Nature Inspired Computation Report

Charlie Wilkinson

November 17, 2023

1 Introduction

Evolutionary Algorithms (EA) have become popular tools in finding solutions to optimisation problems. Most EA's follow a particular framework, whereby they create an initial population of random solutions to a given problem, which are then evaluated by an objective function. The solution is then evolved to either minimise or maximise the set objective (Mirjalili, S., 2019). EA's provide a unique solution to the need of finding an automated solution applicable to a wide range of problems, with an increasing degree of complexity (Eiben and Smith, 2015).

In this report, we focus upon solving the optimisation problem known as the 'Travelling Salesman Problem' (TSP). The problem itself involves a salesman travelling around an n number of cities, with each distance having a cost. We design our EA to identify the route that satisfies our optimality condition. For the TSP, this means discovering the most cost-effective journey (Eiben and Smith, 2015).

The main issue in finding an optimal solution is due to the scale of the problem, with an (n - 1)!/2 number of tours (Razali and Geraghty, 2011). In this report, we will trial a number of different techniques with the aim of finding an optimal solution within a time constraint.

2 Design

2.1 Data

Our analysis is focused upon two sets of data, which have been stored in two separate XML files. The files are structured as a hierarchy, with each city element containing the associated cost of making a trip to every other city. The first file, 'burma14' includes cost information for 14 unique cities, while 'brazil58' includes 58 different cities.

2.2 Initial Population

The first step in building our EA is to represent the problem. A valid solution to the TSP is one that follows a set of rules: The salesman must visit each city once, they cannot return to a previously visited city and they must return to the starting city to make a full loop. Abiding by these rules, the TSP is an order-based problem, whereby the solution is dependent on the order in which the salesman travels to each city. A permutation representation is most appropriate for our set of solutions. We represent each route as a set of integers, where each integer is a city and only occurs once (Eiben and Smith, 2015).

Another interesting aspect of the problem is that the starting point of the route is insignificant. The third rule specifies a route must end at the original city, meaning a route may start and end anywhere. I later leverage flexible start points during the mutation phase to retain diversity within my population

2.3 Fitness Function

The optimisation goal of our EA is to find the cost minimising route. We calculate the cost of a given route using the following formula:

$$cost = \sum_{i=1}^{n-1} D[C[i]][C[i+1]] + D[C[n]][C[1]]$$

Figure 2.1: The Cost of a Given Route

Broken into its key elements, figure (2.1) calculates the total cost of a route as the sum of a n number of edges, where each edge calculates the cost of travelling from one city to the next. The final cost is thus the total cost from of a full loop, back to the starting city C[1].

3 Results

In this section, we aim to investigate the effects of altering different parameters in our model. To do so, we experiment with altering each evolutionary operator independently. The base model evaluates 10000 fitness functions, using the following parameters:

- Tournament size - 5
- Population size – 100
- Number of swaps – 0

- Crossover Operator – Single with fix
- Replacement Strategy – Replace weakest

To strengthen the robustness of our results, we include ten iterations for each EA. The results we collect include the best solution acquired for each iteration, and which generation the solution first appeared in.

3.1 Selection

Selection takes inspiration from Darwin's theory of natural selection, whereby the fittest individual has the greatest chance of passing on it genes to the next generation (Mirjalili, S., 2019). In our genetic algorithm, we implement a tournament-based selection. Focusing on relative fitness over absolute, this operator selects a random n amount of routes from our population, compares their relative cost and returns the route with lowest cost. We run this procedure twice to obtain two parents for further mutations.

The size of the tournament is paramount to its effectiveness. If the number of routes, n, is too large, the higher the expected loss of diversity and our algorithm may tend to a local optimum. Too small, and the algorithm may harm convergence speed (Razali and Geraghty, 2011). We investigate four different values: two, five, ten and twenty.

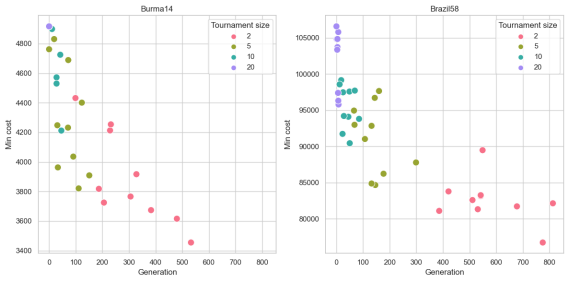

Figure 3.1: Adjustments to the Tournament parameter

The results for 'burma14' and 'brazil58' are both very similar. As anticipated, the largest tournament size, 20, prematurely converges to a suboptimal point. The trend indicates for both data sets that as n falls, the EA generally finds better solutions but takes more generations to do so. Therefore, when selecting the tournament size, it is imperative to consider the trade-off between solution quality and speed.

Another key feature of the results is the variability across the iterations. This variability is more pronounced in the latter plot; however, both plots illustrate a similar trend. When n decreases, there is a noticeable decrease in the concentration of data points associated with the same tournament size. For instance, the largest tournament size (n=20) displays greater concentration of data points, which suggests the population suffers from a lack of diversity, resulting in a narrower search. Meanwhile the smallest (n=2) has a broader spread of data points, signifying a wider exploration across the search space and the discovery of diverse peaks. This distinction showcases how the choice of tournament size influences the algorithm's exploration-exploitation trade-off, affecting its ability to discover and exploit various solutions within the search space.

3.2 Mutations

Mutation operators involve the modification of one or more genes in the child solutions. Mutations play a key role in avoiding the premature convergence to a local optimum, by restoring lost or unexplored genetic material into the population. They ensure no important features are prematurely lost and maintains diversity within the population (Deep and Mebrahtu, 2011a).

We include the most basic mutation, the swap mutation. It involves selecting two random positions within a child encoding and swapping the values. We later evaluate the effect of adjusting the number of swaps, k, on performance.

The other mutation we use is an inverse swap. This randomly select two points within the route, and then reverses the order in which the values appear between these positions. The mutated route is thus made up of three parts, each maintaining the connections between cities. Only the links between the first and middle part, and the middle part and the last part are broken (Eiben and Smith, 2015).

We analysed five different EAs, each with a different swap parameter: Inverse swap, no swap, one swap, two swaps and five swaps.

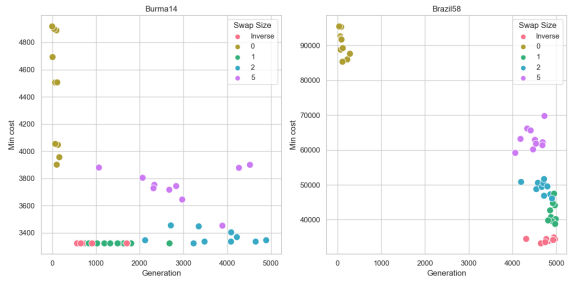

Figure 3.2: Adjustments to the Mutation parameter

Both scatter plots imply that the worst performing EA is the one without a swap mutation, which suffers from an early convergence to a sub-optimal peak. The rankings of best-performing EA's is shared between the two data sets, with the inverse swap achieving the most optimal results.

A key difference between the plots is the differing degrees of variation. In the 'burma14' plot, the results for the EA's employing 1, 2 and 5 swap mutations exhibit higher variability. They generally reach the same minimum cost, but do so over a greater range of iterations. In contrast, the latter plot indicates a greater concentration and clustering among different types of EA. In this case, its evident that regardless

of the number of swaps (excluding zero), each EA engages in a broader exploration of the solution space, which suggests that the swap mutation is successful in maintaining a higher degree of diversity within the population. Therefore the inverse swap is the most successful at striking a balance between randomness and preserving a routes important features.

3.3 Crossovers

A crossover involves combining the genes from two parents to create two new 'child' solutions. We install two types: one-point crossover with a fix and ordered.

The one-point crossover splits both parents at a point that has been randomly generated in the range of the length of the encoding. The child are thus the two combinations by stitching the a part from each parent together. A flaw within this design is that a child encoding may visit the same city twice and miss out a city altogether. To avoid this, we implement a fix, which replaces duplicate values from the main parent with the necessary cities to complete a full loop.

Unlike the one-point crossover, the ordered crossover aims to preserve the relative position of the second parents encoding. It takes a section of a route from one parent and fills the rest of the encoding using the relative order of cities from the other parent, ensuring every city is visited only once (Deep and Mebrahtu, 2011b).

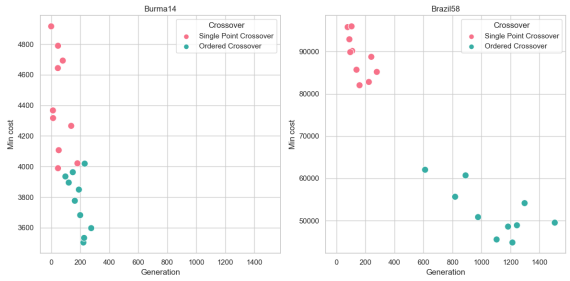

Figure 3.3: Adjustments to the Crossover parameter

The results for both data sets consistently demonstrate a substantial performance enhancement for the ordered crossover operator. The gap between the two types is particularly pronounced for the larger data set. The observed trend strongly indicates that the single-point crossover operator tends to converge prematurely, limiting its exploration capability within the solution landscape. Meanwhile, the ordered crossover operator displays a more comprehensive search across the landscape, resulting in the discovery of more optimal solutions that outperform those obtained compared to its counterpart.

3.4 Replacement Function

In the replacement stage of an EA, solutions within the population are replaced by the offspring. Common replacement strategies often discriminate based upon a solutions fitness or age (Eiben and Smith, 2015). In our EA, we exclusively focus upon those of the fitness-based categorisation. One approach replaces based upon the worst solutions in the population. This strategy is effective for rapidly improving solutions; however, it may lead to premature convergence (Eiben and Smith, 2015). To address this, we also utilise a 'replace first weakest' strategy, which iterates through the population, and replaces the first solution for which each of our offspring is an improvement.

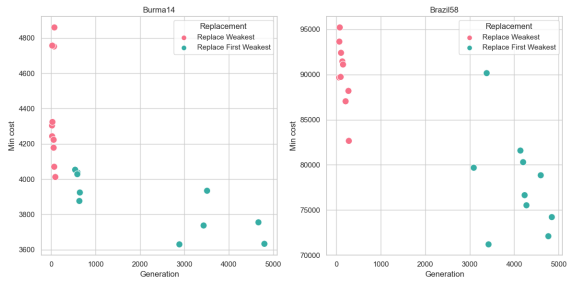

Figure 3.4: Adjustments to the Replacement function

The observed trends align with our predictions, whereby the 'replace weakest' strategy leads to a premature convergence, and the 'replace first weakest' strategy engages in a greater exploration of the solution landscape and displays greater variation. Comparing performance between the two data sets, in 'burma14' the 'replace weakest' strategy finds solutions akin to its counterpart but achieves them at a much quicker rate. However, in the larger data set, 'replace first weakest' strategy consistently outperforms, with all iterations finding lower costs.

4 Conclusion

4.1 Optimal EA

In this section, we utilise our previous findings to create an optimal EA. We evaluate four different designs, each with a unique purpose:

- EA1 - Design taking every parameter that performed the best in the last section
- EA2 - Design taking every parameter that performed the worst in the last section
- EA3 - Design that combines features from EA1 and EA2
- EA4 - Design that combines features from EA1 and EA2

The reason behind our choices is that to explore how the different parameter combinations affect performance. To strengthen the robustness of our findings, our initial population is generated from a range of different seeds.

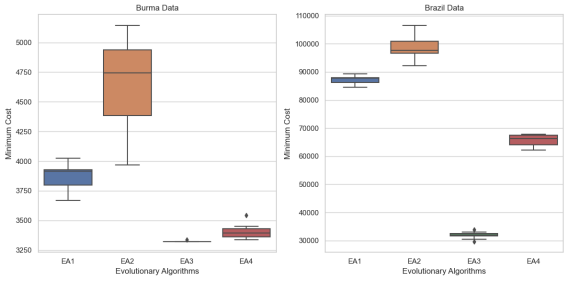

Figure 4.1: Results of different EA's

The results indicate that EA3 is the best performing evolutionary algorithm for both of the data sets, and consistently achieves the lowest cost. It is designed to maintain diversity within the population incorporating features such as ordered crossover and inverse swap whilst avoiding a pure random search, employing a higher tournament size of ten and a 'replace weakest' strategy. It is interesting to note that EA4, another combined EA, lags behind in the latter plot, which could signify the importance of an ordered crossover over its single point counterpart.

Moreover, these findings underscore the drawbacks of failing to balance the exploration-exploitation trade-off. EA1 is designed to find a solution quickly, while EA2 is designed to search for the best solution. Both consistently produce sub-optimal solutions within the 5000-generation span due to their respective designs, and highlight the need for a balanced approach.

4.2 Discussion

In this report, I explored the impact of various operators within an Evolutionary Algorithm. Initially, I conducted a comprehensive evaluation of different operators to assess their individual efficacy, and subsequently designed multiple EA's utilizing combinations of these operators. Upon assessment, it became evident that an EA achieving a delicate balance between maintaining diversity within the population and actively seeking solutions yielded the most promising results.

To optimise my EA, I could have utilised the population size control more effectively. Throughout my analysis, I maintained a constant population of 100, which may have hindered the search for more optimal solutions. Increasing this number would have resulted in a larger and more diverse population. On the other hand, reducing the population size would have increased the focus on finding optimal solutions. Therefore, varying population size may be a useful tool in optimising results.

Another method that could improve the performance of an EA is to implement a local heuristic. If we consider EA2, a local heuristic may have helped the algorithm locate optimal solutions by reducing the randomness, and considering local information to guide exploration. One such method is the 2-opt procedure. This method builds upon a given route, by removing two replacing them in such a way that reduces the cost. This effectively explores the local landscape of a given route, and can potentially find a superior solution in its neighbourhoods (Uddin, 2023).

Outside of our EA, Ant Colony Optimisation (ACO) is an alternative method to solve the TSP. ACO is a swarm intelligence technique, often used for solving combinatorial optimisation problems. Research such as Mirjalili (2018) found that this method is undoubtedly beneficial in solving the TSP, with the 'ants' finding one of the shortest paths in the 'kroA100' data set. These results also suggested that ACO benefited from a consistent convergence rate, and dramatically improved the initial solution.

5 Bibliography

Deep, K. and Mebrahtu, H., 2011a. Combined mutation operators of genetic algorithm for the travelling salesman problem. International Journal of Combinatorial Optimization Problems and Informatics, 2(3), pp.1-23.

Deep, K. and Mebrahtu, H., 2011b. New variations of order crossover for travelling salesman problem. International Journal of Combinatorial Optimization Problems and Informatics, 2(1), pp.2-13.

Eiben, A.E. and Smith, J.E., 2015. Introduction to evolutionary computing. Springer-Verlag Berlin Heidelberg.

Mirjalili, S., 2019. Evolutionary algorithms and neural networks. In Studies in computational intelligence (Vol. 780). Berlin/Heidelberg, Germany: Springer.

Razali, N.M. and Geraghty, J., 2011, July. Genetic algorithm performance with different selection strategies in solving TSP. In Proceedings of the world congress on engineering (Vol. 2, No. 1, pp. 1-6). Hong Kong, China: International Association of Engineers.

Uddin, F., Riaz, N., Manan, A., Mahmood, I., Song, O.Y., Malik, A.J. and Abbasi, A.A., 2023. An Improvement to the 2-Opt Heuristic Algorithm for Approximation of Optimal TSP Tour. Applied Sciences, 13(12), p.7339.

Mirjalili, S. and Mirjalili, S., 2019. Ant colony optimisation. Evolutionary Algorithms and Neural Networks: Theory and Applications, pp.33-42.