Solving Travelling Salesman Problem ATT48 with Simulated Annealing and Genetic Algorithm

Cheuk Yu Lam (2250728)
BSc Computer Science
College of Engineering and Physical Sciences
1 March 2024

Keywords: TSP, ATT48, SA, GA, etc.

ABSTRACT

This report presents a comparative study of two optimisation techniques, Simulated Annealing (SA) and Genetic Algorithm (GA), in their application to solving the Travelling Salesman Problem (TSP) for the 48 contiguous US state capitals, utilising the TSPLIB dataset— a collection of benchmark TSP instances and related problems from a variety of sources and of multiple types[1]. Both algorithms are robust heuristic techniques designed to handle complex optimisation challenges such as the TSP, which demands identifying the shortest possible path that visits each city exactly once and returns to the starting point. This study aims not only to implement these algorithms but also to analyse their design intricacies, adjust their parameters, and statistically compare their performance based on average results and standard deviations across 30 runs. The goal is to ascertain which technique offers a more effective solution strategy for this version of the TSP, providing valuable insights into the suitability of each algorithm for large-scale optimisation problems.

1 INTRODUCTION

Due to its practical implications and complexity, the TSP is one of the most studied problems in computational mathematics and operations research. This report aims to solve the TSP for the 48 contiguous US state capitals provided in the TSPLIB dataset. To navigate this combinatorial optimisation landscape, we harness the potential of two heuristic algorithms: SA and GA. SA, mirroring the physical process of annealing in metallurgy, offers a way to probabilistically escape local optima, thereby enabling the exploration of the solution space more effectively. Inspired by the mechanisms of natural evolution, GA leverages the genetic power of operators to evolve a population of solutions towards optimality iteratively. Both algorithms are renowned for their adaptability and performance in solving NP-hard problems such as the TSP.

This study does not just apply these algorithms; it also delves into the imperative of meticulous parameter tuning, which is critical for their optimal performance. A rigorous procedure involving 100 trial runs for each algorithm calibrated the parameters, followed by 30 independent runs to gauge their effectiveness. These runs not only provided insight into each algorithm efficiency but also a rich dataset for statistical analysis. The performance of SA and GA was then meticulously compared using the Wilcoxon signed-rank test, providing a statistically significant framework to determine which algorithm better tackles the TSP for the given dataset. This introduction sets the stage for an in-depth exploration of these heuristic giants within one of the most challenging problems in combinatorial optimisation.

2 THEORY

2.1 Travelling Salesman Problem

The TSP is a paradigmatic optimisation challenge that aims to ascertain the most efficient route through a set of cities, visiting each once and only once before returning to the point of origin. This quest to pinpoint the shortest possible journey encapsulates a problem that is deceptively straightforward to state yet exponentially challenging to solve as the number of cities increases. As a problem of considerable historical and practical significance, the TSP serves as a litmus test for the effectiveness of various optimisation methods in fields ranging from logistics to artificial intelligence.

Within this framework, our study zeroes in on a specific instance of the TSP—determining the shortest route that connects the 48 contiguous US state capitals. This particular scenario is not merely an academic exercise; it represents a real-world problem that mirrors the operational challenges faced by numerous logistics companies. The rich dataset provided by the TSPLIB stands as the foundation for our explorations, enabling a detailed evaluation of heuristic optimisation techniques such as SA and GA. This report explores the application of two such strategies—Simulated Annealing and Genetic Algorithm—to the TSP with the aim of highlighting their efficacy and comparative performance.

2.2 Simulated Annealing

SA emerges as a compelling approach for tackling a wide range of optimisation problems, whether free of constraints or subject to specific bounds. At its core, SA is inspired by the metallurgical practice of annealing, which involves heating a material to a high temperature to dissolve its structure and then cooling it slowly to forge a state of reduced defects and optimal energy configuration[2]. By applying this principle metaphorically to optimisation, SA seeks to minimise the 'energy' or cost function of the system it aims to optimise.

In practice, SA starts by allowing a generous threshold for accepting solutions, including those that may not be immediate improvements. This is akin to the high-temperature phase in physical annealing, promoting extensive search and avoiding premature convergence on suboptimal solutions. As the 'temperature' drops, the algorithm becomes increasingly selective, mirroring the material's cooling phase, where atoms settle into a more stable structure. The gradual, controlled cooling schedule is pivotal in guiding the SA algorithm toward a solution close to the global optimum. This introduction discusses how SA operates within the parameters of the Travelling Salesman Problem, leveraging its unique balance of exploration and exploitation to navigate the complex landscape of possible itineraries and deduce an efficient route among the 48 contiguous US state capitals.

Below is the general pseudo-code for SA.

Algorithm 1 Simulated Annealing Algorithm

Let $s = s_0$
for $k = 0$ **to** $k_{\max} - 1$ **do**
  $T \leftarrow$ temperature $\left(1 - \frac{k+1}{k_{\max}}\right)$
  Pick a random neighbour, $s_{\text{new}} \leftarrow \text{neighbour}(s)$
  if $P(E(s), E(s_{\text{new}}), T) \geq \text{random}(0, 1)$ **then**
    $s \leftarrow s_{\text{new}}$
  end if
end for
Output: the final state s

The performance and effectiveness of the SA algorithm are contingent upon a myriad of factors, notably the initial temperature setting, the cooling schedule, and the specific nature of the problem under consideration. This study endeavours to meticulously tune these parameters with the aim of identifying a solution that approximates the optimal with a high degree of accuracy.

2.3 Genetic Algorithm

GA are adaptive heuristic search algorithms premised on the evolutionary ideas of natural selection and genetics. As a robust method for solving both constrained and unconstrained optimisation problems[3], GA operates on a population of potential solutions, applying the principle of survival of the fittest to produce increasingly better approximations of a solution. At each generation, a new set of approximations is created by the process of selecting individuals according to their level of fitness in the problem domain and breeding them together using operators borrowed from natural genetics.

This biological analogy extends through the critical mechanisms of GA, including selection, crossover (recombination of genetic material), and mutation (random alteration of candidate solutions). By manipulating a population of solutions, GAs effectively explore the search space from multiple points concurrently, increasing the likelihood of finding a global optimum. Iterative adjustments of populations transition the pool of solutions towards greater fitness, much like a species evolves traits that are well-adapted to its environment over time.

This report investigates the application of GA to the intricate problem of the TSP, exploring how the algorithm's evolutionary strategies can be leveraged to efficiently navigate the combinatorial landscape presented by the 48 contiguous US state capitals. Through the careful orchestration of selection, crossover, and mutation, GA seeks to uncover a route that represents the shortest possible journey between these nodes.

Below is the general pseudo-code for GA.

Algorithm 2 Genetic Algorithm

Initialize population with random individuals
Evaluate the fitness of each individual in the population
while termination condition not met **do**
  Select parents from the population
  Perform crossover on parents to create offspring
  Perform mutation on offspring
  Evaluate the fitness of the offspring
  Select individuals for the next generation
end while
return The best solution found

GA are powerful tools for solving complex optimisation problems where traditional methods may be inefficient. By simulating the process of natural evolution, GAs can explore a vast solution space and converge towards optimal or near-optimal solutions. The efficiency and effectiveness of a GA depend significantly on the design of its components, including the representation of solutions, the fitness function, and the parameters for selection, crossover, and mutation.

3 METHODOLOGY

3.1 Simulated Annealing

The methodology for applying the SA algorithm to solve the Travelling Salesman Problem (TSP) begins with a strategic foundation, focusing on representing the problem efficiently, iterating through solutions intelligently, and tuning parameters for optimal performance. The challenge of TSP is finding the shortest possible route that visits each city once and returns to the origin, a problem both intriguing and complex due to its NP-hard nature. Here are the implementation and design choices.

Implementation

3.1.1 Parsing TSP Files The parse_tsp_file function is designed to read and extract coordinates from a given TSP file format. This function specifically looks for the "NODE_COORD_SECTION" to start parsing the coordinates, indicating modularity and adaptability in handling standardised TSP files. This approach ensures that the SA implementation can be applied to various TSP instances by simply changing the input file.

3.1.2 Distance Calculation calculate_distance and total_distance functions are implemented to compute Euclidean distances between cities and the total distance of a given path, respectively. Using numpy for these calculations enhances performance due to its optimised backend for numerical operations.

3.1.3 Solution Generation and Manipulation

1. Initial Solution Generation: generate_initial_solution creates a random initial solution. This randomness is vital for the SA algorithm to explore the solution space from different starting points, thus avoiding local minima early on.
2. Neighbour Generation: generate_neighbor randomly swaps two cities in the solution to generate a neighbouring solution, facilitating local exploration.
3. 2-opt Local Search: Reverse implements a simple 2-opt swap mechanism as part of the local search strategy to further explore the neighbourhood of a current solution. This strategy can significantly improve solution quality by untangling routes.

Algorithm 3 Reverse Function

function REVERSE(coords, solution, max_attempts)
  best_distance $\leftarrow$ total_distance(coords, solution)
  for attempt $\leftarrow$ 1 to max_attempts **do**
    start, end $\leftarrow$ sorted(random.sample(range(1, len(solution)), 2))
    new_solution $\leftarrow$ solution[: start] + solution[start : end][:: -1] + solution[end :]
    new_distance $\leftarrow$ total_distance(coords, new_solution)
    if new_distance $<$ best_distance **then**
      solution, best_distance $\leftarrow$ new_solution, new_distance
    end if
  end for
  return solution, best_distance
end function

3.1.4 Simulated Annealing Core The simulated_annealing function encapsulates the essence of SA, including temperature management, iteration control, and acceptance criteria for new solutions based on the Metropolis criterion. Incorporating a 2-opt local search (reverse function) within the SA loop is an excellent strategy for intensifying the search around promising solutions.

3.1.5 Experimental Setup Parameter Tuning: Through conduct_trials, it systematically explore various combinations of SA parameters (initial temperature, cooling rate, stopping temperature, and 2-opt attempts) to find an effective set for the problem at hand. This exploratory step is crucial for adapting SA to specific instances of the TSP.

Performance Evaluation: perform_multiple_runs and related plotting functions (plot_solution and plot_run_statistics) are designed to assess the robustness and effectiveness of the chosen SA parameters over multiple runs, providing insights into the average performance and variability of the solutions.

Main Execution The execution flow demonstrates a comprehensive experimental setup.

Algorithm 4 Main Algorithm

Start
Read TSP file (*parse_tsp_file*)
Conduct trials to find best parameters (*conduct_trials*)
Generate parameter combinations
for each trial **do**
    Choose random parameters
    Run simulated annealing (*simulated_annealing*)
end for
Perform multiple runs with best parameters (*perform_multiple_runs*)
for each run **do**
    Run simulated annealing (*simulated_annealing*)
    Update best overall solution if needed
end for
Display results in plots (*plot_solution, plot_run_statistics*)
End

Design Choices and Considerations

The incorporation of a 2-opt local search within the SA algorithm represents a hybrid approach, enhancing the algorithm's ability to escape local minima and improve solution quality. Systematic parameter tuning based on experimental trials helps in identifying a set of parameters that are likely to perform well for the given TSP instance. Using numpy for numerical operations and matplotlib for visualisation demonstrates a balance between performance efficiency and user-friendly output representation.

Evaluation

3.1.6 Pros Hybrid Approach with 2-opt Local Search: Integrating the 2-opt local search into the SA algorithm significantly enhances its capability to escape local minima and refine solution quality. This strategic blend of global and local search mechanisms bolsters the algorithm's effectiveness in navigating the complex search landscape of the TSP.

Systematic Parameter Tuning: A methodical approach to parameter tuning involving experimental trials to identify an optimal set of parameters exemplifies a robust process that likely contributed to improving the SA algorithm's performance. This demonstrates a deep understanding of the importance of parameter settings in the success of heuristic algorithms.

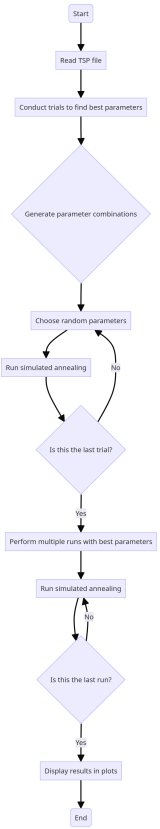

Fig. 1: Flow Chart for SA.

3.1.7 Cons Dependence on Initial Configuration and Parameter Settings: While the report suggests a comprehensive parameter tuning process, SA's performance can still be highly sensitive to its initial configuration and parameter settings. Finding the optimal setup may require extensive experimentation, which can be time-consuming.

Risk of Premature Convergence: Despite efforts to escape local optima, there is still a risk of premature convergence if the cooling schedule is not adequately managed. This could lead to suboptimal solutions if the temperature decreases too rapidly or if the algorithm does not adequately explore the search space.

In summary, the methodology for applying SA to the TSP is designed to leverage the algorithm's strengths while mitigating its weaknesses through strategic problem representation, parameter tuning, and iterative improvement. This balanced approach aims to navigate the complex search space of the TSP to identify a route that minimises the total travel distance efficiently.

3.2 Genetic Algorithm

Exploring the Genetic Algorithm (GA) for the Traveling Salesman Problem (TSP) highlights strategic design choices central to its implementation. This method begins with parsing TSP files for city coordinates—critical for real-world dataset applications—and uses Euclidean distance for effective inter-city calculations. Initial population diversity fosters broad solution exploration, while tournament selection ensures a balance between exploration and exploitation. Crossover and mutation strategies adhere to TSP constraints, fostering notable solution enhancements. The GA's core processes, coupled with rigorous

parameter tuning and computational resource adjustments, underscore its evolutionary dynamic aimed at continuous solution quality improvement. Visualisations and performance metrics offer deep insights into the GA's efficacy, merging theoretical and practical approaches to TSP.

In solving the TSP, the GA shares foundational functionalities with the SA approach, particularly in the parsing of TSP files and the calculation of distances. Both algorithms initiate by extracting city coordinates using a specific format and calculating Euclidean distances between cities. This shared basis allows for an equitable evaluation of their problem-solving capabilities.

3.2.1 Tournament Selection Tournament selection for parent selection strikes a balance between exploration and exploitation. This method effectively maintains genetic diversity while favouring individuals with superior fitness, facilitating a robust evolutionary process.

3.2.2 Genetic Operators The crossover function combines genetic material from two parents to produce offspring, respecting the unique constraints of the TSP. This operation is critical for introducing new genetic variations into the population, driving the evolutionary process forward.

Algorithm 5 Crossover Function

Require: $parent1, parent2$ ▷ Two parent solutions
$size \leftarrow$ length of $parent1$
$child \leftarrow$ array of size $size$, initialised with $None$
$[start, end] \leftarrow$ sorted(random two indices from range $(0, size)$)
$child[start : end] \leftarrow parent1[start : end]$ ▷ Copy a segment from parent1 to child
$p2\_index \leftarrow end$
$c\_index \leftarrow end$
while $None$ in $child$ **do** ▷ Fill in the remaining None values in child
    if $parent2[p2\_index\%size]$ not in $child$ **then**
        $child[c\_index\%size] \leftarrow parent2[p2\_index\%size]$
        $c\_index \leftarrow c\_index + 1$
    end if
    $p2\_index \leftarrow p2\_index + 1$
end while
return $child$

3.2.3 Mutation The mutation strategy implemented is particularly suited for the TSP. By reversing segments of the tour, it explores small, local changes that can lead to significant improvements, embodying the essence of mutation in evolutionary algorithms.

3.2.4 Genetic Algorithm The iterative process of selection, crossover, mutation, and generation renewal—demonstrates a well-conceived evolutionary strategy. This process is designed to gradually enhance the population's fitness, guiding it towards optimal solutions.

3.2.5 Parameter Tuning The inclusion of a parameter tuning phase is a testament to the algorithm's adaptability. By experimenting with various configurations of population size, crossover rate, and mutation rate, the algorithm can be finely tuned to the specificities of the input data, optimising performance.

3.2.6 Execution with Tuned Parameters Adjusting the number of generations based on the population size to keep computational resources in check showcases practical consideration. This approach ensures that the algorithm remains scalable and adaptable to different problem sizes and computational constraints.

Algorithm 6 Mutation Function

Require: $solution, coords$ ▷ Initial solution and coordinates
Require: $maxAttempts \leftarrow 10$ ▷ Maximum mutation attempts
$bestDistance \leftarrow$ TOTALDISTANCE$(coords, solution)$ ▷ Calculate the total distance of the initial solution
for $i \leftarrow 1$**to**$maxAttempts$ **do** ▷ Attempt to mutate the solution up to maxAttempts times
    $indexList \leftarrow$ RANDOMSAMPLE(range$(1, \text{len}(solution)), 2)$
    $[start, end] \leftarrow$ SORT$(indexList)$ ▷ Create a new solution by reversing a segment
    $newSolution \leftarrow solution[: start] +$ REVERSE$(solution[start : end]) + solution[end :]$
    $newDistance \leftarrow$ TOTALDISTANCE$(coords, newSolution)$ ▷ Calculate the total distance of the new solution
    if $newDistance < bestDistance$ **then** ▷ If the new solution is better, update the best solution and distance
        $solution \leftarrow newSolution$
        $bestDistance \leftarrow newDistance$
    end if
end for
return $solution$ ▷ Return the potentially mutated solution

Main Execution The execution flow demonstrates a comprehensive experimental setup.

1. Initialise Parameters: Set up initial parameters such as population size, number of generations, crossover rate, and mutation rate.
2. Load City Coordinates: Parse the TSP file to extract the coordinates of the cities involved in the problem.
3. Generate Initial Population: Create the initial population of solutions randomly. Each solution is a permutation representing a tour through the cities.
4. Evaluate Fitness: For each solution in the population, calculate its fitness. In the TSP, the fitness is usually the inverse of the total distance of the tour, so lower distances are better.
5. Selection: Use tournament selection (or another selection method) to choose individuals from the population to breed. This step selects the fitter solutions with a preference for passing their genes to the next generation.
6. Crossover: Pair selected individuals and perform crossover to produce offspring. Crossover involves combining parts of two parents' genes to create children, introducing new genetic combinations to the population.
7. Mutation: With a certain probability, apply mutation to the offspring. Mutation introduces random changes to some individuals, helping to maintain genetic diversity in the population and preventing premature convergence on suboptimal solutions.
8. Create New Generation: Replace the old population with the newly created offspring, forming the next generation of solutions.
9. Check for Termination: Repeat steps 4 through 8 for a set number of generations or until another termination condition is met, such as achieving a solution below a predefined threshold of distance.
10. Select Best Solution: At the end of the execution, identify the best solution encountered throughout all generations, typically the one with the shortest tour distance.

11. Output Result: Return or display the best solution found, including the order of cities to visit and the total distance.

Algorithm 7 Genetic Algorithm for TSP

```
populationSize, nGenerations ← appropriate values
crossoverRate, mutationRate ← appropriate values    ▷ Load TSP city coordinates
coordinates ← PARSETSPFILE(file path)
                                                    ▷ Generate initial population
population ← CREATEINITIALPOPULATION(populationSize, length of coordinates)
bestDistance ← ∞
bestTour ← None                                     ▷ Main GA loop
for generation ← 1 to nGenerations do
    scores ← CALCULATESCORES(population, coordinates)
    for i ← 1 to length of population do
        if scores[i] < bestDistance then
            bestDistance ← scores[i]
            bestTour ← population[i]
        end if
    end for                                         ▷ Select individuals for mating
    selected ← TOURNAMENTSELECTION(population, scores)
    children ← []
    for i ← 1 to length of selected by 2 do
        parent1, parent2 ← selected[i], selected[i + 1]
        if RANDOM < crossoverRate then
            child1, child2 ← CROSSOVER(parent1, parent2)
        else
            child1, child2 ← parent1, parent2
        end if
        if RANDOM < mutationRate then
            child1 ← MUTATE(child1, coordinates)
        end if
        if RANDOM < mutationRate then
            child2 ← MUTATE(child2, coordinates)
        end if
        Append child1, child2 to children
    end for                                         ▷ Replace old population with new generation
    population ← children
end for                                             ▷ Return the best solution found
return bestTour, bestDistance
```

Evaluation

3.2.7 Pros

1. Flexibility: GAs are inherently flexible and capable of handling various optimisation problems, including the NP-hard TSP. The design of GA, with its customisable operators (selection, crossover, and mutation), allows for easy adjustments to tailor the algorithm to specific problem instances or constraints.
2. Parallelism: The population-based approach of GAs facilitates parallel computation. Multiple solutions are processed simultaneously, speeding up the search for an optimal or near-optimal solution compared to sequential approaches.
3. Adaptability: Through mechanisms like crossover and mutation, GAs can adaptively explore and exploit the search space. This dynamic balance increases the chances of finding global optima by diversifying the search and intensifying it around promising areas.

3.2.8 Cons

1. Convergence Time: One of the main drawbacks of GAs, including the implementation, is the potentially slow convergence to the optimal solution. This is particularly pronounced in complex problem instances of the TSP, where the search space is vast.

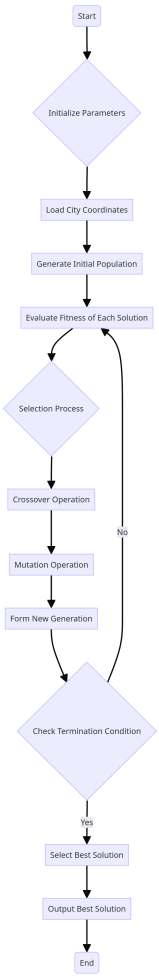

Fig. 2: Flow Chart for GA.

2. Parameter Sensitivity: The performance of GAs significantly depends on the tuning of their parameters (population size, crossover rate, mutation rate, etc.). Finding the right combination of parameters for the GA can be time-consuming and may require extensive empirical testing or domain knowledge.
3. Computational Cost: Especially for large instances of the TSP, the computational cost of running a GA can be high. This is due to the need to evaluate the fitness of many solutions over many generations to adequately explore the search space.

GA design for the TSP is characterised by a thoughtful integration of evolutionary principles with practical considerations tailored to the problem at hand. Through careful selection of genetic operations, parameter tuning, and insightful analysis, the algorithm seeks not only to find solutions to the TSP but also to offer a framework that is adaptable, efficient, and capable of handling the complexities inherent in optimisation problems.

4 ALGORITHM PARAMETERS AND PERFORMANCE

4.1 Simulated Annealing

SA is a probabilistic technique for approximating the global optimum of a given function. In the context of the TSP, the SA performance of algorithm is highly dependent on the tuning of its parameters.

4.1.1 Tunable Parameters The main parameters involved in SA include the initial temperature, cooling rate, stopping temperature, and the number of reverse attempts, each of which influences the algorithm's exploration and exploitation balance:

Initial Temperature A crucial parameter that influences the probability of accepting worse solutions early on to escape local optima. Based on the literature [4;5], the initial temperature was explored from 1000 to 3000, avoiding too low values that increase standard deviation notably.

Cooling Rate This rate, set at 0.99 following [6], dictates how quickly the temperature decreases, balancing between exploration and exploitation. A slower cooling allows for a more thorough search of the solution space.

Stopping Temperature Indicates when the algorithm should cease, with tests on 0.005 and 0.025 to ensure the search does not prolong unnecessarily, based on guidelines from [5].

Reverse Attempts Reflects the local search effort within the broader SA framework, initially set at 100 but varied between 150 to 200 based on trial outcomes to optimise between search depth and computational load.

4.1.2 Tuning Strategy and Impact The tuning involved a series of 30 trial runs, systematically exploring combinations of these parameters to evaluate their effects:

The initial temperature range was selected to maintain a low standard deviation, ensuring consistent quality across solutions. Temperatures below 500 lead to undesirable variability.

A cooling rate of 0.99 was chosen to mirror recommendations from [6], aiming for a balance that favors an exhaustive exploration before convergence.

Adjustments in stopping temperature and reverse attempts were made to fine-tune the stopping criteria and the intensity of local optimisations, striving for an efficient computational time without sacrificing solution quality.

4.1.3 Simulated Annealing Tuning Results The adjustment of parameters based on trial outcomes led to notable findings:

Setting the initial temperature to a higher range effectively reduced solution variability, as evidenced by a standard deviation that became more favourable compared to lower temperature settings.

The selection of a cooling rate close to 1 and optimal stopping temperatures contributed to a systematic and effective exploration of the solution space, manifesting in competitive best distances achieved across different trials.

Modifying reverse attempts demonstrated a delicate balance between enhancing solution quality and incurring additional computational costs. A moderate increase to 150-200 attempts optimised this balance, improving solution consistency without excessively prolonging runtime.

In summary, the SA parameter tuning was meticulously executed, leveraging empirical data from multiple trials to discern the optimal settings that enhance algorithm performance. This data-driven approach not only optimised the computational efficiency but also ensured the reliability of solution quality, underscoring the importance of parameter tuning in achieving a high-performing SA algorithm for the TSP.

Initial Temp Range	Cooling Rate	Stopping Temp	Reverse Attempts	Best Distance	Computation Time (s)
1000–3000	0.99	0.005–0.025	150–200	34291.48	9.91
0–1000	0.99	0.005–0.025	250–300	34820.26	8.29
1000–3000	0.99	0.005–0.025	10–50	41664.53	6.18
1000–3000	0.99	0.005–0.025	50–100	36285.95	7.79
1000–3000	0.99	0.005	100	34907.20	8.18
1000–3000	0.99	0.005	400–500	34291.48	12.61
1000–3000	0.99	0.025	150–200	34924.10	9.91

Table 1. Summary of Simulated Annealing Parameter Tuning for TSP

4.2 Genetic Algorithm

Parameter tuning is pivotal in Genetic Algorithms for the TSP, where optimal settings balance solution quality against computational efficiency, as detailed in our trials with population and mutation rates.

4.2.1 Tunable Parameters **Population Size** Determines the number of individual solutions in each generation. A larger population can offer greater genetic diversity, potentially leading to better solutions but at the cost of higher computational resources and longer computation time. Initially set the population to 100 as suggested by literature [7] [8], but due to long computation times, I halved the population size to 50 for efficiency.

Number of Generations (n_generations): This parameter defines how many cycles of selection, crossover, and mutation will occur. More generations can lead to a more refined solution but increase the runtime. In the trials, generation has been fixed at 100 [9] [10].

Crossover Rate: A probability that determines how often crossover will be performed. A higher crossover rate increases the diversity of the population by combining the genetic information of parents more frequently. Experimented with various rates: 0.6, 0.7, 0.8, and 0.9 [10].

Mutation Rate: The probability of a mutation occurring in an offspring. Mutation introduces new genetic structures into the population, which is vital for maintaining genetic diversity and for exploring new areas of the search space. Experimented with rates of 0.01, 0.05, and 0.1, in contrast to the 0.3 and 1% rates suggested by literature [10] and [7].

4.2.2 Tuning Strategy and Impact To tune these parameters, I employed a strategy of trial runs, where it would select different combinations of crossover and mutation rates to observe their impact on the GA's performance.

By reducing the population size from 100 to 50, computation time decreased, and the standard deviation of the distances reduced from around 1000 to 700, indicating more consistent performance.

For crossover and mutation rates, the trials aimed to balance exploration and exploitation—finding new solutions while refining existing good ones. Higher crossover rates were tested, probably leading to a broader search of the solution space, while different mutation rates provided a check against convergence to local optima.

The number of generations was kept constant at 100, which suggests that there is a satisfactory number of iterations for the GA to converge to good solutions within a reasonable computation time.

Through the 100 trial runs with these parameter settings, the best overall distance, average distance, and standard deviation for the solutions were found, indicating how well the GA performed.

4.2.3 Genetic Algorithm Tuning Results The GA was tuned over the following ranges of parameters:

- Crossover rates: 0.6, 0.7, 0.8, 0.9
- Mutation rates: 0.01, 0.05, 0.1
- Number of generations: 100

The following table summarises the best distances achieved for population sizes of 50 and 100:

Population Size	Best Distance	Spent Time (s)
50	34327.82	75.2
100	34162.69	168.89

Table 2. Tuning results for Genetic Algorithm on TSP

The best overall distance achieved with a population of 50 was 34327.81, which is competitive with the best distance found using a population of 100, 34162.69. This suggests that halving the population did not significantly deteriorate the best performance of the GA.

The standard deviation with a population of 50 was lower than with a population of 100, implying that the solutions were more consistent, though this must be balanced against the best and average distances achieved.

The decrease in computation time was significant when using a smaller population, which could make the GA more practical for certain applications where time efficiency is crucial.

In conclusion, the parameter tuning strategy was data-driven, leveraging multiple trials to empirically determine an effective balance between genetic diversity and search efficiency. The changes made to the population size notably decreased computational demands while maintaining performance, which is a desirable outcome in many practical scenarios. However, it is also important to consider the trade-off between search thoroughness and computational resources, especially when scaling to larger problem instances or requiring more refined solutions.

5 STATISTICAL COMPARISON & DISCUSSION

Considering the average distances and the standard deviations reported:

Table 3. Performance Comparison of SA and GA

	Simulated Annealing	Genetic Algorithm
Average Distance	37,852.11	35,422.67
Standard Deviation	2,052.31	806.87
Runtime (seconds)	13.79	85.86

The GA algorithm has a better (lower) average distance than the SA algorithm, and its solutions are more consistent, as indicated by a smaller standard deviation. This suggests that GA not only finds better solutions on average but also does so more reliably from run to run.

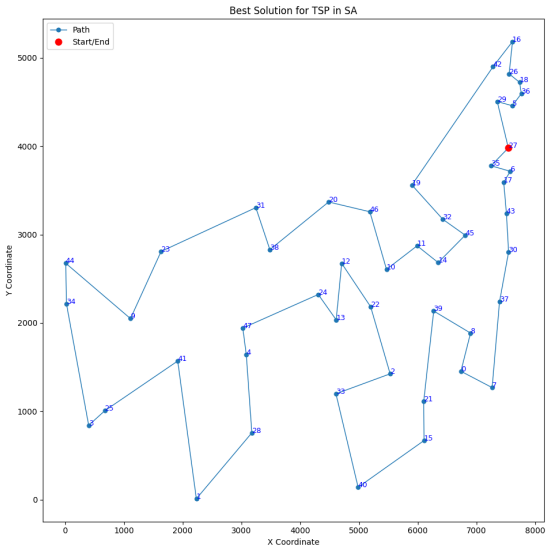

Fig. 3: SA Best Solution

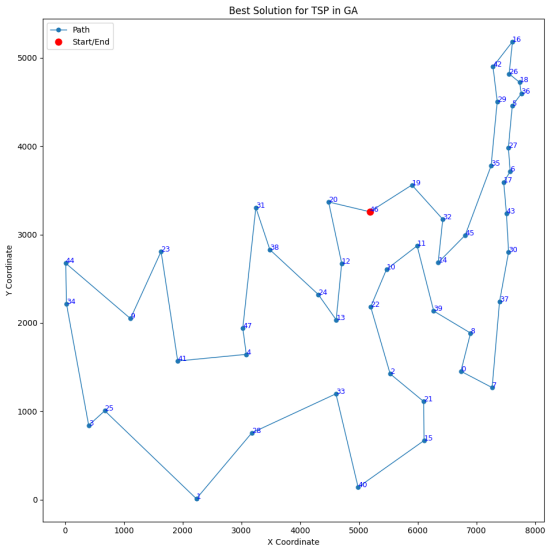

Fig. 4: GA Best Solution

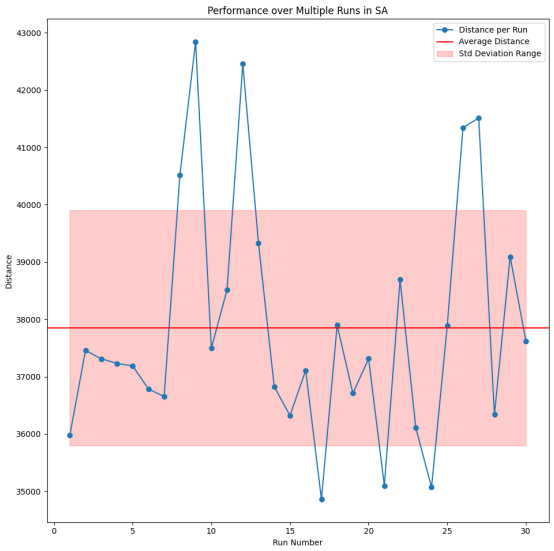

Fig. 5: Average Result and Standard Deviations in SA

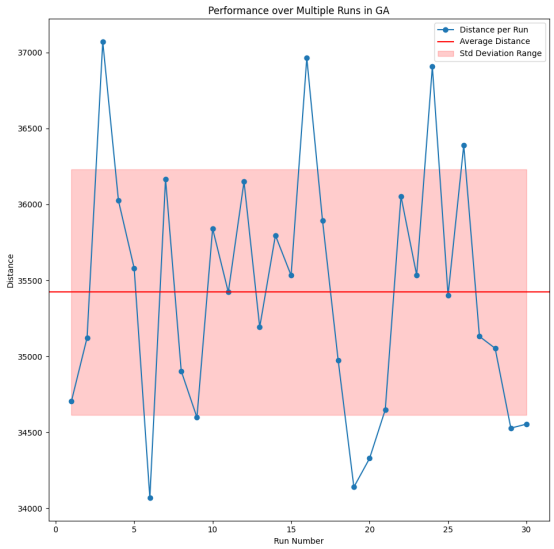

Fig. 6: Average Result and Standard Deviations in GA

However, there's a trade-off with runtime. SA is much faster, completing its runs in just 13.79 seconds, whereas GA takes a longer time at 85.86 seconds. This suggests that while GA may provide higher solution quality, it does so at the expense of computational time. This trade-off between solution quality and runtime is a key consideration in algorithm selection for practical applications: if faster runtime is critical, SA might be the preferred choice, even if it sacrifices some solution quality. Conversely, if the quality of the solution is paramount and runtime is less of a concern, the GA would be more suitable despite its longer computational time.

5.1 Statistical Test Results

The Wilcoxon signed-rank test is a non-parametric statistical test used to compare two related samples, matched pairs, or repeated measurements on a single sample to assess whether their population mean ranks differ. It is used when the data doesn't meet the assumptions of the t-test, especially the assumption of the normal distribution of differences. In the context of algorithm performance comparison, it can be used to determine if there is a statistically significant difference between the performance outcomes, such as solution quality or distances in TSP, of two different algorithms over multiple runs.

5.1.1 Test Statistic (22.0) This value serves as a key indicator of the disparity in performance between the SA and GA regarding the TSP. Derived from the sum of ranks for the differences between paired observations, it suggests a substantial difference in the efficacy of the two algorithms. A statistic this low is indicative of significant performance variation, highlighting the divergent outcomes achieved by SA and GA in solving the TSP.

5.1.2 P-value (approx. 9.98e-07) The p-value plays a crucial role in interpreting the statistical significance of the observed differences

Table 4. Comparison of SA and GA performances using the Wilcoxon Signed-Rank Test

Run	SA Distance	GA Distance	Difference	Abs. Difference	Rank	Negative Rank	Positive Rank
16	37111.094627	36963.641526	147.453102	147.453102	1.0	0.0	1.0
3	37311.838624	37072.015051	239.823573	239.823573	2.0	0.0	2.0
21	35098.681227	34649.109009	449.572218	449.572218	3.0	0.0	3.0
7	36653.712501	36167.583504	486.128996	486.128996	4.0	0.0	4.0
23	36108.842626	35534.888210	573.954416	573.954416	5.0	0.0	5.0
15	36322.039113	35532.818942	789.220171	789.220171	6.0	0.0	6.0
14	36826.081044	35796.235684	1029.845361	1029.845361	7.0	0.0	7.0
17	34861.388446	35895.236720	-1033.848275	1033.848275	8.0	-8.0	0.0
4	37230.959857	36024.968285	1205.991572	1205.991572	9.0	0.0	9.0
1	35975.614245	34703.241858	1272.372388	1272.372388	10.0	0.0	10.0
28	36338.786826	35053.012697	1285.774129	1285.774129	11.0	0.0	11.0
5	37187.928149	35580.716390	1607.211759	1607.211759	12.0	0.0	12.0
10	37494.263556	35840.721769	1653.541787	1653.541787	13.0	0.0	13.0
24	35070.194765	36907.988060	-1837.793294	1837.793294	14.0	-14.0	0.0
2	37458.369116	35122.450871	2335.918245	2335.918245	15.0	0.0	15.0
25	37887.577481	35401.848445	2485.729035	2485.729035	16.0	0.0	16.0
19	36709.210920	34139.759691	2569.451228	2569.451228	17.0	0.0	17.0
22	38691.351530	36054.078434	2637.273095	2637.273095	18.0	0.0	18.0
6	36784.782085	34068.840175	2715.941910	2715.941910	19.0	0.0	19.0
18	37898.769865	34973.601071	2925.168794	2925.168794	20.0	0.0	20.0
20	37312.839870	34328.597506	2984.242364	2984.242364	21.0	0.0	21.0
30	37620.855769	34554.235507	3066.620261	3066.620261	22.0	0.0	22.0
11	38516.103341	35422.432428	3093.670913	3093.670913	23.0	0.0	23.0
13	39325.573322	35191.641350	4133.931972	4133.931972	24.0	0.0	24.0
29	39092.312228	34527.797139	4564.515089	4564.515089	25.0	0.0	25.0
26	41345.834571	36389.198180	4956.636391	4956.636391	26.0	0.0	26.0
8	40515.936441	34903.517484	5612.418957	5612.418957	27.0	0.0	27.0
12	42455.148503	36150.222929	6304.925574	6304.925574	28.0	0.0	28.0
27	41512.484186	35131.403709	6381.080476	6381.080476	29.0	0.0	29.0
9	42844.819592	34598.294996	8246.524596	8246.524596	30.0	0.0	30.0
				Sum of Negative Ranks:	-22		
				Sum of Positive Ranks:	406		
				Total Sum of Ranks:	384		

between SA and GA. It represents the probability of encountering results at least as extreme as those observed under the presumption that the null hypothesis, which is no difference in median distances produced by the algorithms, holds true. Given that this p-value is drastically below conventional significance levels, e.g., 0.05 or 0.01, it compellingly suggests rejecting the null hypothesis, affirming a statistically significant difference in performance between the two algorithms.

This analysis indicates that the solution quality yielded by SA and GA for the TSP is not just different, but significantly so. It suggests that the inherent mechanisms and optimisation strategies employed by these algorithms lead to markedly distinct outcomes when addressing the same problem set. This differentiation in performance, validated by statistical analysis, affords valuable insights into the algorithms' efficiency and effectiveness in navigating the complex landscape of the TSP.

In summary, the Wilcoxon test's outcome and the associated data allow us to conclude that GA consistently yields better solutions than SA for the TSP, albeit at a higher computational cost in terms of runtime.

6 CONCLUSION

This research rigorously compares Simulated Annealing (SA) and Genetic Algorithm (GA) in solving the TSP ATT48, aiming to identify the superior optimisation technique through statistical analysis, focusing on parameter tuning and performance. The study systematically assessed both algorithms across multiple trial runs, revealing SA's capacity for quick, albeit variable, solutions, and GA's consistency in delivering quality results. Statistical testing confirmed a significant performance difference favoring GA, highlighted by its lower average distances and deviations. The conclusion acknowledges GA's efficiency and consistency over SA, suggesting further exploration into hybrid approaches for optimising solution quality and speed.

REFERENCES

[1]University of Heidelberg. (Year the page was accessed, 2024) TSPLIB - a library of TSP instances. [Online]. Available: http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/

[2]MathWorks. (Year the page was accessed, 2024) What is simulated annealing? [Online]. Available: https://www.mathworks.com/help/gads/what-is-simulated-annealing.html

[3]——. (Year the page was accessed, 2024) What is a genetic algorithm? [Online]. Available: https://www.mathworks.com/help/gads/what-is-the-genetic-algorithm.html

[4]A. E.-S. Ezugwu, A. O. Adewumi, and M. E. Frîncu, "Simulated annealing based symbiotic organisms search optimization algorithm for traveling salesman problem," *Expert Systems with Applications*,

vol. 77, pp. 189–210, 2017.

[5]X. Geng, Z. Chen, W. Yang, D. Shi, and K. Zhao, "Solving the traveling salesman problem based on an adaptive simulated annealing algorithm with greedy search," *Applied Soft Computing*, vol. 11, no. 4, pp. 3680–3689, 2011. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1568494611000573

[6]S.-h. Zhan, J. Lin, Z.-j. Zhang, Y.-w. Zhong *et al.*, "List-based simulated annealing algorithm for traveling salesman problem," *Computational intelligence and neuroscience*, vol. 2016, 2016.

[7]S. Chatterjee, C. Carrera, and L. A. Lynch, "Genetic algorithms and traveling salesman problems," *European Journal of Operational Research*, vol. 93, no. 3, pp. 490–510, 1996. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0377221795000771

[8]H.-K. Tsai, J.-M. Yang, Y.-F. Tsai, and C.-Y. Kao, "An evolutionary algorithm for large traveling salesman problems," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 34, no. 4, pp. 1718–1729, 2004.

[9]R. S. Beed, S. Sarkar, A. Roy, and S. Chatterjee, "A study of the genetic algorithm parameters for solving multi-objective travelling salesman problem," in *2017 International Conference on Information Technology (ICIT)*, 2017, pp. 23–29.

[10]S.-M. Chen and C.-Y. Chien, "Solving the traveling salesman problem based on the genetic simulated annealing ant colony system with particle swarm optimization techniques," *Expert Systems with Applications*, vol. 38, no. 12, pp. 14 439–14 450, 2011. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417411006907