SET11108 Emergent Computing for Optimisation

Coursework Report

Matriculation number: 40452569

I. Introduction. (5 marks)

Team pursuit track cycling is an optimisation challenge. Within this coursework, among the other metaheuristics such as local search, simulated annealing and ant colony, a novel approach was used to tackle the above challenge. An evolutionary approach was chosen to experiment with its efficiency in optimising the performance of the woman team pursuit in terms of pacing strategy and transition strategy. The experiment is also to determine to what extends do the evolutionary algorithm's (EA) parameters such as population size, tournament selection size and iteration limit affect its efficiency in finding the desired solution.

In terms of pacing strategy, the front cyclist output energy was chosen to optimise as the upfront cyclist spend more energy than the rest of the team due to aerodynamics and other physics forces such as friction (Wagner et al., 2013). By optimising this, the team can improve their power usage to reduce the overall time. In terms of the transition strategy, a more even energy distribution among team members can be achieved while maintaining the most effective velocity profile (ibid.).

This report will outline the experiments as follow. First, a brief description of the experiment method will be presented with the solution representation, optimisation algorithm and the selection of parameters to be optimised. Second, the raw result of the experiments will be introduced using graphical illustration along with a discussion on the data analysis. Finally, a conclusion will be drawn based on the discussion and further comments on future work will also be included.

II. Method (20 marks)

This section will explain how a solution to the proposed issue will be represented in this experiment. As we have mentioned in the previous section, there are two main objectives for this experiment which are to optimise the number of times the first riders need to race prior to swapping with their teammates and also the power output of the cyclist when leading the team.

Within team pursuit, a transition is defined as a move when the front cyclist moves to the rear of the team and the second cyclist takes the lead. This move can only be performed at one of the two banked turns of the cycling track. While there are only 11 possible transitions, to anticipate the worst-case scenario when the team makes a transition at each and every turn, a default solution of 22 transitions was proposed to represent the team transition strategy (Wagner et al., 2013). As for pacing (power applied), the proposed representation is shown in half-laps. Due to the fact that the team can only make the transitions after the first 1.5 half-laps and before the last 1.5 half-laps, and we have the representation pacing strategy as ***n – 2 x (1.5) +2*** (with n is the total of half-laps) pacing power (ibid).

For this experiment, an unoptimized Java coded library was provided. In order to optimise the team strategies, a model was structured with an array of 22 binary values to indicate whether the team should swap places of the cyclist at the front to the back and the immediate after to the front. Also, another array with 23 integer values was used to represent the power that the front cyclist needs to apply to optimise the overall race time.

The default strategies were given as follow:

DEFAULT_WOMENS_TRANSITION_STRATEGY = {true, true};

DEFAULT_WOMENS_PACING_STRATEGY = {300, 300, 300, 300, 300, 300, 300, 350, 350, 300, 300, 350, 350, 350, 350, 300, 300, 350, 350, 350, 350, 300, 300};

In terms of the fitness function, the main focus of this experiment is to optimise or minimise the total time needed for the team to complete the race. Hence, it is logical to derive the fitness function as the minimisation of a variable. With the supplied library, the fitness function was basic which only based on the fact of whether the team had managed to complete the race, the elapsed time would be then provided as the fitness score. Had the said team did not finish the race, a penalty point of 1000 would be given as the fitness score.
It was proposed to tune this fitness function to better reflect the fitness comparison between the unfinished chromosomes(solution) as well as the finished chromosomes. With the test runs from the default strategies, it was found that the fitness score for the finished solutions was in the range of 235. Hence, the proposed penalty point for unfinished solutions was decided at 300. From a high-level point of view, the main idea behind the fitness function in this experiment is defined as the penalty for the remain or wasted energy with a reduction of the percentage of the completed portion of the race. The proposed fitness function for the unfinished chromosomes is as follow:

$$F = 300 + \frac{E\ remain}{Pcompleted}$$

However, when testing out the test data, the wasted energy was measured in joules and the values range from 1000 joules to 30000 joules. The difference in these values does not necessarily describe the goodness of the solution. Hence, a log scale of this value was introduced to dampen the effect of wasted energy and with the log scale introduced, a padding of 1 was added to avoid undefined value of log (0) should it happen and a multiplication of 10 times was also used to further justify the differences between the finished and unfinished configurations. The completed fitness function is as follow:

$$F = 300 + 10 \times \frac{Log(1 + E\ remain)}{Pcompleted}$$

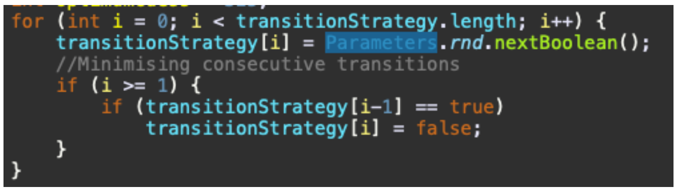
During this experiment, a simple evolutionary optimisation algorithm would be used to optimise the solutions (chromosomes). Evolution algorithm takes two parent's configurations and blends their genetic materials to derive a children generation that would then compete with each other through a mechanism of selection that favour the fitter offspring (Marks, 2007). Using this algorithm, the optimal or near-optimal will eventually emerge from the pool of solution configurations (gene pool) (ibid.). This was, in fact, also mentioned in the detail paper from Wagner et. al. (2011). This evolutionary algorithm (EA) is, however, subject to randomisation and premature convergence which results in the search algorithm being stuck in local optimum (Ramadan, 2012). Also proposed in the same paper, using both crossover operators and mutation will help with the diversity of the solution pools and improve EA's vulnerabilities to premature convergence (ibid.).

Apart from the crossover and mutation operators, overall parameters such as population size, tournament selection size and the maximum number of iterations would also be fine-tuned to achieve the lowest possible result.

III. Experiments & Analysis (20 marks)

Default Strategy Optimisation

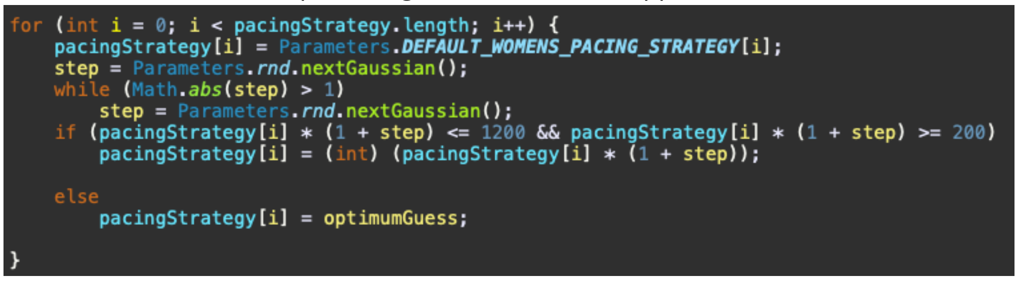
Rather than initialise the library with the default strategy when the team would make the transition at every turn, a random decision of not making such move at any point during the race along with an approach was proposed to minimise the changes in position to reduce the extra stress on the team by discouraging consecutive transitions.



```
for (int i = 0; i < transitionStrategy.length; i++) {
    transitionStrategy[i] = Parameters.rnd.nextBoolean();
    //Minimising consecutive transitions
    if (i >= 1) {
        if (transitionStrategy[i-1] == true)
            transitionStrategy[i] = false;
    }
}
```

Figure 1. Initial Transition Strategy Optimisation

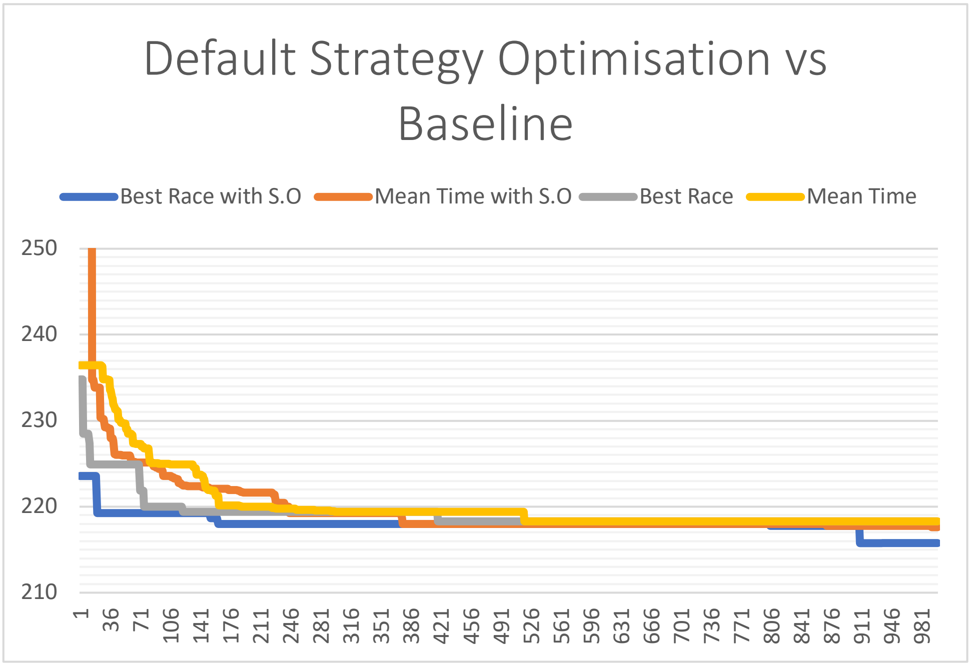
For the pacing strategy, a random variation in value within the Gaussian distribution was added to the default pacing strategy along with a safeguard for values that go over the power limit of between 200 and 1200. With such values, an optimum guess value will be applied.



```
for (int i = 0; i < pacingStrategy.length; i++) {
    pacingStrategy[i] = Parameters.DEFAULT_WOMENS_PACING_STRATEGY[i];
    step = Parameters.rnd.nextGaussian();
    while (Math.abs(step) > 1)
        step = Parameters.rnd.nextGaussian();
    if (pacingStrategy[i] * (1 + step) <= 1200 && pacingStrategy[i] * (1 + step) >= 200)
        pacingStrategy[i] = (int) (pacingStrategy[i] * (1 + step));

    else
        pacingStrategy[i] = optimumGuess;

}
```

Figure 2. Initial Pacing Strategy Optimisation

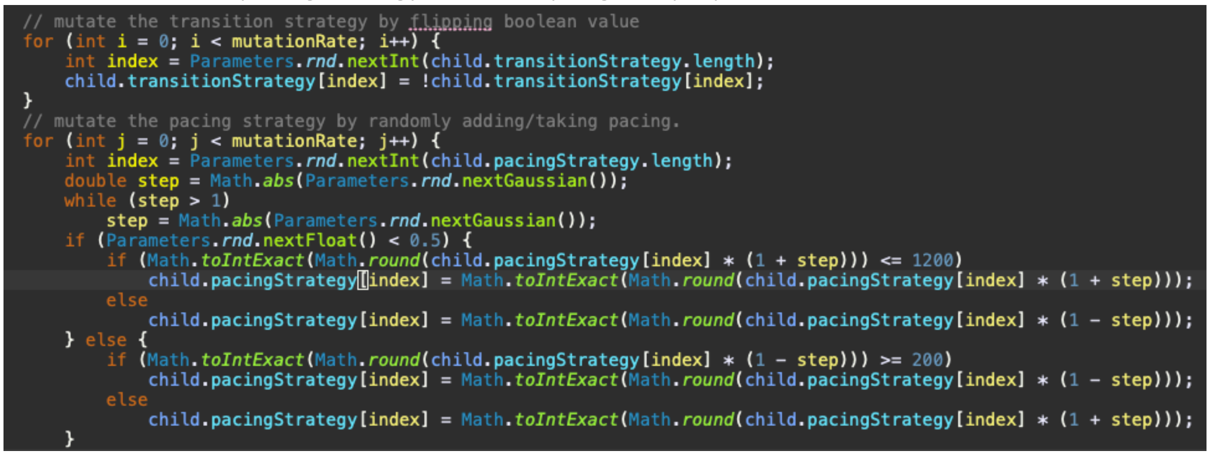
The result of this experiment is as follow:



As seen from the data above, with strategy optimisation, even with the starting mean-time was very high, topping above 250 seconds, the near-optimal performance was found at the very start at about iteration 36th that outperformed the baseline while it plateaued out with a breakthrough at about 900 iterations with 215.771 seconds.

Mutation & Crossover Operator Optimisation

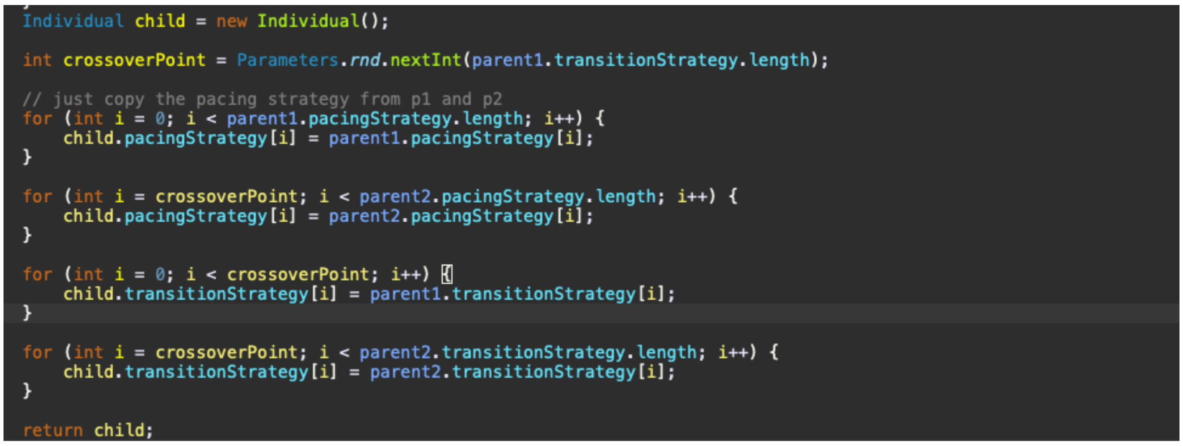
The default mutation only mutates the transition strategy of the offspring. Hence an experiment with both transition and pacing strategy of the offspring was proposed.



```java
// mutate the transition strategy by flipping boolean value
for (int i = 0; i < mutationRate; i++) {
    int index = Parameters.rnd.nextInt(child.transitionStrategy.length);
    child.transitionStrategy[index] = !child.transitionStrategy[index];
}
// mutate the pacing strategy by randomly adding/taking pacing.
for (int j = 0; j < mutationRate; j++) {
    int index = Parameters.rnd.nextInt(child.pacingStrategy.length);
    double step = Math.abs(Parameters.rnd.nextGaussian());
    while (step > 1)
        step = Math.abs(Parameters.rnd.nextGaussian());
    if (Parameters.rnd.nextFloat() < 0.5) {
        if (Math.toIntExact(Math.round(child.pacingStrategy[index] * (1 + step))) <= 1200)
            child.pacingStrategy[index] = Math.toIntExact(Math.round(child.pacingStrategy[index] * (1 + step)));
        else
            child.pacingStrategy[index] = Math.toIntExact(Math.round(child.pacingStrategy[index] * (1 - step)));
    } else {
        if (Math.toIntExact(Math.round(child.pacingStrategy[index] * (1 - step))) >= 200)
            child.pacingStrategy[index] = Math.toIntExact(Math.round(child.pacingStrategy[index] * (1 - step)));
        else
            child.pacingStrategy[index] = Math.toIntExact(Math.round(child.pacingStrategy[index] * (1 + step)));
    }
```

Figure 3. Mutation Operator Optimisation

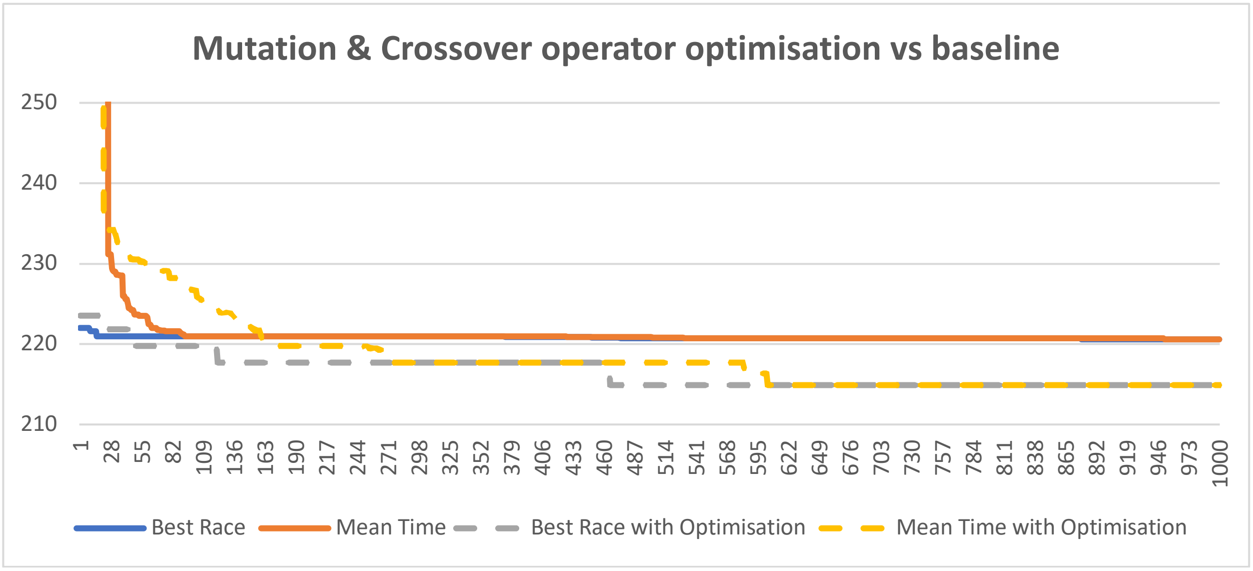
Regarding the crossover operator, additional genetic input from both parents were used to modify the offspring's pacing strategy as opposed to the default library which only makes use of one parent's genetic information.



```java
Individual child = new Individual();

int crossoverPoint = Parameters.rnd.nextInt(parent1.transitionStrategy.length);

// just copy the pacing strategy from p1 and p2
for (int i = 0; i < parent1.pacingStrategy.length; i++) {
    child.pacingStrategy[i] = parent1.pacingStrategy[i];
}

for (int i = crossoverPoint; i < parent2.pacingStrategy.length; i++) {
    child.pacingStrategy[i] = parent2.pacingStrategy[i];
}

for (int i = 0; i < crossoverPoint; i++) {
    child.transitionStrategy[i] = parent1.transitionStrategy[i];
}

for (int i = crossoverPoint; i < parent2.transitionStrategy.length; i++) {
    child.transitionStrategy[i] = parent2.transitionStrategy[i];
}

return child;
```

Figure 4. Crossover Operator Optimisation

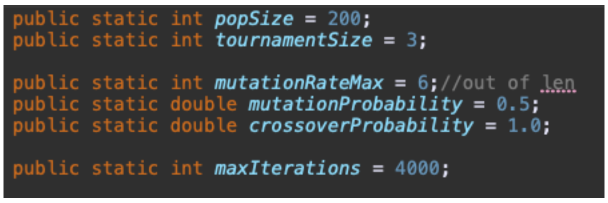
The result of this experiment is as follow:



As seen from the illustration, without the mutation and crossover operator, the search algorithm prematurely converged at about 85 iterations while the optimisation shows multiple breakaways from local optimum and stabilised at about 595 iterations with 214.9 seconds.

Parameters Tuning

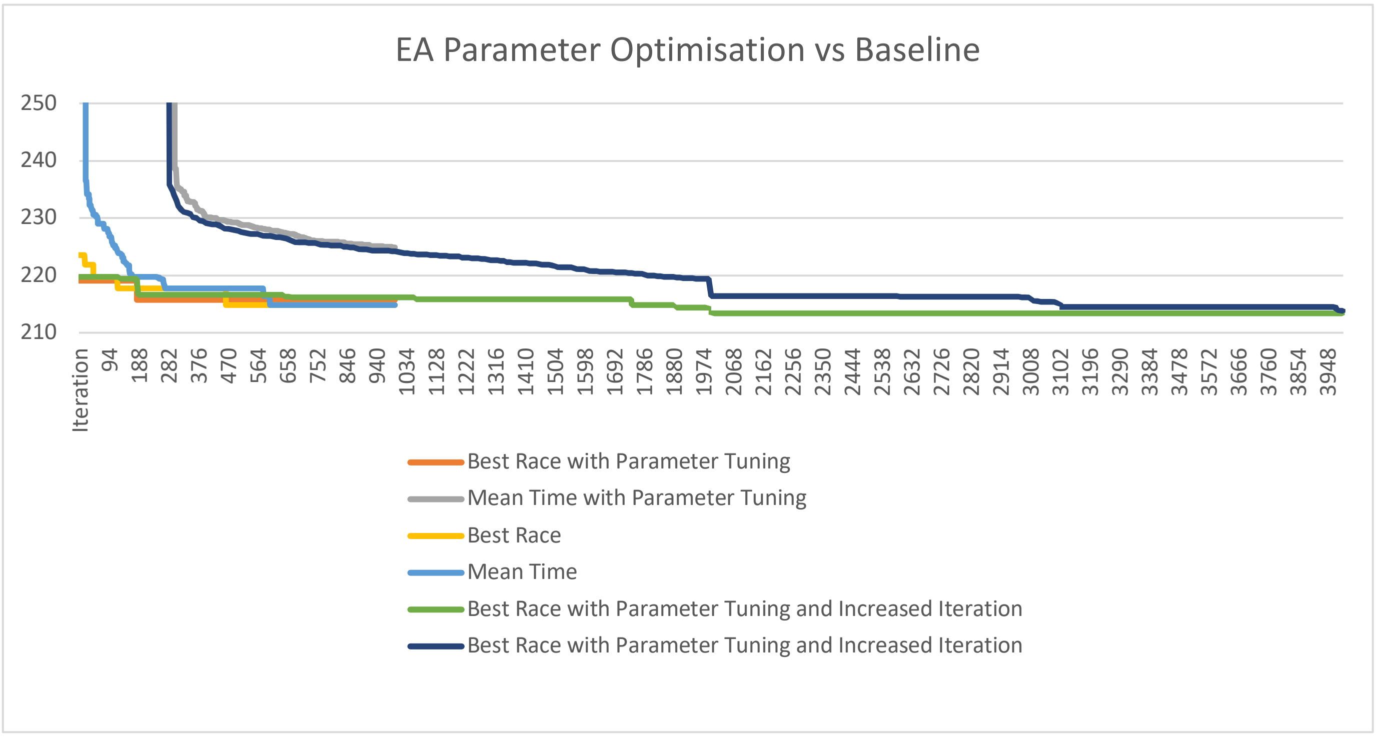
Further optimisation was performed by increasing the tournament size to force the chromosomes to have to compete with more fitter competitions. The population size was also increased to reduce the selection pressure to some extends and avoid premature convergence. However, as the experiment was conducted, it was found that simply increasing the population size and tournament size not only did not help with the overall performance, it also resulted in a poorer solution. It has come to attention that that might be due to the search algorithm had not converged. Hence a further tuning on maximum iteration was performed.



```
public static int popSize = 200;
public static int tournamentSize = 3;

public static int mutationRateMax = 6;//out of len
public static double mutationProbability = 0.5;
public static double crossoverProbability = 1.0;

public static int maxIterations = 4000;
```

Figure 5. EA Parameter Optimisation

The result of this experiment is as follow:



As can be seen from the illustration above, with the increased iteration number, the search algorithm slowly escaped the plateaus and converged at about 4000 iterations.

IV. Conclusions (10 marks)

As from the experiments, the diversified initial population acted as a seed for a wider pool of solution while mutation and crossover operators acted as a barrier to early convergence by increasing the diversity of the gene pool. Last but not least, increased population size, tournament size and the number of maximum iterations allowed the evolutionary algorithm to perform at its optimum capacity as an emergent computing technique for optimisation.

To conclude the experiment, the finding is detailed as follow:
Best race time achieved:
- 213.37800000001405 (seconds)

Best transition strategy:
- [true,true,true,true,false,true,false,true,true,true,true,false,true,true,true,false,false,false,true,false,false,true]

Best pacing strategy:
- [488,324,578,369,366,283,555,327,397,325,373,510,371,457,488,599,325,325,696,422,325,540,325]

Best EA parameter set:
- Population size: 200
- Tournament size: 3
- Mutation Rate Max: 6
- Mutation Probability: 0.5
- Crossover Probability: 1.0
- Maximum Iteration: 4000

V. Future Work (5 marks)

In retrospective, further improvements and research on a selecting mechanism should also improve the EA's efficient in solving trivial challenges. Also, other parameters can be considered and optimised such as mutation rate and probability.

Word Count: 1685.

VI. Works Cited

Marks, P., 2007. Evolutionary algorithms now surpass human designers. New Sci.

Ramadan, S., 2012. Reducing Premature Convergence Problem in Genetic Algorithm: Application on Travel Salesman Problem. Comput. Inf. Sci. 6, p47. https://doi.org/10.5539/cis.v6n1p47

Wagner, M., Day, J., Jordan, D., Kroeger, T., Neumann, F., 2013. Evolving Pacing Strategies for Team Pursuit Track Cycling, in: Di Gaspero, L., Schaerf, A., Stützle, T. (Eds.), Advances in Metaheuristics. Springer New York, New York, NY, pp. 61–76. https://doi.org/10.1007/978-1-4614-6322-1_4