

Emergent Computing for Optimisation

Introduction

The aim of this coursework is to demonstrate my understanding of Evolutionary Algorithms (EA) by completing a personal project in which I design, implement and evaluate a series of variations to the EA to increase its chances of finding a better solution to a problem about Women's Track Cycling. The approach taken was to optimise both the transition and pacing strategies by trying different selection, crossover, mutation and diversity methods and drawing results to decide which ones produced the smallest race times. Each method was tested with different parameters to get the most of it and make sure they were used correctly. The choice of this approach was due to the fact that the provided EA had very simple functions and after research and attending classes it was clear that to produce better results, those functions had to be improved. Both pacing and transition strategies were optimised at the same time since they are both interrelated in real-life and by only optimising one of them, the algorithm would only be searching half of the possible search space and miss the other areas.

Method

Solution Representation

A representation is a method of writing down a potential solution to a problem, and given the representation, all the possible solutions to the problem represent the search space.

The phenotype of the Team Pursuit problem represents the fitness of the racers to complete a race in the least amount of time possible. In the genotype space, solutions are represented as 2 separate arrays:

- The transition strategy is represented by a binary string where the only two possible values are true or false, therefore binary optimisations can be applied to it. The search space for binary problems is 2^n therefore since this problem has 22 genes, there are a total of 4194304 possible solutions to represent the transition strategy in this problem.
- The pacing strategy contains integers representing the energy (in joules) used per round, and therefore is treated like a numerical permutation. The values for this representation range from 200 to 120 and since there are 23 laps, the number of possible solutions is 23^{1000} which sums 5.34×10^{1361} .

Fitness Function

The fitness function is an evaluative function that measures the quality of the proposed solution, and it is derived from the problem's objective. In this problem, we measure how good a race is depending on how quick it is finished. The smaller the time, the faster it was completed and therefore the better the solution is. This also allows us to compare different solutions to manage how we use the worst and best solutions of the problem.

The fitness function used in this problem checks if the race is completed or not. If it is completed, it returns the time taken to complete it and if it is not completed, it returns the proportion of race completed so it can be used to still rank the solutions from bad to worst.

For the good races that are finished, it also evaluates the sum of the 3 cyclists' remaining energies and shows that the fitness (time to finish race) and the energy remaining are directly proportional. To prove this, an additional class was made that runs the EA for as many times as you set and then outputs important values to a csv file, so every time an experiment took place, as little as it was, the information was being recorded. The time of every finished

race was saved, no matter how good it was, and the remaining energy and then plotted the information into a graph to be able to evaluate it.

The algorithm was left to run for 20 consecutive times to make sure the results were reliable and all of the completed race values were recorded and plotted using Matplotlib in a Python script (Figure 1). The results proved the hypothesis that the quicker the race was finished, the less energy (in Joules) was left. This information could then be used to produce the initial pacing strategy and start with a good population instead of a completely random population.

Algorithm Optimisation

The goal of the EA is to only look at a subset of all possible solutions in attempt of finding the global optimum, however the search space landscape can be unforgiving, therefore clever crossover, mutation and selection operators are needed to guide the algorithm to areas of high fitness. For this to happen, the algorithm used needs to have a good ratio between exploration and exploitation. Exploration searches new areas of the search space while exploitation searches neighbouring solutions of past examined solutions (Črepinšek, M (2013)).

Initialisation: The algorithm starts by initialising a random population of a set number of solutions (chromosomes) defined by the user and runs for a set amount of iterations. Within an iteration it selects two parents, applies a crossover method to them to create a child, mutates the child and evaluates it, if its fitness is better than the worse solution's fitness then it is replaced by the new child, progressing to a higher quality population.

The first random initialisation was modified so instead of creating a completely random population, it is seeded with some good solutions obtained from previous experience. The values in which the pacing strategy could be initialised were calculated by starting with the max value and subtracting a random number between 0 and the subtraction range. Each of them was tested 20 times so an average could be drawn. Results are shown Table 1.

Maximum value	Subtraction range	Race time
300	50	246.9
300	100	252.1
350	50	233.14
	...	
500	200	218.1
500	250	220.29
500	300	223.83

Table 1 Initializing Population Evaluation

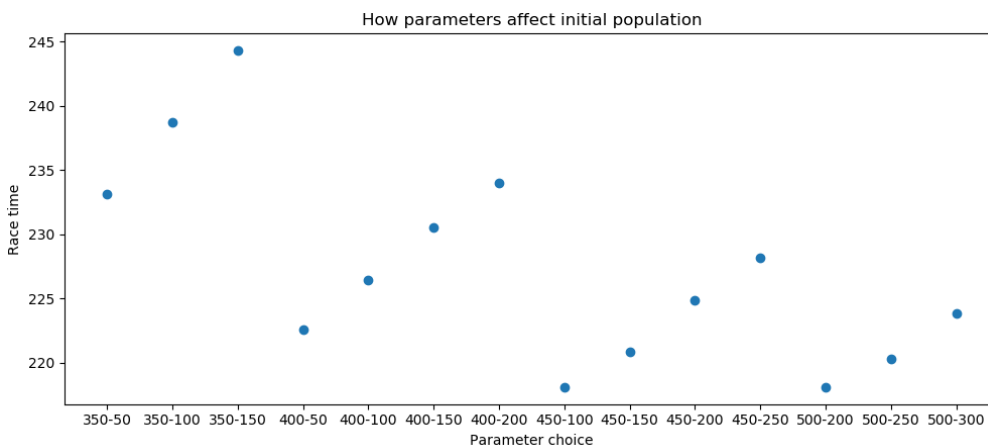


Figure 2 Initializing pacing strategies between a defined range

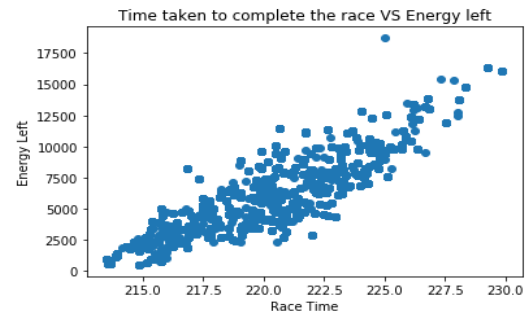


Figure 1 Fitness Function Selection

The results show the average of the best races times after only 30 iterations of the EA running for 20 times. The possible starting energy ranges from 200-1200. This allowed me to select the parameters that initialised the population with good candidate solutions (exploitation) and allow the algorithm to find good solutions faster, and later on, new solutions are introduced to increase diversity (exploration).

The results in Figure 2 show that the best starting values are a maximum value of 450 and a subtraction range of 100 (starting values range 450 to 350) or a maxValue of 500 and a subtraction range of 200 (starting values range 500 to 300).

Selection methods choose individuals to become parents and drive the evolution forward by biasing selection towards better individuals. A study by Saini, N. 2017 states that the most common selection methods are Roulette, Rank, Tournament and Boltzmann Selections. Therefore, most of those algorithms were tried in this coursework.

The Roulette Wheel Selection method, also known as Fitness proportionate selection works by selecting a parent based on a random spinner spinning around all of the possible fitness's. The proportion of the circle that every fitness occupies is calculated by reversing it ($\frac{1}{\text{solution fitness}}$) so that all solutions have a chance of being chosen, even the worse ones, but smaller race times (i.e. better fitness) have a larger proportions and therefore a larger probability of being chosen. This is an inconvenience since there will be little selection pressure when many chromosomes have similar fitness values, leading to premature convergence because the method was favouring the best individuals (Abd Rahman R, 2016).

Ranking selection sorts the population based on fitness and ranks worst chromosome as rank 0 and the best as N-1 (N: size of population). The rank is used to calculate the proportion of circle that the solution gets and then, just as in the Roulette wheel, a spinner choses a parent. The highest rank has the highest probability of being selected. This solves the problem of the roulette wheel of having similar portions for similar fitness values, since it assigns values depending on discrete rankings and prevents too quick convergence.

Tournament Selection selects a random subset of the population and from there, it selects the best one to be the new parent. This is a simpler method that does not require fitness scaling and selection pressure can be easily changed with different tournament sizes. The higher the tournament size, the higher the chance of selecting the best individuals therefore improving the population.

Crossover methods create offsprings by mixing the genes of 2 parents. In a simple crossover, a random crossover point is chosen in the gene, then, the first part of the gene up to the crossover is copied from parent 1, and the remaining from parent 2. The unused genes are combined to produce the second child. This keeps genes from a good parent and also represents unseen solutions, increasing diversity and therefore the chance of finding a better solution.

A multipoint crossover is similar to a simple crossover, but instead of having only one crossover point, it has n points. In this case, 2 crossover points were applied to both the transition and pacing strategies.

In uniform crossover, the gene isn't divided into segments, but instead, every gene is treated separately. A random probability is used to decide if that gene is taken from one parent or the other. However, to increase selection pressure, the probability can be biased to have a higher chance of choosing genes from the better parent, though this could lead to premature convergence.

Mutation methods introduce new areas of the search space that were not explored before. Mutation happens based on a probability, because if mutations are significant, the EA will not be searching the search space in an effective way but instead doing a random search of it. Mutations introduce new genes and therefore increase diversity; this allows the population to not get stuck in a local optima and also consider other points of the search space.

The most simple binary operation that can be applied to the transition strategy is to flip one of the genes. But in this problem it was also applied to the pacing strategy by creating a random number between 100 and 200 and adding it to the selected gene. This mutation can be controlled by choosing how many genes can be mutated in every iteration. Let's say that the maximum number of genes that can be mutated is 3. This will create small mutation in every gene and not add a much diversity since sometimes only 1 gene will change, others 2 and others 3. But if the mutation rate is 15 and 15 genes out of 23 are modified, the mutation can be too big and, as mentioned before, perform a random search. However, the bigger the mutation the more diversity is added.

Swap mutation, as the name implies, swaps two genes from a chromosome. It does not introduce new genes so there is less exploration and limits the scope of the search space in which the algorithm can explore. This operator can change 2 independent genes or a subset of the chromosome with another subset.

Diversity methods introduce new individuals into the population, allowing the algorithm to search new areas of the search space and increasing the chances of finding a better solution. The algorithms used for this problem were:

Sawtooth, which decreases the population size over a set number of iterations and then refills the population with randomly initialised individuals, allowing the algorithm to search different areas of the search space.

The Hill climber algorithm takes the best solution of the population and constantly creates random mutations on it, if at any of those mutations it gets a better solution, then it does mutations on the newly created solution.

Evaluation

To evaluate the best selection, crossover, mutation and diversity methods, the selection methods were tested first, regardless of the other ones. The algorithm ran 20 times, 5000 iterations each, and the best race time of each time was saved to a file, so then an average of the best 20 times could be calculated and the results evaluated and compared.

For the 2 selections methods used (tournament and roulette wheel), using the same parameters in both, it was observed that roulette gave, on average, better results.

Tournament	Roulette
214.39318181	213.60268181

The results for the tournament and roulette selection runs ([Appendix 1](#)) were compared and evaluated using statistical functions such as the Shapiro Wilk Test, and T test. However, the T test can only be used in datasets that have passed the Shapiro Wilk Test first, explained below.

The Shapiro Wilk test determines if a random sample (i.e. the best race times of 20 times running the EA), comes from a normal distribution. A small test result (W) indicates that the sample is **not** normally distributed, and the null hypothesis can be rejected. However, the limitations for this test are that there is a bias by sample size. The result will be more statistically significant when the sample size is larger, since the research findings will represent more of the population and will therefore be more meaningful (Royston, P. (1992)).

A script in Python was written to calculate the W and p values of the Shapiro Wilk test resulting in $W = 0.9439427852630615$ and $p = 0.23847633600234985$. Therefore since the p value > 0.05 , the null hypothesis that the sample comes from a population which has a normal distribution cannot be rejected, which means that the model is consistent even though it has a stochastic start every time it is initialised i.e. the results of the 20 experiments all ended up in the same range of values for the race times. The sample itself does not have a normal distribution (as seen in Figure 3 using R), but it is not expected since it is only a sample that is being evaluated. The distribution of the underlying population is still unknown, hence why it is called a hypothesis.

The Shapiro Wilk Test was also calculated for the roulette wheel race times and had similar results. Once both tests passed the Shapiro Wilk Test, they could be used to do the T test.

The T test bases test results on t-values, which are test statistics (in a hypothesis test, t-values are the standardized values, calculated from the sample data). It compares the data to the expected data under the null hypothesis. As the t-value increases, it means that the difference between the results and the null hypothesis increases, therefore a t-value of 0 means that they are exactly the same.

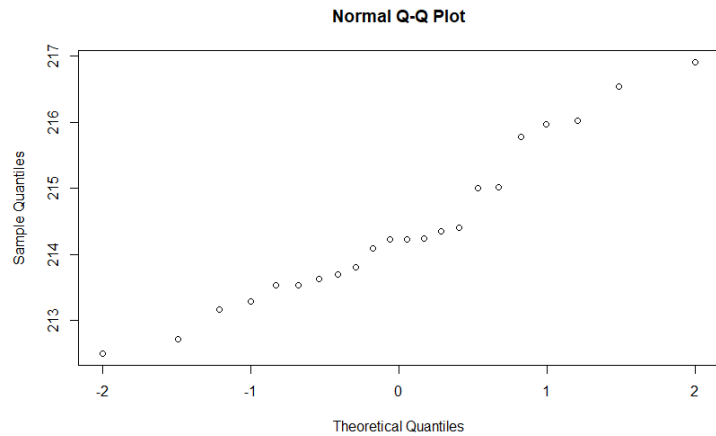


Figure 3 QQ Plot in R for tournament testing

SciPy has a function in Python that calculates the T-test for the means of two independent samples of scores. The null hypothesis is that they both have identical average values, assuming the population has identical variances by default. The T test for the tournament and roulette values showed that since the p value (0.016691) < 0.05 , the null hypothesis can be rejected and therefore the results do not have identical averages scores, which is exactly what is expected since both algorithms are supposed to give different results. Therefore, it can be concluded that the Roulette Wheel selection method provides better race times than Tournament selection and the results that prove this are reliable.

The crossover, mutation and diversity results were tested the same way as the selection methods. They all produced an average score of 213 seconds race time except for when sawtooth was included in the algorithm, then the average score was lowered to 212 seconds. Since the p values for all of them are < 0.05 , it can be concluded that the results are reliable and therefore the best configuration for the algorithm is as follows:

SECTION roulette wheel, **CROSSOVER** multipoint, **MUTATION** swap, **DIVERSITY** sawtooth. However, the best results appeared when both diversity methods, hill climber and sawtooth were used in the algorithm (race time average: 212.35680000001435), demonstrating that inserting novelty and diversity and allowing the algorithm to search new areas of the search space produces better results. The values for the averages, Shapiro and T test results for the different methods can be viewed in [Appendix 1](#).

Parameter Testing

After the best methods were evaluated, the parameters of the best configuration were tested to choose the best ones.

- The mutation and crossover probabilities were either 1.0 or 0.5, but regardless of what they were, the results produced similar averages.
- The iterations for every EA were set to 5000, even though 10000 produced better results, obviously, but that's more like doing an exhaustive search than actually optimising the algorithm. Therefore, all of the tests were done with 5000 iterations each.
- For the sawtooth algorithm, different values were tested such as the population size, the number of iterations after which an individual is removed from the population and the minimum size of the population before it starts refilling again. After various test, it was determined that the best combination was a

population of 30, remove an individual every 30 iterations and refilling the population when it hit a size of 20 individuals.

- The maximum rate of mutation determines how many genes of the chromosome can be mutated. For instance, there are 23 genes in the pacing strategy and 22 in the transition strategy. In the EA algorithm, the mutation rate is generated by selecting a random number between the maximum rate and adding one. Therefore the maximum number for this parameter is 21. If the random number, by chance, was 21 and we add the plus one, the mutation would affect every single gene in the chromosome and create a completely new individual, but if we select a higher number for the mutation rate and then we add the plus one, the algorithm would throw an error. To test the best values for this parameter, normal (valid) data, boundary (extreme) data and abnormal (erroneous) data were used. However, they do not seem to have a massive impact in the general race time therefore the normal value 6 was set for all the other tests.

	Normal		Boundary		Abnormal	
mutation rate max	6	14	0	21	-1	23
race time (average of 10 iterations)	212.356	213.673	212.367	212.358	error	error

Conclusion

The best race time achieved was 210.7650000, however, this was after 100,000 iterations using roulette selection, uniform crossover, simple mutation and no diversity methods, hence this was more of an exhaustive search than an optimised algorithm. When all the tests were run with 5,000 iterations, the best race time (211.2580000) was given using roulette wheel selection, multipoint crossover, swap mutation and both sawtooth and hill climber. The parameters were the usual ones explained in the section above.

Overall, the approach was successful, and managed to produce good results. To reduce the race time even more, there are some other algorithms that could have been used, mentioned in the Future work section below. However, the algorithms were evaluated thoroughly with different tests and programs and this made it possible to find the best configuration for the algorithms and therefore increase their effectiveness in reducing race time. The choice of algorithms was good and many of the ideas came from the lecture slides and some research. However, there is always room for improvement and this algorithm will never be optimum unless an exhaustive search of the search space is done, and the best race time can be compared with the one achieved using the algorithm.

An interesting observation was that crossover and mutation turned out to be the factors that influenced the results the least. More significant changes happened when different selection methods were evaluated or when the diversity methods were added. The biggest change was definitely shown by the inclusion or not of diversity methods.

Future work

Since the beginning, the strategy has been to crossover and mutate both the pacing and the transition strategy following the hypothesis that this would provide better results since they are both interrelated in the race. However, in future work this should be tested to confirm or disconfirm the hypothesis. Another hypothesis to test is if Simulated Annealing would produce better results, or adding a taboo list so that the recently visited solutions which were not good aren't visited again. Simulated Annealing is useful in finding global optima in the presence of large numbers of local optima, and is basically hill-climbing but instead of selecting the best race time, it selects a random race time, and if the move improves that race time, then it is accepted (Aarts, E. (1998)).

References

Aarts, E., & Korst, J. (1988). Simulated annealing and Boltzmann machines.

Abd Rahman, R., Ramli, R., Jamari, Z., & Ku-Mahamud, K. R. (2016). Evolutionary algorithm with roulette-tournament selection for solving aquaculture diet formulation. *Mathematical Problems in Engineering*, 2016.

Črepinšek, M., Liu, S. H., & Mernik, M. (2013). Exploration and exploitation in evolutionary algorithms: A survey. *ACM Computing Surveys (CSUR)*, 45(3), 35.

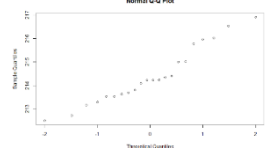
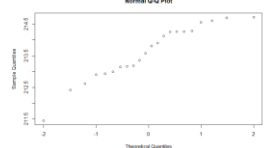
Royston, P. (1992). Approximating the Shapiro-Wilk W-Test for non-normality. *Statistics and computing*, 2(3), 117-119.

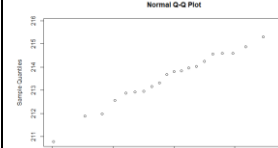
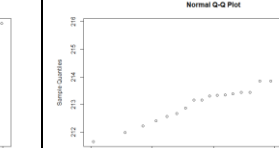
Saini, N. (2017). Review of selection methods in genetic algorithms. *International Journal Of Engineering And Computer Science*, 6(12), 22261-22263.

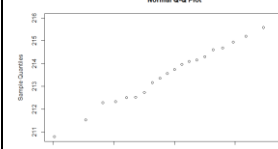
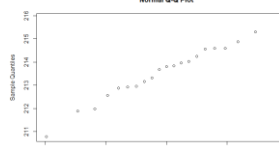
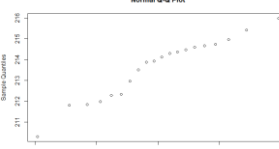
Sharma, P., & Wadhwa, A. (2014). Analysis of selection schemes for solving an optimization problem in genetic algorithm. *International Journal of Computer Applications*, 93(11).

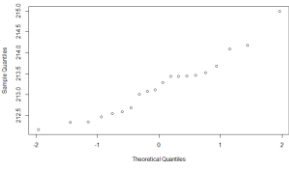
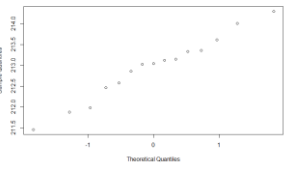
Appendices

Appendix 1

SELECTION	Tournament	Roulette
qqPlot in R		
Average	214.39318181	213.60268181
Shapiro W	0.943942785	0.937561690
Shapiro p	0.238476336	0.176234304
T-test	t = 2.493050082 p = 0.016691839	

MUTATION	Simple	Swap
qqPlot in R		
Average	213.613714285	213.284900000
Shapiro W	0.985066413	0.942402064
Shapiro p	0.978938758	0.266095042
T-test	t = -0.948343418 p = 0.348794355	

CROSSOVER	Uniform	MultiPoint	OnePoint
qqPlot in R			
Average	213.62057142858538	213.61371428572826	213.62640000001397
Shapiro W	0.985724151134491	0.9850664138793945	0.9462866187095642
Shapiro p	0.9831384420394897	0.9789387583732605	0.3142116367816925
T-test	t = 0.030344444927830866 p = 0.9759471096139326		
T-test	t = 0.01730862434549485 p = 0.9862764544113543		

DIVERSITY	Hill climber	Sawtooth
qqPlot in R		
Average	213.19470000001417	212.943466666668088
Shapiro W	0.9453279376029968	0.9753602147102356
Shapiro p	0.30167096853256226	0.9279804825782776
T-test	t = 0.9908058032282363 p = 0.3289901599458629	