Emergent Computing for Optimisation

Coursework Report

Rossa Heraty Quirke

40204617

1. Introduction

This report looks at the ability of evolutionary algorithms (EAs) to solve an optimisation problem. A team of female cyclists are participating in a track-cycling event. The cyclists can alternate the pace at which they cycle, and at which point they transition positions so that a new cyclist is in the front, but their abilities are fixed. Changing these strategies can result in increased performance as their energy usage and aerodynamic profile improves. Their performance, fitness in evolutionary terms, is measured by their finish time. This problem has many possible solutions due to the huge number of combinations of pacing and transition strategies available.

An EA was designed and implemented to evolve a population of cyclist teams. The transition and pacing strategies were evolved by algorithm simultaneously. Several different operators were experimented with and parameters of the evolutionary algorithm were tuned in order to optimise the cyclist’s finish time. By evolving both strategies, rather than for example evolving the transition strategy for set pacing values, a wider area of the search space is made available, it encourages diversity in the population, and treats each strategy with equal importance.

1. Method
   1. Implementation Details

Solutions to the track cycling problem were represented with two chromosomes for each individual member of the population. The chromosomes represented the pacing strategy, implemented as an array of 23 integers with values between the minimum and maximum power each cyclist can use, 200 and 1200 respectively, and the transition strategy, implemented as an array of 22 boolean values. By having two chromosomes different types of operators could be implemented that only work with certain types of values, for example uniform crossover cannot operate on booleans but can be used for the pacing strategy.

Fitness in the algorithm is defined as the time taken to complete the race, with a penalty for those that fail to finish the race at all. If a race is incomplete the fitness is a min-max normalised value, between 400 and 500, based on the proportion of the race complete. Being very near to completing the race gives a fitness close to 400 and having barely completed it gives near 500. This allows the algorithm to favour those close to finishing, which may be viable solutions with some adjustment, from very poor solutions which struggle to make much progress.

To optimise this problem an initial steady-state evolutionary algorithm was implemented, in order to be experimented on – which is discussed in Section 3. Steady-state refers to an EA in which an initial population is initialised and then evolved with a small number of new children, often only a single child, replacing the unfit members of the population. This is in contrast with a generational EA where in a whole new population is generated from the existing population. A generational EA is good for maintaining diversity as many new solutions are generated. Steady-state algorithms can be more robust, in that they the population is less likely to lose average fitness across generations, justifying their use.

Some constraints due to the nature of the problem must be outlined. As the two chromosomes are represented in the code as two different data types this meant some variants of operators could not be used uniformly across both. An example is mutation, to evolve the transition strategy, being an array of booleans, it makes sense to have the mutation operator randomly flip a boolean value from true to false and vice versa. However, the same method cannot be applied to integers. In order to solve this problem, the mutation operator implemented treats the chromosomes separately, with the transition chromosome using the boolean flip as mentioned and the pacing chromosome having its genes being either added to or subtracted by a fraction of its integer value.

* 1. Operators & Parameters

An EA traditionally consists of initialisation, selection, crossover, mutation, and replacement. The operators as they were implemented in the initial algorithm designed are outlined here along with their parameters.

**Initialisation**: A population of pacing and transition chromosomes were randomly generated. For each gene of the transition chromosome a random decision was made between it being true or false. For the pacing strategy a random integer was generated within a set range of 200 and 700. This range was based on the assumption that generating solutions where cyclists would be using their max or near to max power would quickly fatigue them and result in them unable to finish the race and so the team having a very low fitness. The population size was 100.

**Selection**: The selection operator chosen was tournament selection. This takes a subset of the population and chooses the individual, in this case referring to a cyclist team, with the best fitness compared to the others in the tournament. The tournament size was set to 10, this helped to maintain diversity by allowing weaker solutions from the population to be selected at time. Two parents for each iteration were selected to generate a single child.

**Crossover**: Two methods were implemented for crossover uniform and 1-point crossover. The uniform crossover was used for the pacing strategy. It had a 50/50 chance to pick a gene from either parent one or parent two, on average this results in the child having equal genetic data from each parent. For the transition strategy 1-point crossover was used. This selects a random cut-off point in the parent’s chromosome, and the segments are inherited by the child, a child could inherit only a small amount from one parent and most from the other or half of each parent’s chromosome though this is unlikely. All children were created as result of a crossover.

**Mutation**: In the mutation operator the strategies were again treated differently as previously mentioned. Mutations were made with a probability of 1/10 for each gene. When a pacing strategy gene was mutated it was either subtracted by or added to by its value multiplied by 0.05, a small value was chosen to not drastically change a possibly good existing solution and prevent the search becoming too random. If this mutation would cause a violation of the cyclist’s power limits it would do the reverse, so if a gene with a value of 1200 was randomly chosen to mutate and have 60 added, it would instead subtract 60. The transition strategy had the same probability for mutation and randomly inverted the boolean value.

**Replacement**: Tournament replacement was used to determine which individual in the population to replace with the newly generated child. The reasoning behind this choice is to help maintain diversity, by not always replacing the worst member of the population the homogenisation of individuals can be slowed down and maintain diversity through more iterations. Even if the worst fitness member in the tournament has higher fitness than the child it will still be replaced, so that new genetic information is always being injected. The tournament size was set to 10, the same as the selection tournament size.

1. Experiments & Analysis

This section details the experiments that were conducted to tune and optimise the EA. It shows the effects on fitness from the use of different operators, parameters, and diversity measures. All experiments ran 10 times with 1000 iterations of the algorithm.

* 1. Population & Tournament Sizes

Population controls the number of initial solutions, and therefore diversity, and is a method of controlling selection pressure when paired with some methods of selection, such as tournament selection. This experiment shows the changes in average fitness in relation to the population level. It ran with the parameters outlined in Section 2.

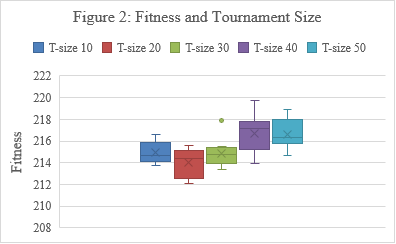
As seen in Figure 1 there appears to be a weak correlation in population size and fitness, with an improved fitness until over a population size of 150. However, some populations sizes had higher volatility in their results. In Table 1 we see that though the fitness differences are small different population levels appear to give more consistent results.

**Table 1**: Population Size with Standard Deviation of Fitness Results

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Pop. Size | 25 | 50 | 75 | 100 | 125 | 150 | 175 | 200 |
| Std. Dev. | 2.84 | 1.01 | 1.42 | 2.06 | 1.59 | 0.96 | 1.32 | 1.25 |

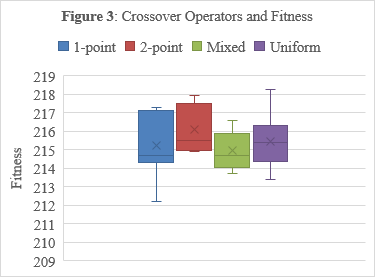
Looking at this we can see there doesn’t appear to be a clear trend in population size and standard deviation. However, we can say the best population size from this experiment can be considered to be 150. There is a miniscule difference between it and a size of 125 in terms of fitness, and at 150 the standard deviation was lower and therefore results were more consistent.

Tournament size strongly affects selection pressure with a high T-size exerting a strong pressure and vice versa. In this experiment tournament sizes for selection and replacement are modified, other parameters remain the same as in the initial design, aside from population size which has been changed to 150 based on the previous experiment in Section 3.1.

From this we can clearly see that a tournament size of 20 is achieving the best results, no other size is ever reaching a fitness of 212, though the spread of fitness values is wider than other levels such as 10 or 30. The better fitness achieved by a T-size of 20 is likely due to it striking the right balance of selection pressure for the population level, not too high that the solutions become homogenous but not too low as to not exploit effectively, allowing it to find optimums in the search space.

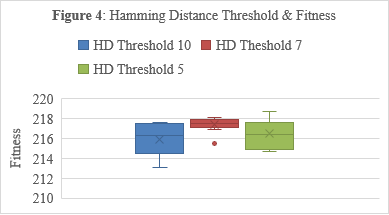
* 1. Crossover Operators

Many crossover operators could be applied to this problem. This experiment shows the affects of different crossover operators when paired with the tuned algorithm, with a population of 150 and T-size of 20 – all other parameters remain the same. The table compares Uniform, 2-point, 1-point, and a mix of uniform and 1-point – the Mixed being the initial implementation.

From Figure 3 we can see that the Mixed seems to perform the best overall, though it did not achieve results as good Uniform or 1-point it achieved results better than their mean consistently. The most comparable in terms of their mean are the Uniform and Mixed and digging deeper their T-test results reveal a two-tailed P value equals 0.2890, meaning the difference is not statistically significant. This may be due to the limited number of iterations run but from these results we can say there is no motive to change the Mixed implementation from the initial algorithm.

* 1. Fitness Sharing

From running the previous experiments, it could be observed that in later iterations the best fitness found remained static meaning a local optimum had been reached. Fitness sharing may offer a way to prevent this stagnation, by creating sub-populations via restricted mating based on a shared fitness value. Implementing fitness sharing for this problem poses some problems however, mainly due to the differing data types of the chromosomes. Hamming distance can be used on the transition strategy to measure difference but fails with the pacing as even a difference of 1, e.g. 100 vs 101, would increase the Hamming distance - though the expression of the gene is functionally the same. Euclidean distance is difficult to apply as the total sum of the pacing strategy genes do not reflect how they are expressed; two different pacing chromosomes could have the same Euclidean distance but be using different ‘resources’ in the search space. This implementation comprises and calculates the Hamming distance for the transition chromosome and restricts mating based on it, if a suitable mate could not be found after 5 attempts a random individual was chosen – 5 attempts being enough to search most of the individuals in the population.

As can be seen in Figure 4 across all thresholds tested it appears that the fitness sharing resulted in consistent results but overall worse than the algorithm without fitness sharing implemented. This may be explained by the lowering of the selection pressure this method caused, preventing effective exploitation. Several of the best fitness from runs were also extremely close in value, suggesting that sub-populations may have become stuck in local optima while others were filled with poor solutions.

* 1. Self-Adaptive EA

Adding a self-adaptive component to an EA can let allow it to dig out of local optima and explore or exploit the search space depending on its current state. In this example the self-adaptive feature is a gene added to the end of the pacing strategy chromosome that determines the amount the other pacing genes will mutate for. It takes a uses a random value from a distribution with a mean of 0 and a standard deviation of 1 to determine and multiplies it by the gene to determine the mutation amount, this gene itself is mutated with this method. The same parameters were used as previously described, with the mutation rate of 0.1. Self-adaptation seems to have negatively affected the algorithm’s performance; the mean fitness achieved was 215.93 with a standard deviation of 0.98, compared to the Mixed EA which achieves a mean of 214.04 with a standard deviation of 1.255. This drop may be explained by the limited number of iterations not giving the self-adaption mechanism enough time to self-correct. Also, the parameters such as population size and tournament size which worked well with the base EA may impede the self-adaptation, running it with a relatively large population of 150 may mean the self-adaption gene does not have enough chances to evolve. The T-test of results between the Mixed EA discussed previously and the Self-Adaptive EA reveals that the two-tailed P value equals 0.0015. This shows the differences are statistically significant and therefore showing a new underlying distribution is being displayed by the Self-Adaptative EA.

1. Conclusions

This approach successfully developed an algorithm that consistently achieves good results for the given problem. A good range of experiments were conducted to improve the algorithm and provide some insights. Parameter tuning the population size and tournament sizes seemed to prove most effective. Larger changes such as fitness sharing and self-adaption still provided overall good results but the results did not make a strong enough case for their adoption within the framework of the EA developed. However, these approaches may prove more successful with further tuning or modification. The biggest increase from the default strategy was due to evolving both strategies at once. While the final algorithm could achieve a finish time of 214 seconds regularly the best run observed during testing achieved a finish time of 212.12 seconds. This is compared to a default strategy which finishes the race in 263.268 seconds. The parameters used are displayed in Table 2.

Table 2: Parameters and Operators used for Best Finish Time



1. Future Work

The work detailed in this report could be expanded upon. Further niching methods could be experiment such as Crowding, ensuring that new individuals replace similar members of the population. Similarly, an Island Model EA could be tested, with different sub-populations being subjected to different parameters and operators. Further experiments could investigate how heuristics could be applied to generate a population with fitter individuals than random, based on previously found good solutions. Though this may come at the cost of some diversity.

Finally, another approach that was not considered in this report is consecutive evolution of the pacing and transition strategies. By, for example, optimising the transition strategy for a number of iterations, then ceasing to evolve it, and then optimising the pacing for that transition. This approach could be tuned and then compared to the approach of simultaneous evolution that was taken here. It would also allow for the option of developing different algorithms for each strategy allowing for the design to be more customised to the representation of the chromosome.