SIMULATED ANNEALING ALGORITHM WITH METROPOLIS CRITERION

Abstract

In most of the scenarios, the functions that have to find the optimum solution are complex functions. Thus, there is a need for optimization methods that can explore search spaces effectively and efficiently suitable for multidimensional and multimodal functions. The simulated annealing (SA) algorithm is a continuous optimization method that is designed to find the global optimum solution while avoiding being trapped in local optimum solutions. This paper presents a version of simulated annealing that has been developed using the metropolis algorithm. The performance of the algorithm was analyzed and compared by applying two different cooling schemes, constant cooling factor (CCF) and variable cooling factor (VCF). The SA algorithm was able to find the global optimal solution of the unimodal test functions and multimodal test functions. The SA algorithm with CCF gives the more accurate solution while the convergence to the optimum solution was faster with the VCF.

Keywords: Simulated annealing, Metropolis algorithm, Cooling schedule, Global optimization

Introduction

The simulated annealing algorithm is a powerful optimization technique that is designed to find the global optimum of a given function. This algorithm was inspired by the metallurgical process of annealing, a material is heated and then slowly cooled to achieve its stable crystalline structure. Simulated annealing employs a probabilistic approach to explore the solution space. While traditional techniques often get stuck in local optima, this method can avoid that and is especially useful for complex problems.

The core of the simulated annealing algorithm is the Metropolis algorithm. The Metropolis algorithm is a type of Monte Carlo method that generates a sequence of solutions by gradually reducing the "temperature" of the system. At each step, it accepts new solutions based on probability. This probability depends on the difference in their objective function values and the current temperature. Sometimes, it accepts worse solutions. This helps the algorithm to escape local minima and explore a wider solution space.

The effectiveness of simulated annealing depends on its cooling schedule and cooling factor. These determine how the temperature is lowered over time. A well-designed cooling schedule ensures that the algorithm has enough time to explore the solution space at higher temperatures. It gradually refines the search as the temperature decreases. Even Though there is much research available that has been done by applying linear, exponential, geometric, and various types of cooling schedules, there is no standard method that is known to be better than the other schedules.

This report explores the details of the simulated annealing algorithm and covers the theoretical foundations and practical implementations. By examining benchmark functions and applying different cooling schedules, we aim to provide a comprehensive understanding of how simulated annealing can be effectively utilized in various optimization scenarios.

Materials and Methods

The primary focus of this research is to explore the effective utilization of the simulated annealing (SA) algorithm in finding the global optimum of complex optimization problems. The study also examines the impact of different cooling schedules and cooling factors on the algorithm's performance.

The research framework for this study involves a comprehensive examination of the simulated annealing algorithm, especially focused on the cooling schedule and cooling factor.

This framework includes:

1. **Theoretical Foundations**: Understanding the principles behind the simulated annealing algorithm and the Metropolis algorithm.
2. **Implementation**: Developing and implementing the algorithm in Python.
3. **Benchmarking**: Applying the algorithm to various benchmark functions to assess performance.
4. **Comparison**: Comparing the results obtained using different cooling schedules and cooling factors.

Method

Theoretical Foundation

Simulated annealing is based on the physical process of annealing in metallurgy. This process involves heating a material and then slowly cooling it. The goal is to remove defects and reach a stable crystalline state. In optimization, simulated annealing uses a probabilistic technique to explore the solution space of a given function.

The algorithm uses the Metropolis algorithm, a Monte Carlo method. At each iteration, a new solution is generated. This solution is accepted based on a probability that depends on the difference in objective function values between the new and current solutions and the current temperature. This probability is given by the **Boltzmann factor**:

$$P = exp(\frac{-\Delta E_{ij}}{T})$$

where ΔE_{ij} is the change in the objective function value, and T is the current temperature.

Implementation

The SA algorithm was implemented using Python, leveraging libraries such as NumPy for numerical computations. Key components of the implementation include:

1. **Initialization**: Setting the initial temperature, cooling schedule, and initial solution.
2. **Iteration**: At each step, a new solution is generated by perturbing the current solution. The new solution is accepted with a probability determined by the Boltzmann factor.
3. **Cooling**: The temperature is gradually reduced according to a predefined cooling schedule

Cooling Schedules

Two primary cooling schedules were investigated:

1. **Cooling factor (Constant cooling factor)**

   $$T_{k+1} = T_k \cdot 0.85$$

   where T_k is the temperature at the current iteration and T_{k+1} is the temperature at the next iteration

2. **Geometric cooling schedule (Variable cooling factor)**

$T_{k+1} = \alpha^{k} \cdot T_{k}$, where $\alpha = \frac{T_{min}^{\frac{1}{n-1}}}{T_{max}}$

T_{max} = Maximum temperature (Start Temperature)

T_{min} = Minimum temperature

n = Number of transition States, $n > 1$

k = current transition state ($k = 2, 3, \ldots\ldots, n - 1$)

Benchmarking

The performance of the simulated annealing algorithm was evaluated using various benchmark functions, including unimodal and multimodal functions.

1. **Unimodal Function (Ackley Function)** :
   - **Objective Function**: $f(x, y) = -200e^{-0.02\sqrt{x^2+y^2}}$
   - **Global Minimum**: x = 0, y = 0, f(0,0) = −200

2. **Multimodal Function (Camel Function)**:
   - **Objective Function**: $f(x, y) = 2x^2 - 1.05x^4 + \frac{x^6}{6} + xy + y^2$
   - **Global Minimum**: x = 0, y = 0, f(0,0) = 0

Each function was tested with different cooling schedules and parameters to assess the algorithm's ability to find the global optimum.

Comparison and Analysis

The results from different cooling schedules were compared. We looked at how accurate the solutions were and how quickly they converged. We used error metrics and visualizations to show the differences and determine which cooling strategy was most effective.

Justification of Methods

Simulated annealing was chosen for this study because it handles complex optimization problems well. The algorithm's probabilistic nature helps it escape local minima and explore a larger solution space. This increases the chance of finding the global optimum. We used standard and well-documented cooling schedules. These provide a solid basis for comparison and analysis. The benchmark functions cover a range of optimization challenges, making the findings broadly applicable.

This study examines different cooling schedules and their impact on the algorithm's performance. The goal is to provide valuable insights into using simulated annealing in various optimization scenarios.

Results and Discussion

Initially, we tested the effectiveness of our simulated annealing algorithm on the simple quadratic function $f(x) = x^2$ to validate its basic functionality and performance.

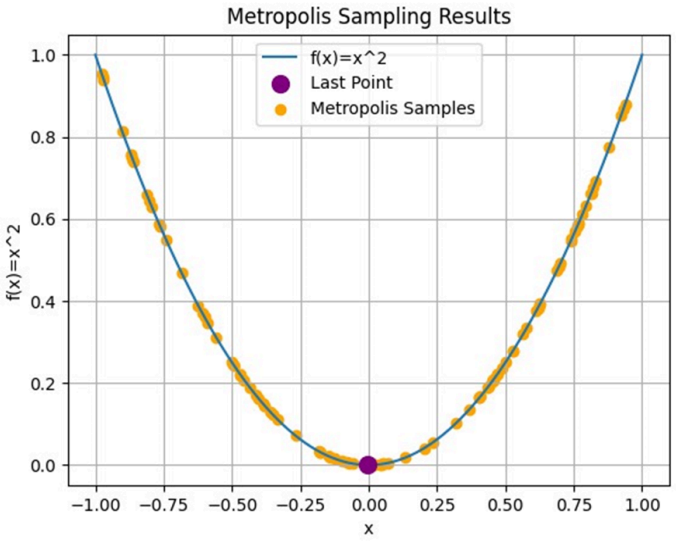

Figure 1 graph of x^2

The global minimum for the quadratic function $f(x) = x^2$ occurs at x = 0, where f(0) = 0.

Next, we plotted the value of the accepted states in each iteration at different temperatures to analyze the convergence behavior of our simulated annealing algorithm.

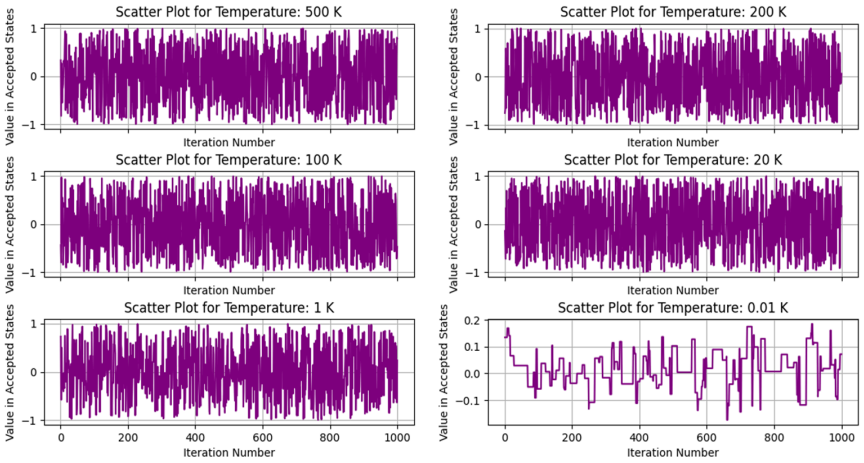

Figure 2 Scatter plots for different temperatures

According to Figure 2, when the temperature decreases, the fluctuations in the values of accepted states become less noticeable.

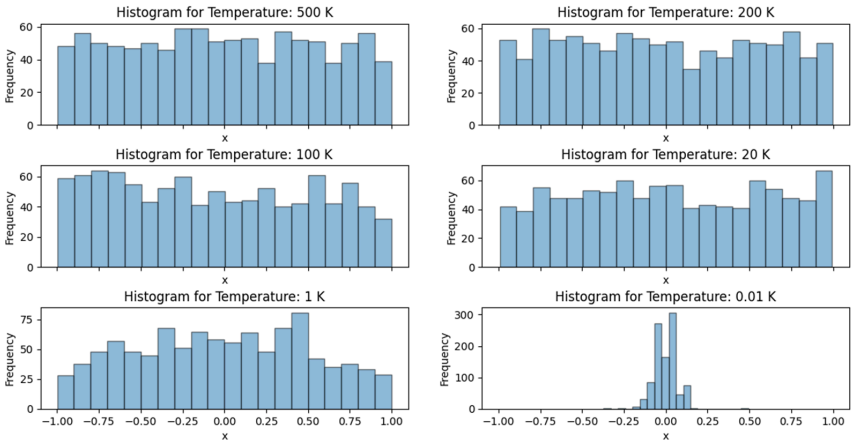

Figure 3 Histograms for different temperatures

The histograms in Figure 3 illustrate the distribution of accepted states for different temperatures. At high temperatures, the accepted states are uniformly distributed across the range indicating extensive exploration. As the temperature decreases, the accepted states increasingly concentrate around the optimal value of x = 0, reflecting the algorithm's focus on exploiting the most promising solutions.

Cooling Factor (Constant cooling factor)

We cannot decrease the temperature manually, so we use a cooling factor to gradually lower the temperature over time. The cooling factor regulates the rate of temperature decrease, allowing the algorithm to transition smoothly from extensive exploration of the solution space to focus on the most promising solutions. In our study, we use a cooling factor of 0.85. By applying the formula $T_{k+1} = T_k \cdot 0.85$, we calculate the temperature for the next iteration.

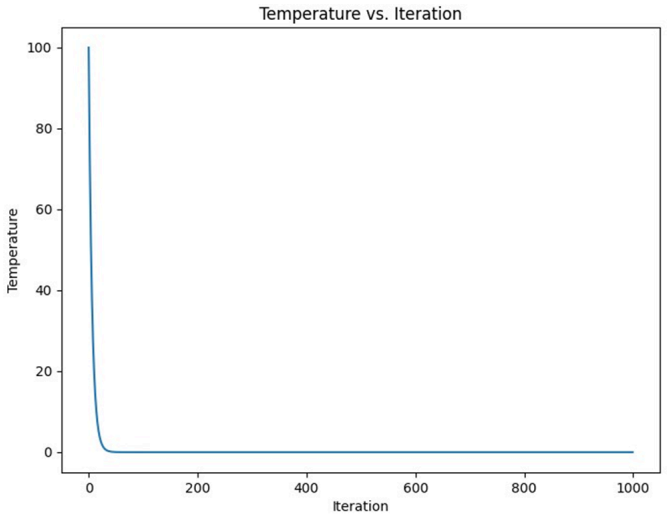

Figure 4 Temperature vs. Iterations graph

Figure 4 illustrates the relationship between temperature and iterations. There is a sudden drop in temperature before even the 50th iteration. This sudden drop indicates a rapid cooling process.

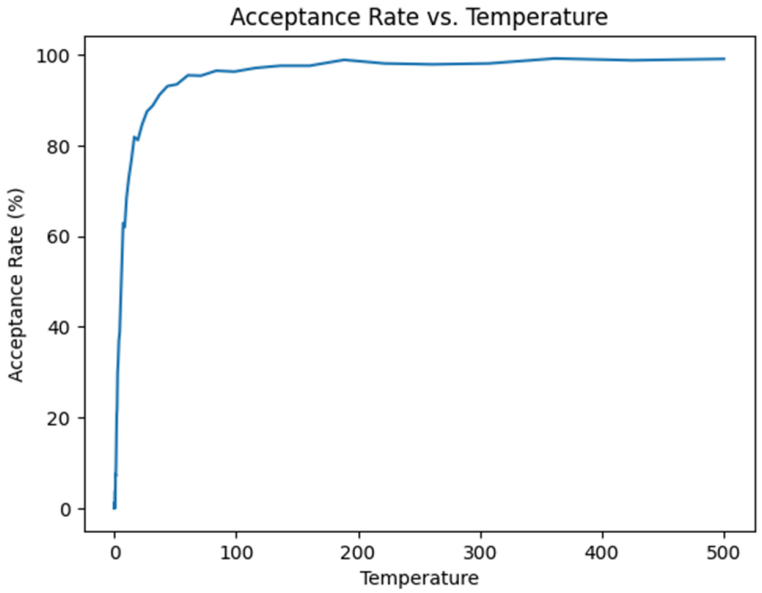

Figure 5 Acceptance Rate vs. Temperature graph

Figure 5 shows the acceptance rate versus iterations. As the temperature decreases, the acceptance rate increases. This trend indicates that, at higher temperatures, the algorithm accepts a wide range of states, including many suboptimal ones, facilitating extensive exploration. As the temperature lowers, the algorithm becomes more selective, accepting fewer worse solutions and focusing more on refining the best solutions, thereby increasing the acceptance rate of optimal states.

Next, we modify our objective function to a nonlinear two-variable function, $f(x, y) = (x - 5)^2 + y^2 - 3y + 2$, to further evaluate our algorithm's performance. The global minimum for this function is found at x = 5 and y = 1.5, where f(5,1.5) = −0.25. We set the parameter ranges to x∈(0,6) and y∈(0,6), and conduct the optimization over 1000 iterations.

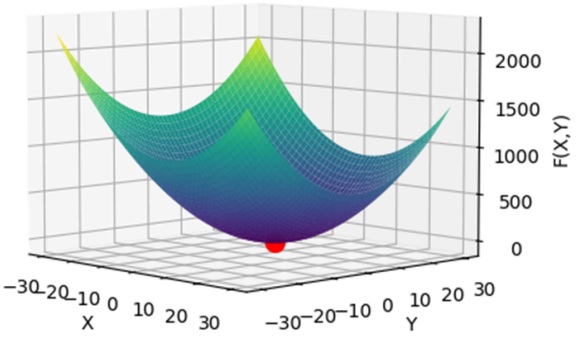

Figure 6 Surface plot of the objective function

First, we plot the function values at each iteration to observe the changes in the objective function throughout the optimization process at an initial temperature of 500K with a cooling factor of 0.85. Next, we plot the accepted states in each iteration under the same conditions. These plots help us visualize how the algorithm explores and exploits the solution space, providing insights into its performance and convergence behavior.

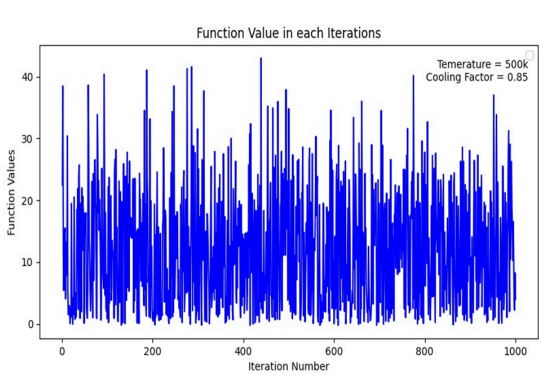

Figure 7 Function Value vs. Iteration Number graph

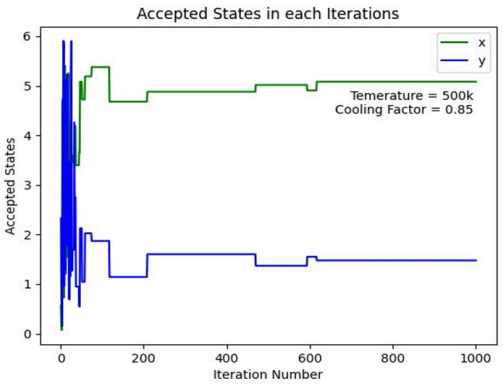

Figure 8 Acceptance Rate vs. Iteration Number graph

```
Final Optimum Values:
x = 5.081584862745347
y = 1.4768728449365873
Optimum minimum value of f(x,y): -0.2428090448694955
```

Figure 9 Final Output Values

As shown in Figure 8, the states of x and y reach their optimal values after approximately 600 iterations. In Figure 9, the output values of our algorithm can be seen. These values closely approximate the global optimum values of our objective function, demonstrating the effectiveness of the simulated annealing process.

Geometric Cooling Schedule (Variable cooling factor)

Next, we test this function with the geometric cooling schedule. We set the parameter ranges to $x \in (0,6)$ and $y \in (0,6)$, and conduct the optimization over 350 transition States and 2000 Markov Chain Iterations.

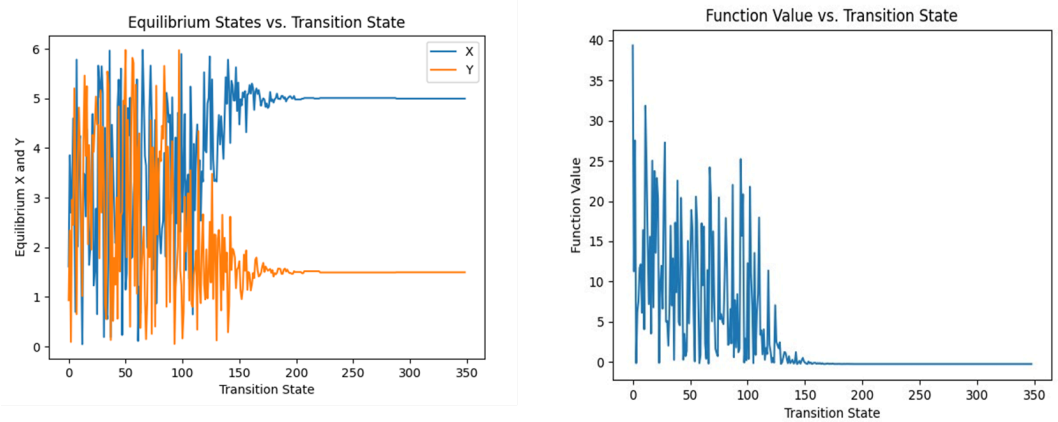

Figure 10 x value, y value, Function value variation with Transition States

```
Final X: 4.996718898741357
Final Y: 1.4922105117642646
Final Function Value: -0.2499285582475559
```

Figure 11 Final Optimal Solution

As shown in Figure 10, the states of x and y reach their optimal values after approximately 200 transition states. In Figure 11, the output values of our algorithm can be seen. These values closely approximate the global optimum values of our objective function, demonstrating the effectiveness of the simulated annealing process.

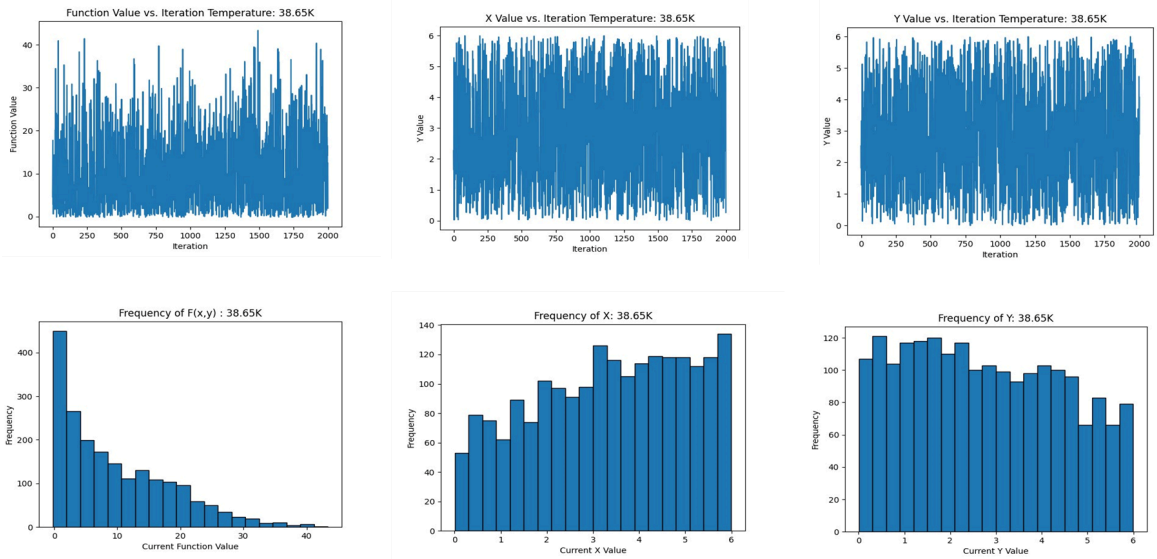

Figure 12 Distribution of x,y values and function Values Across 2000 Iterations at 90th transition state

Figure 12 illustrates the changes in x values, y values, and function values over 2000 iterations at the 90th transition state. More function values are near zero. Additionally, there are more x values in the 4-6 range and more y values in the 0-2 range. This indicates that the algorithm tends to generate values closer to the optimal solution as the temperature decreases. Therefore, we can conclude that the geometric cooling schedule approximates a Maxwell-Boltzmann distribution.

Benchmark Functions

Ackley 2 Function

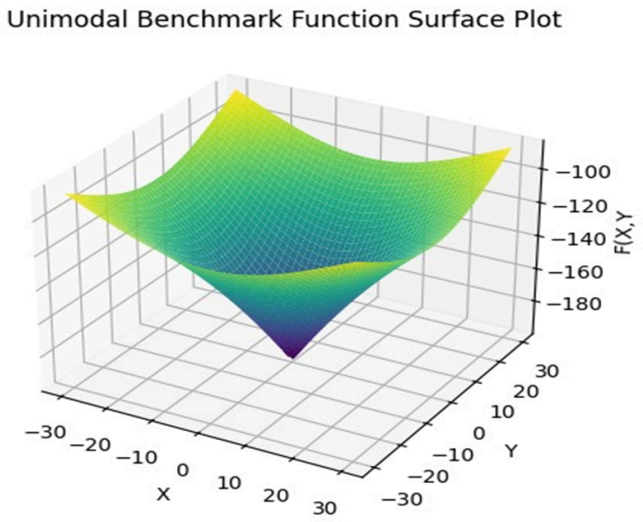

Figure 13 Unimodal Benchmark Function Surface plot

Figure 13 represents the Ackley 2 function over its defined range of $-32 \leq x, y \leq 32$. The function is the standard benchmark for testing optimization algorithms due to its large number of local minima and one global minimum at x = 0, y = 0, and f(x, y) = -200.
This complex landscape shown in the graph tests the ability of the simulated annealing algorithm to avoid getting trapped in local minima and instead find the global minimum.

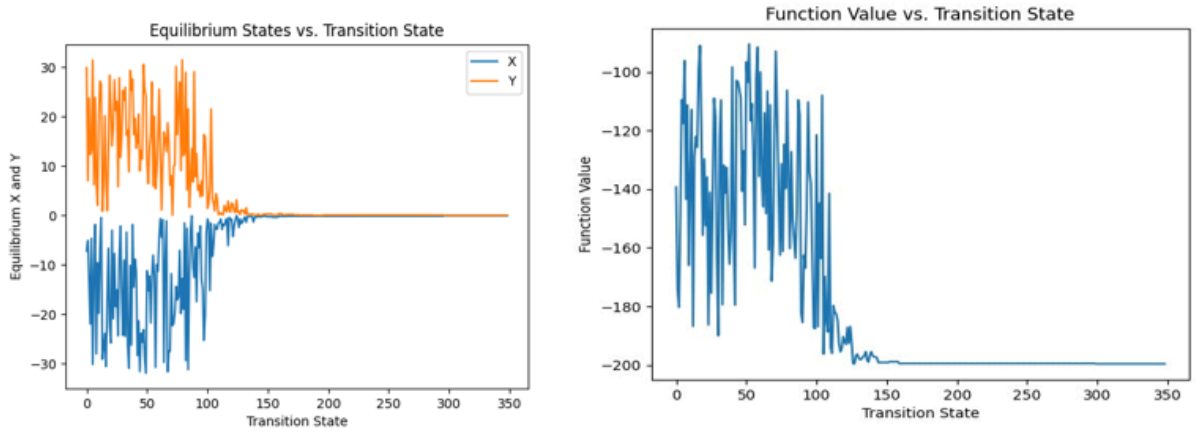

Figure 14 x value, y value, Function value variation with Transition States

The left side of Figure 14 shows the equilibrium state x and y range for the Ackley function optimization using the geometric cooling schedule that was last discussed. It demonstrates how the search range changes over iterations, indicating the broad exploration initially, which eventually focuses on a smaller range near the global minimum.

Right side of Figure 14 represents the function value range for the Ackley function under the same cooling schedule. Similar to the left graph, the right graph shows the search space gradually narrowing down, focusing more closely on the optimal solution as the temperature decreases systematically.
However, it illustrates the effectiveness of the cooling schedule in optimizing the Ackley function. This methodical approach ensures that the global minimum is identified with high precision.

Camel Function

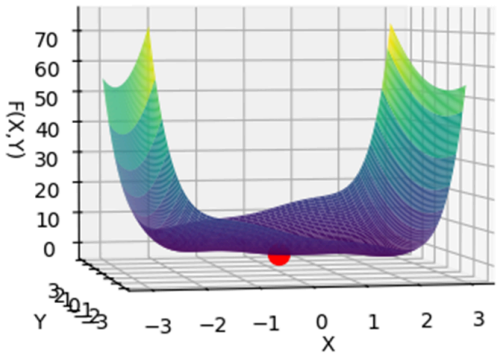

Figure 15 Unimodal Benchmark Function Surface plot

Figure 15 depicts the Camel function, also known as the "Six-Hump Camel" function, within the range of $-5 \leq x,y \leq 5$. This is another common benchmark function with multiple local minima, making it challenging for optimization algorithms to find the global minimum. The global minimum is at x = 0, y = 0, and f(x,y) = 0.

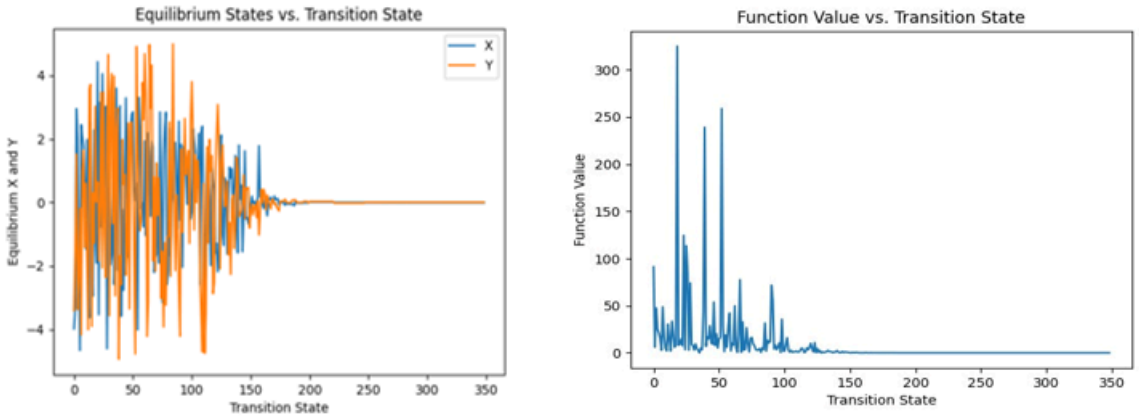

Figure 16 x value, y value, Function value variation with Transition States

Figure 16 demonstrates the dynamics of the simulated annealing algorithm in optimizing the multimodal Camel function. The left graph shows the gradual stabilization of the equilibrium states, while the right graph illustrates the corresponding decrease and stabilization in the function values. These results highlight the efficiency of the cooling schedule in guiding the algorithm through the complex landscape of the Camel function, ultimately achieving convergence to an optimal solution.

Comparison of Cooling factor vs Cooling schedule for unimodal

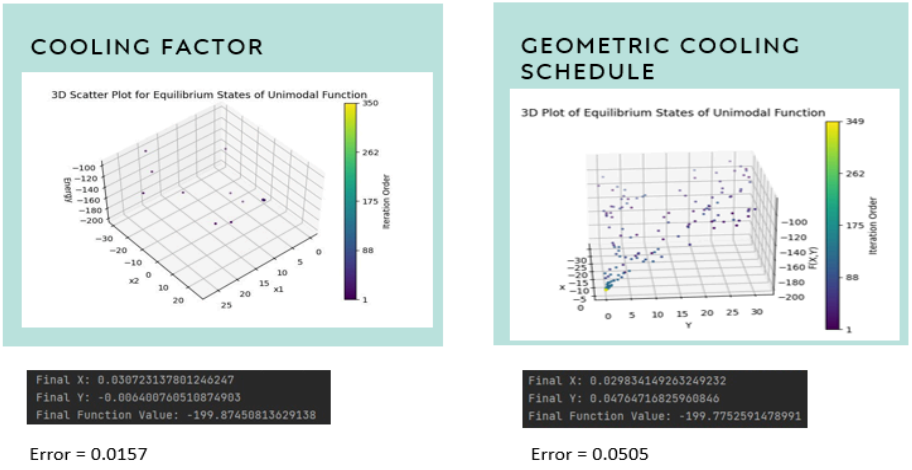

Figure 17 3D Scatter plots of Equilibrium States, Final outputs and MSE

Figure 17 represents a comparative analysis of two cooling strategies Cooling Factor and Geometric Cooling Schedule applied to an unimodal function. The performance of these strategies is evaluated through 3D scatter plots depicting the equilibrium states during the optimization process.

The comparative analysis of the Cooling Factor and Geometric Cooling Schedule strategies reveals that the Cooling Factor method is more effective for unimodal function optimization. This is evidenced by the lower error value and the denser concentration of equilibrium states near the optimal value. The Geometric Cooling Schedule, while still effective, exhibits greater variability and a slightly higher error, indicating a less precise convergence to the global minimum. These findings highlight the importance of choosing an appropriate cooling strategy to enhance the performance of the simulated annealing algorithm in optimization tasks.

Comparison of Cooling factor vs Cooling schedule for multimodal

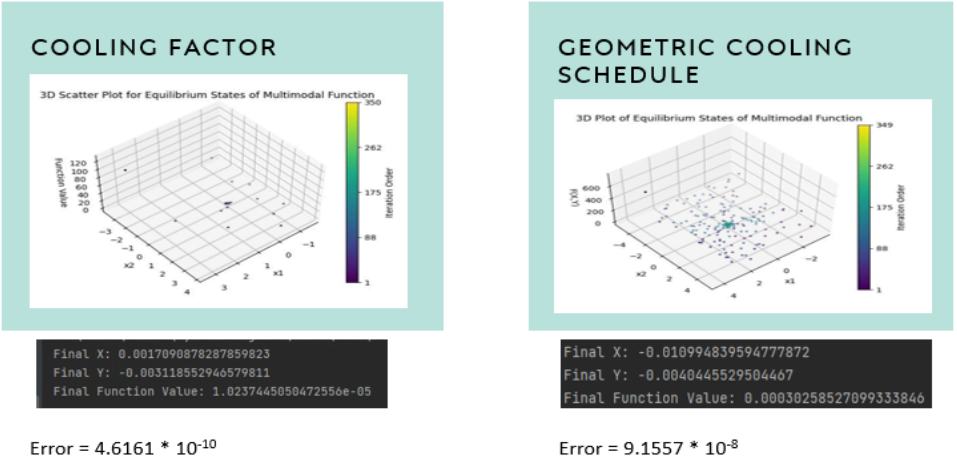

Figure 18 3D Scatter plots of Equilibrium States, Final outputs and MSE

Figure 18 shows a comparative analysis of two cooling strategies, the Cooling Factor and the Geometric Cooling Schedule, applied to a multimodal function. The performance of these strategies is evaluated through 3D scatter plots depicting the equilibrium states during the optimization process.

The comparative analysis of the Cooling Factor and Geometric Cooling Schedule strategies reveals that the Cooling Factor method is more effective for multimodal function optimization. This is evidenced by the lower error value (4.6161×10^{-10})and the denser concentration of equilibrium states near the optimal value. The Geometric Cooling Schedule, while still effective, exhibits greater variability and a slightly higher error (9.1557×10^{-8}), indicating a less precise convergence to the global minimum. These findings highlight the importance of choosing an appropriate cooling strategy to enhance the performance of the simulated annealing algorithm in optimization tasks.

Conclusion

Although the Cooling Factor method generally performs better and gives more accurate results than the Geometric Cooling Schedule, the final analysis shows that the Geometric Cooling Schedule can sometimes find solutions closer to the best possible answer for certain complex problems. This suggests that while the Cooling Factor method often provides better overall accuracy, the Geometric Cooling Schedule might be more effective for specific types of functions.

These findings highlight that the choice of cooling strategy can depend on the problem being solved. So, while the Cooling Factor is usually the better option, the Geometric Cooling Schedule could be more useful for certain situations. This insight helps in choosing the best cooling strategy for improving the performance of simulated annealing algorithms based on the specific problem at hand.

In our current algorithm, hyperparameters such as the initial temperature, the number of transition states, the number of Markov chains, and the x, and y domains need to be manually adjusted each time the function changes. This process can be time-consuming and may not always yield the best results. For future work, we suggest automating the adjustment of these parameters to improve efficiency and accuracy. By incorporating adaptive mechanisms, the algorithm can dynamically adjust these settings based on the function's characteristics, leading to potentially better performance.

Furthermore, our algorithm currently iterates through a predefined number of transition states and Markov chains. This rigid structure does not account for situations where the optimal solution might be found earlier. Therefore, we propose developing an improved version of the algorithm that can

detect when the optimal solution is reached and stop the iterations at that point. This enhancement would not only save computational resources but also make the algorithm more responsive and effective in finding the optimal solution in a shorter time.

References

- Madigan, R. J. (1995). The language of psychology: APA style as epistemology. *American Psychologist*, 50(6), 428-435.

- Bazerman, C. (1981). What written knowledge does: Three examples of academic discourse. *Philosophy of the Social Sciences*, 11(3), 361- 387.

Appendix

GitHub Link

https://github.com/agra98/Simulated-Annealing-Algorithm-with-Metropolis-criterion