Performance Comparison of Simulated Annealing and Genetic Algorithm Solivng Odyssey Travelling Salesman Problem

Jack Mortimer

February 2023

1 Introduction

A Travelling Salesman Problem (TSP) is the question 'Given a set of cities, what is the shortest route which visits every city exactly once and returns to the start?'. Such a question formally amounts to the finding a Hamiltonian cycle in a weighted complete graph of a set of vertices. The Odyssey TSP is based on Ulysses' adventures described in Homer's epic the Iliad. Historians have identified a number of cities alleged to have been visited by Ulysses during his travels. What is the most efficient route that Ulysses could have sailed? This question makes for an interesting toy problem, however there many applications, particularly in logistics and planning, which make then ability to produce solutions enticing.

TSPs are highly combinatorial. The original publication of this problem by Grötschel and Padberg (2001) names 16 cities, to enumerate all possible routes would take a considerable amount of time and is impractical or even impossible for larger problems. The worst case time complexity for brute force approaches is $\mathcal{O}(n!)$. There are a number of improved brute force techniques such as branch-and-bound, but it is often necessary to apply heuristics to achieve sub-optimal but 'good' solutions.

This report discusses the implementation and comparative performance of two such heuristic methods, Simulated Annealing (Kirkpatrick, Gelatt, and Vecchi 1983) and Genetic Algorithm (Holland 1992), with a 22 city problem based on the Odyssey TSP. We will first describe the algorithms, implementation and design choices, and then perform statistical analysis on the results of each. A key requirement that was considered when designing the implementations was a limit of 10,000 fitness calculations for each run of the algorithms.

2 Simulated Annealing

Simulated Annealing is an algorithm inspired by the physical process of annealing in metallurgy, where a metal is heated to a high temperature and gradually cooled. The process consists of first 'melting' the system with a high temperature parameter, T, and slowly cooling the temperature according to an annealing schedule until the system reaches convergence and no further changes occur (Kirkpatrick, Gelatt, and Vecchi 1983). This annealing schedule may be defined by the user according to the requirements of the problem at hand.

Solutions take the form of a permutation of the list of cities in the problem. With each iteration we generate a 'neighbour' solution x_{new} using local search. We then compute the objective function (energy); in the case of TSPs this is simply the total distance of the route.

$$e_{new} = f(x_{new})$$

We then calculate $P(e, e_{new}, T)$, where e is the objective evaluation of the current solution, as

$$P(e, e_{new}, T) := \begin{cases} 1 & \text{if } e_{new} < e \\ \exp(\frac{e - e_{new}}{T}) & \text{otherwise.} \end{cases}$$

If $e > e_{new}$ then we move to the new solution, and otherwise generate some random number, r, in range $(0, 1)$ and compare $P(e, e_{new}, T)$. If it is greater than r then we move to this solution. Thus, with some probability proportional to T we may search neighbours that are 'worse' but may lead to greater improvements, avoiding local minima. The algorithm returns the best found solution over all searched, x_{best}. Pseudocode for Simulated Annealing is shown in algorithm 1.

Algorithm 1 Simulated Annealing (He 2023a)

```
procedure SimulatedAnnealing($x$)
    $x := x_0$; $e := f(x)$                              ▷ Initialise random solution
    $x_{best} := x$; $e_{best} := e$                     ▷ Initialise best found solution
    $T := t_0$
    $k := 0$
    while $k < k_{max}$ do                               ▷ Repeat procedure until max iterations is reached
        $T \leftarrow temperature(T)$                    ▷ Update T according to annealing Schedule
        $x_{new} \leftarrow neighbour(x)$                ▷ Find neighbour solution
        $e_{new} \leftarrow f(x_{new})$
        if $P(e, e_{new}, T) > R(0, 1)$ then             ▷ Decide if move to solution
            $x \leftarrow x_{new}$; $e \leftarrow e_{new}$
        end if
        if $e_{new} < e_{best}$ then                     ▷ Check if best found
            $x_{best} \leftarrow x_{new}$; $e_{best} \leftarrow e_{new}$
        end if
        $k \leftarrow k + 1$
    end while
    return $x_{best}$                                    ▷ Return best found solution
end procedure
```

Simulated Annealing was implemented in Python for the Odyssey TSP. There were a number of design considerations made to suit the specifics of the problem and achieve good performance. The primary considerations were the annealing schedule and local search method for generating neighbours.

The selected annealing schedule was defined as follows:

$$T_{k+1} = T_k \cdot (1 - \frac{\alpha}{K})$$

where $0 < \alpha < K$ is a cooling parameter affecting the rate at which the temperature is reduced, and N is the maximum number of iterations. Higher values lead to a greater rate of cooling. It

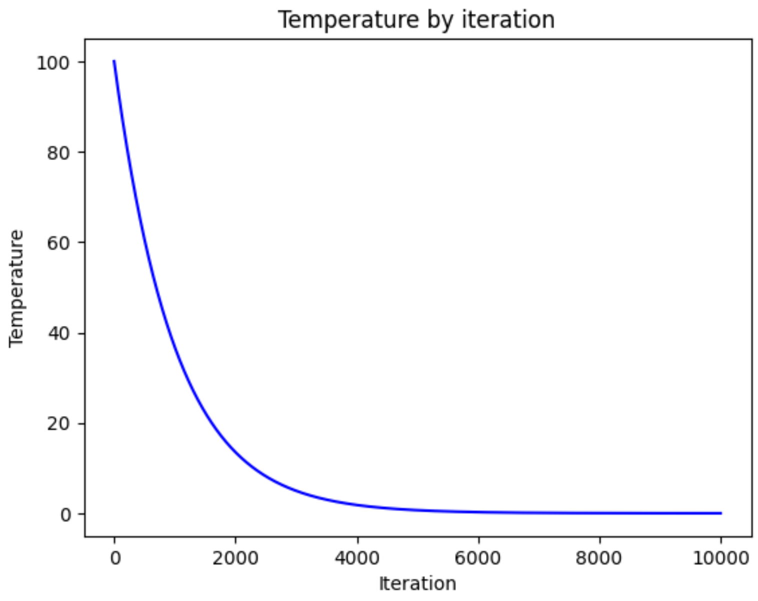

Figure 1: Curve showing temperature reduction by iteration for $10,000$ iterations of Simulated Annealing, with $\alpha = 10$.

was found that this function with $\alpha = 10$ lead to an initially fast rate of cooling which slowed at a smooth rate before converging to 0. As the temperature approaches 0, the algorithm becomes simple hill descent and converges to a minimum value. This allowed sufficient time for initial exploration and convergence to good solutions, leading to excellent performance. The temperature curve is shown in figure 1.

To find neighbouring solutions the user must make a choice of local search algorithms. A popular choice is 2-opt, which has been used for this problem as it is suitable and effective for TSPs. We Generate two random cities in the route and reverse the order of the cities between them. This results in an immediate neighbour.

Algorithm 2 2-opt (He 2023a)

```
procedure 2-OPT($x$)
    Generate $c_1, c_2$ in range(len($x$)) such that $c_1 \neq c_2$
    add $[x[0] : x[c_1]]$ to $x_{new}$
    Reverse order of $[x[c_1 + 1] : x[c_2]]$ and add to $x_{new}$
    add $[x[c_2] : x[-1]]$ to $x_{new}$
    return $x_{new}$
end procedure
```

A diagram of a 2-opt operation on a solution for a very simple four city TSP is shown in figure 2.

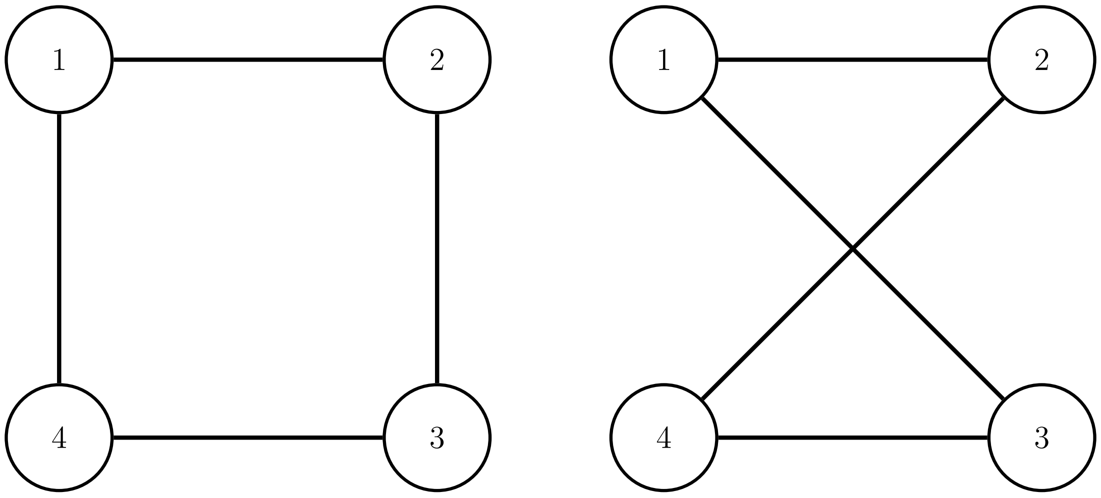

Figure 2: 2-opt procedure on a four city TSP solution. After selecting cities 3 and 4, reverse the traversal order between them.

3 Genetic Algorithm

Genetic Algorithms were initially proposed by Holland (1992), imitating natural selection and evolution in nature. We begin with a population of candidate solutions, **X**, selecting parents with the best fitness to be mutated and carried over to the next generation.

Algorithm 3 Genetic Algorithm (Claussen and He 2023)

```
procedure GeneticAlgorithm($\mathbf{X}$)
    $\mathbf{X} := \mathbf{X_0}$
    $k := 0$
    Evaluate fitness of individials in $\mathbf{X_k}$
    $x_{best} \leftarrow \text{argmin}_{x \in \mathbf{X}} f(x)$
    while $k < k_{max}$ do
        Parents $\leftarrow$ Selection($\mathbf{X_k}$)
        Children $\leftarrow$ Mutation(Parents)
        Evaluate fitness of new individuals
        $\mathbf{X_{k+1}} \leftarrow$ Generate(Parents, Children, $\mathbf{X_{k+1}}$)
        $x_{new} \leftarrow \text{argmin}_{x \in \mathbf{X_{k+1}}} f(x)$
        if $f(x_{best}) > f(x_{new})$ then
            $x_{best} \leftarrow x_{new}$
        end if
        $k \leftarrow k + 1$
    end while
    return $x_{best}$
end procedure
```

Much of a Genetic Algorithm is left to the user to tailor to the specific needs of the problem. Design choices must be made about the size of the population, method and number of parent selection, methods and number of mutations, and how each new generation is produced. It is also essential to select a suitable encoding scheme to represent the problem, which can have a definite effect on performance.

A Genetic Algorithm was also implemented in Python for the Odyssey TSP. It uses a population size of 50, with 20 parents selected and each produces a single child. The remaining 10 in the new generation are a uniformly random selection from the previous. The encoding scheme was simply permutations of city coordinates as in Simualted Annealing as this is suitable and effective for TSPs.

To select the parents we use linear ranked selection. It was found by Razali and Geraghty (2011) that linear ranking lead to a performance advantage over other popular methods tournament and proportional selection for TSPs. The population is ordered by objective evaluation from highest to lowest and given a rank, γ, beginning at 0 for the worst individual and $N-1$ for the best. We then calculate their selection probabilites according to:

$$p(\gamma) = \frac{\alpha + (\beta - \alpha) \cdot \frac{\gamma}{N-1}}{N}$$

where $0 \leq \beta \leq 2$ and $\alpha + \beta = 2$. N is the population size. This leads to the expectation that the best individual is produced β times, with $p(N-1) = \frac{\beta}{N}$. The worst individual is expected to be produced α times, with $p(0) = \frac{\alpha}{N}$ (He 2023b). This balances exploration and exploitation, allowing poor individuals to be selected with small probability to avoid local minima. For the selection of parameters α and β, we use $\alpha = 0$ and $\beta = 2$. It was found that this lead to the best performance of Genetic Algorithm, possibly due to the problem being relatively small as well as a required limitation on the maximum number of generations leading to this extreme. While still allowing exploration, poor candidates are selected with small probability compared to those with best fitness, favouring exploitation.

There are a wide selection of mutation operators that one could choose from, however the chosen operator is to again use 2-opt to generate children. This displayed high efficacy when used for local search in simulated annealing, and is simple to implement. 2-opt does not deviate too dramatically from good solutions as the mutations lead to an immediate neighbour which may be advantageous in small problems where changes to the individuals may lead to proportionately large differences in fitness, which could cause trouble in convergence to good solutions.

The new generation is composed of the selected parents, children and the remaining candidates are randomly selected with uniform probability from the previous generation. This allows for more opportunity for exploration while still favouring exploitation of good candidates for mutation.

4 Performance Comparison

Both algorithms demonstrated ability to find good solutions to the Odyssey TSP. There was, however, an apparent difference in performance favouring Simulated Annealing. It also ran faster than Genetic Algorithm and was thus more efficient. Due to the design constraints, Simulated Annealing was run with 10,000 iterations (one fitness calculation per iteration) and Genetic Algorithm with 497 generations (50 initially, plus 20 per generation).

First we look at the characteristics of Simulated Annealing Genetic Algorithm from a random single run of each. Simulated Annealing initially has a high amount of exploration, with the search trajectory, shown in figure 3, erratic as the algorithm explores poor solutions in hope of future gains. As the temperature reduces the magnitude of fluctuations in the search trajectory reduces and the algorithm begins to converge to a good solution, as it approaches hill descent.

This results in a steady improvement to the best distance found. The improvement over time can be seen in figure 4. The improvements appear to be greater to begin with, before becoming

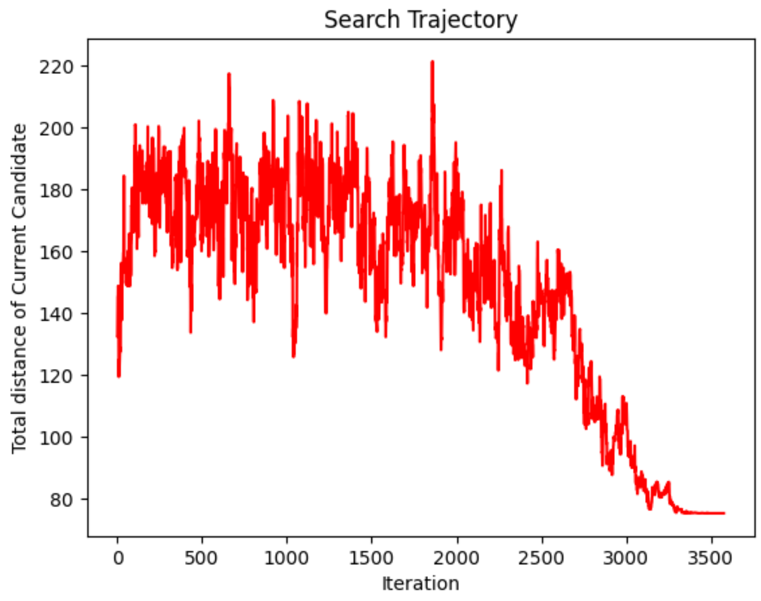

Figure 3: Search trajectory of a Simulated Annealing run, showing the total distance of the current candidate solution

more refined which correlates with the expected behaviour of the algorithm as the temperature is reduced. This was a particularly successful test of Simulated Annealing, finding an optimal solution with a distance of 75.309. The resulting route is shown in figure 5.

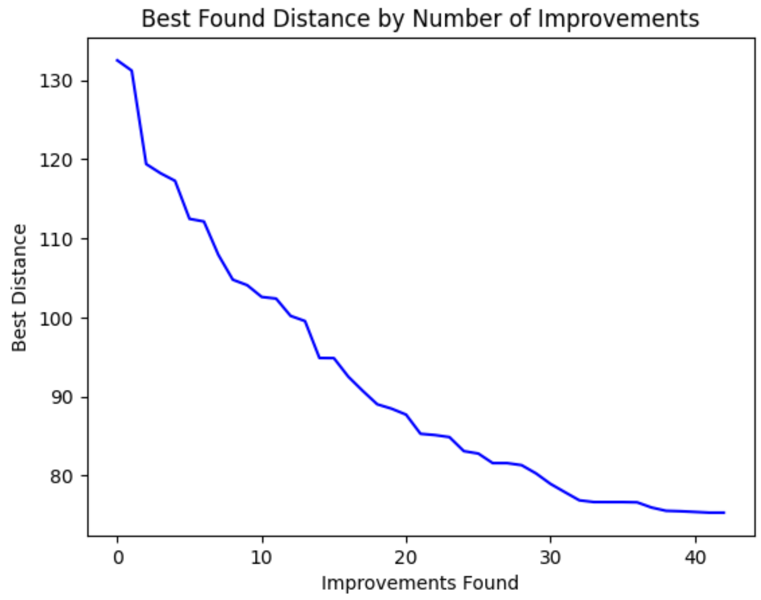

Figure 4: Improvements found by Simulated Annealing

Genetic Algorithm comparatively produced a satisfactory but less optimal result. The search trajectory, shown in figure 6, shows rapid improvement within the first 100 generations, although this slows and while Genetic Algorithm continues to explore it finds few improvements. This may indicate that the design of the Genetic Algorithm is susceptible to local minima.

The Genetic Algorithm appears to make improvements of a more linear magnitude as shown

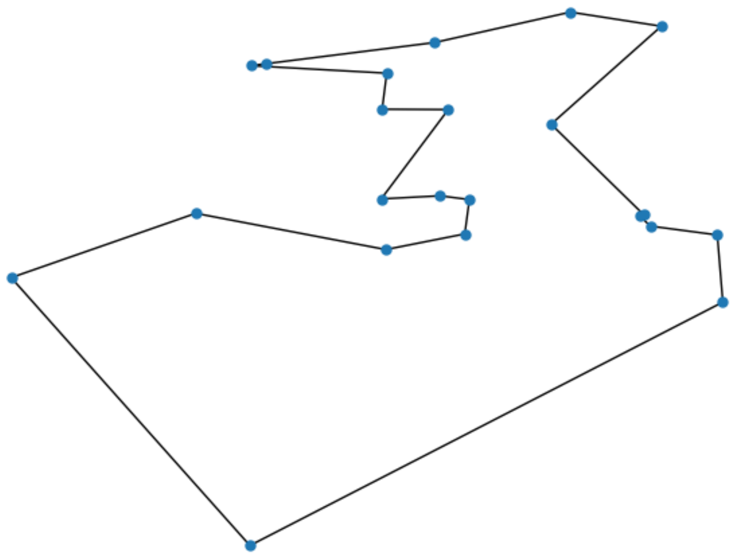

Figure 5: Best route found by Simulated Annealing

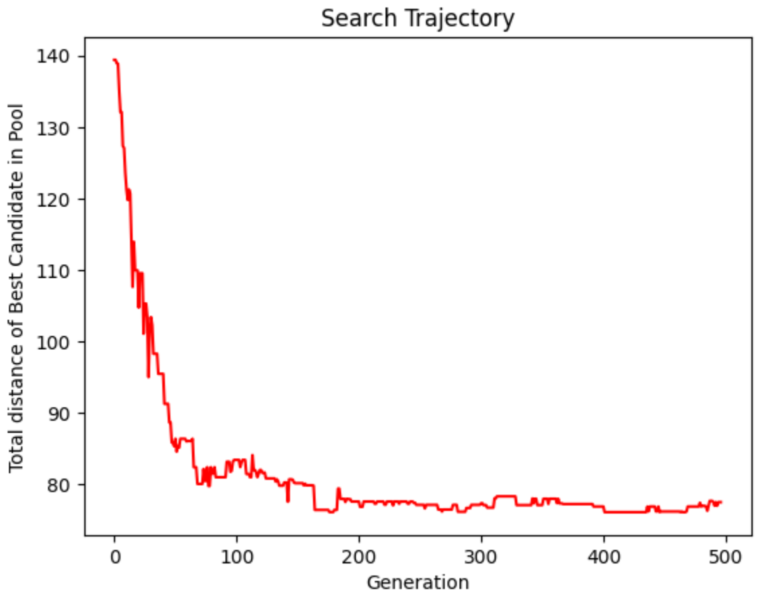

Figure 6: Search trajectory of Genetic Algorithm

in figure 7, however this may be due to the relatively low number found due to the local minima. The solution produced has a total distance of 76.065 and is shown in figure 8.

From a sample of 30 runs, which can be found in appendix B, Simulated Annealing completed in a mean time of $0.458s$, whereas Genetic Algorithm completed in a mean time of $0.734s$. This may be due to the specific implementation in Python, or the number of calculations, sortings and searches that comparatively need to be performed per loop of Genetic Algorithm.

From the 30 test runs, Simulated Annealing had a mean distance of 75.949, and standard deviation of 0.423. Genetic Algorithm produced a mean distance of 76.729 and standard deviation

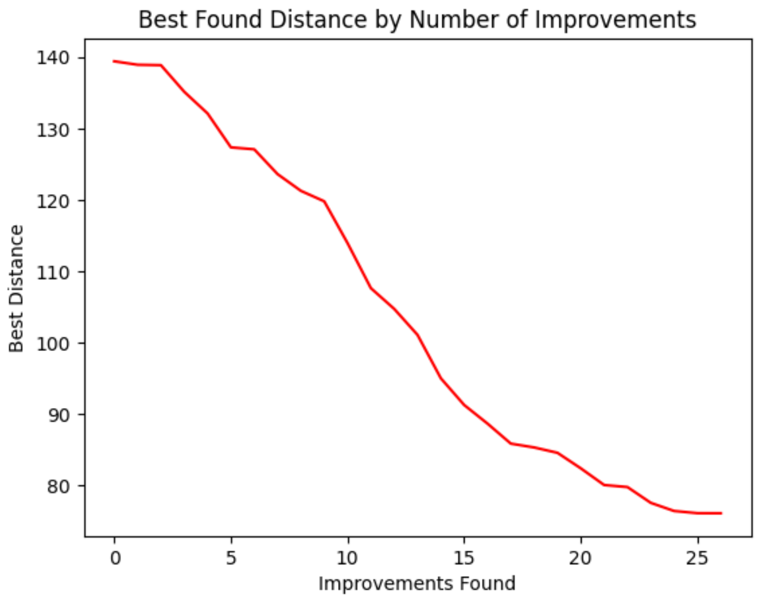

Figure 7: Improvements found by Genetic Algorithm

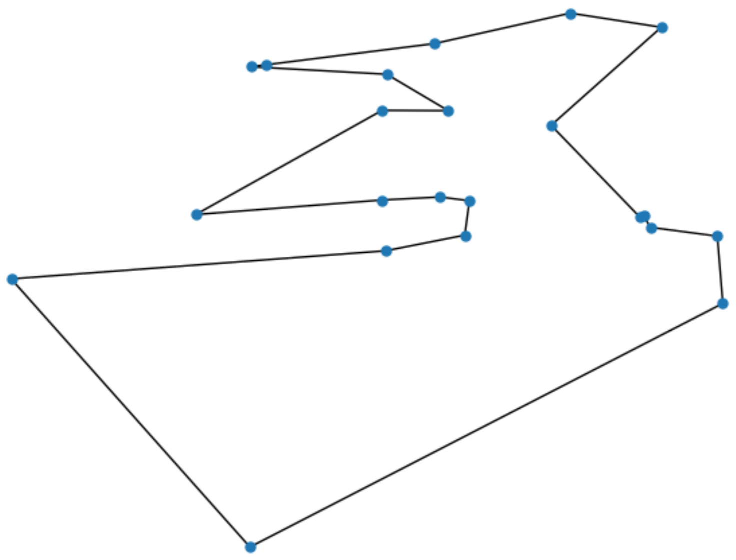

Figure 8: Best route found by Genetic Algorithm

of 0.708, indicating that while both Simulated Annealing and Genetic Algorithm produced good solutions, the annealing had better performance and a slight advantage in consistency with a smaller variation.

4.1 Wilcoxon Signed-Rank Test

To test the hypthosesis that Simulated Annealing produced significantly better results, a two-tailed Wilcoxon Signed-Rank Test was conducted (Wilcoxon 1945). We assume that the data is sampled from joint cumulative distirbution of pairs (X_i, Y_i), $F(x, y)$, where x is test sample from Simulated Annealing, and y from Genetic Algorithm. We hypothesise that F is not symmetric about $\mu = 0$, and test with a significance level of 5%, $\alpha = 0.05$.

$$H_0 : F(x, y) = F(y, x)$$
$$H_1 : \text{for some } \mu \neq 0, F(x, y) = F(y + \mu, x - \mu)$$

For each (X_i, Y_i) pair calculate the difference, $Z_i = X_i - Y_i$, and then order the differences by absolute value to achieve a ranking, R_i. We then calculate

$$T^+ = \sum_{0 \leq i \leq n,\ X_i > 0} R_i \tag{1}$$
$$T^- = \sum_{0 \leq i \leq n,\ X_i < 0} R_i \tag{2}$$

and the test statistic is then equal to $\min(T^+, T^-)$. We calculate the p-value as $p = 2.76 \cdot 10^{-6}$. $p < 0.025$ so reject the null hypthesis. We can conclude that with a significance level of 5%, results are not distributed uniformly about $\mu = 0$, suggesting Simulated Annealing has significantly better performance.

5 Conclusions

Both algorithms displayed efficacy at producing good solutions to the Odyssey TSP. Both implementations were also reasonably efficient, running in very short time for the constrained number of fitness calculations, certainly a substantial improvement over brute force methods. However, we can conclude that for the given problem the designed implementation of Simulated Annealing was more suitable. The Genetic Algorithm displayed characteristics in search pattern and number of improvements that indicate it may be getting stuck in local minima. This could potentially be reduced by increasing the amount of exploration, although if excessive it may harm performance. It may also be possible to improve this by choice of mutation operator. Tailoring Genetic Algorithms to achieve a high level of performance can require a great deal of work and deep understanding of the problem at hand, and so the comparatively simple Simulated Annealing may be a better choice. An alternative approach which may use advantages of both algorithms to produce better results could be to explore a combined GA-SA algorithm as proposed by Deng, Xiong, and Wang (2021). This could potentially lead to better performance than either of Simulated Annealing or Genetic Algorithm alone.

References

[1] Jens Christian Claussen and Shan He. *Week 1 Lecture 3: Evolutionary Computation -Genetic and Evolutionary Algorithms.* Lecture Notes. Feb. 2023.

[2] Yanlan Deng, Juxia Xiong, and Qiuhong Wang. "A Hybrid Cellular Genetic Algorithm for the Traveling Salesman Problem". In: *Mathematical Problems in Engineering* 2021 (Feb. 2021), pp. 1–16. DOI: 10.1155/2021/6697598.

[3] Martin Grötschel and Manfred W. Padberg. "The Optimized Odyssey". English. In: *AIROnews* VI.2 (2001), pp. 1–7.

[4] Shan He. *Week 2 Lecture 2: Evolutionary Computation - Stochastic Local Search algorithms.* Lecture Notes. Feb. 2023.

[5] Shan He. *Week 3 Lecture 1: Evolutionary Algorithms - Selection and Reproduction.* Lecture Notes. Feb. 2023.

[6] John H. Holland. "Genetic Algorithms". In: *Scientific American* 267.1 (1992), pp. 66–73. ISSN: 00368733, 19467087. URL: `http://www.jstor.org/stable/24939139` (visited on 02/25/2023).

[7] Scott Kirkpatrick, Charles D. Gelatt, and Michelle Vecchi. "Optimization by Simulated Annealing". In: *Science* 220 (1983), pp. 671–680.

[8] Noraini Razali and John Geraghty. "Genetic Algorithm Performance with Different Selection Strategies in Solving TSP". In: World Congress on Engineering. Vol. 2. July 2011.

[9] Frank Wilcoxon. "Individual Comparisons by Ranking Methods". In: *Biometrics Bulletin* 1.6 (1945), pp. 80–83. ISSN: 00994987. URL: `http://www.jstor.org/stable/3001968` (visited on 02/26/2023).

Appendix

A GitHub Repository

The GitHub repository containing implementations of each algorithm and all files for this project can be found at: `https://github.com/JackMortimer/Odyssey-Travelling-Salesman-Problem`

B Test Data

Table 1: Simulated Annealing and Genetic Algorithm distances from test data of 30 runs.

Test	**SA**	**GA**
1	75.310	76.430
2	75.310	76.400
3	75.310	77.470
4	75.790	76.000
5	75.910	76.380
6	75.910	77.050
7	76.590	75.310
8	76.020	77.540
9	75.790	75.450
10	75.670	76.530
11	75.510	76.590
12	77.100	76.250
13	75.910	76.120
14	75.910	77.100
15	76.130	76.280
16	76.020	78.150
17	76.430	75.790
18	75.910	77.420
19	76.130	77.020
20	76.130	75.310
21	75.310	76.900
22	75.510	76.470
23	76.330	75.890
24	75.670	75.940
25	76.100	78.000
26	76.130	75.550
27	75.510	76.340
28	76.590	76.570
29	76.100	75.970
30	76.430	77.440