85068-7_3

Sylt, C. (2018, April). Ferrari: 570 million f1 budget revealed. Retrieved September 26, 2023, from https://www.forbes.com/sites/csylt/2018/04/15/ferrari-570-million-f1-budget-revealed/?sh=5098b8d67fa5

Togelius, J., De Nardi, R., & Lucas, S. M. (2007). Towards automatic personalised content creation for racing games. *2007 IEEE Symposium on Computational Intelligence and Games*, 252–259. https://doi.org/10.1109/CIG.2007.368106

Togelius, J., Nardi, R. D., & Lucas, S. M. (2006). Making racing fun through player modeling and track evolution.

Williamson, M. (2022). A timeline of formula one. Retrieved September 26, 2023, from http://en.espn.co.uk/f1/motorsport/story/3836.html

Wilson, R. (1963). *A simplicial algorithm for concave programming* [Doctoral Dissertation]. https://nrs.harvard.edu/URN-3:HUL.INSTREPOS:37372525

Wloch, K., & Bentley, P. J. (2004). Optimising the performance of a formula one car using a genetic algorithm. In X. Yao, E. K. Burke, J. A. Lozano, J. Smith, J. J. Merelo-Guervós, J. A. Bullinaria, J. E. Rowe, P. Tiňo, A. Kabán, & H.-P. Schwefel (Eds.), *Parallel problem solving from nature - ppsn viii* (pp. 702–711, Vol. 3242). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-540-30217-9_71

Wolpert, D., & Macready, W. (1995). *No free lunch theorems for search* (tech. rep.). Santa Fe Institute.

Zhu, E. L., & Borrelli, F. (2024). A sequential quadratic programming approach to the solution of open-loop generalized nash equilibria for autonomous racing. *arXiv (Cornell University)*. https://doi.org/https://doi.org/10.48550/arxiv.2404.00186