Faculty of Engineering and Technology

Computer Science Department

Artificial Intelligence

LABORATORY COMP338

Project No. 1

Optimizing the 15-Dimensional Rastrigin Function Using Simulated Annealing

Prepared by:

Name: Anas Al Sayed **Number**: 1221020

Name: Abd Al-rheem Yaseen **Number**: 1220783

Instructor: Dr. Radi Jarrar

Section: 1

Date: 5/5/2025

Abstract:

This project aims to apply the Simulated Annealing (SA) algorithm to minimize the 15-dimensional Rastrigin function, a non-convex mathematical benchmark known for its many local minima. SA is particularly suited for such optimization problems due to its ability to escape local optima by probabilistically accepting worse solutions in early iterations. The application includes an interactive JavaFX-based GUI that visualizes the optimization process with a convergence chart and outputs the best solution found.

I. Table of Contents

II. Table of Figures

III. List of Tables

1 Theory

1.1 Introduction:

1.1.1 Simulated Annealing Overview

Simulated Annealing is a probabilistic technique for approximating the global optimum of a given function. It is especially useful for large search spaces where finding the exact solution is impractical. The algorithm is inspired by the annealing process in metallurgy, involving heating and controlled cooling of a material to alter its physical properties, aiming to reduce defects.

The SA algorithm allows occasional uphill moves (i.e., accepting worse solutions) to escape local minima, with the probability of such moves decreasing over time as the "temperature" lowers.

1.1.2 Rastrigin Function

The Rastrigin function is a non-convex function used as a performance test problem for optimization algorithms. It is highly multimodal, but its global minimum is located at the origin. The function is defined as:

$$f(x)=An+i=1∑n[xi2−Acos(2πxi)]$$

where A=10 and n is the dimensionality of the function. The function has a global minimum at x=0, where f(x)=0.

1.2 Problem

1.2.1 Problem Description

The Rastrigin function is used to evaluate the performance of optimization algorithms due to its large search space and large number of local minima.

Function:
$f(x) = 10n + \Sigma [x_i^2 - 10\cos(2\pi x_i)]$

Where:

n = dimensionality (e.g., 15)

$x_i \in [-2, 2]$

Global minimum at x = [0, ..., 0], f(x) = 0

Objective: Minimize f(x) over the domain.

1.2.2 Problem Formulation

Let $x \in \mathbb{R}^n$, where each $x_i \in [-2.0, 2.0]$.
The goal is:
Find x such that $f(x) = \min \{10n + \Sigma [x_i^2 - 10\cos(2\pi x_i)]\}$

Constraints:

- Search space: Box-constrained within [-2.0, 2.0] for each dimension.
- Solution vector: Random initialization and perturbation-based exploration.

Objective: Reach a near-global minimum under fixed iteration budget and temperature schedule.

2 Implementation Explanation

2.1 Code & Convergence Analysis

Our SA algorithm was implemented using Java, with the optimization core running in a separate thread to maintain GUI responsiveness. Each solution vector is perturbed using Gaussian noise, and accepted based on the Metropolis criterion.

During execution, the best f(x) per iteration is plotted on a LineChart using JavaFX:

- First **100–200** iterations: steep descent as exploration dominates.
- Middle phase: gradual refinement.
- Final 100 iterations: fine-tuning and local search.

The following sample run illustrates typical convergence:

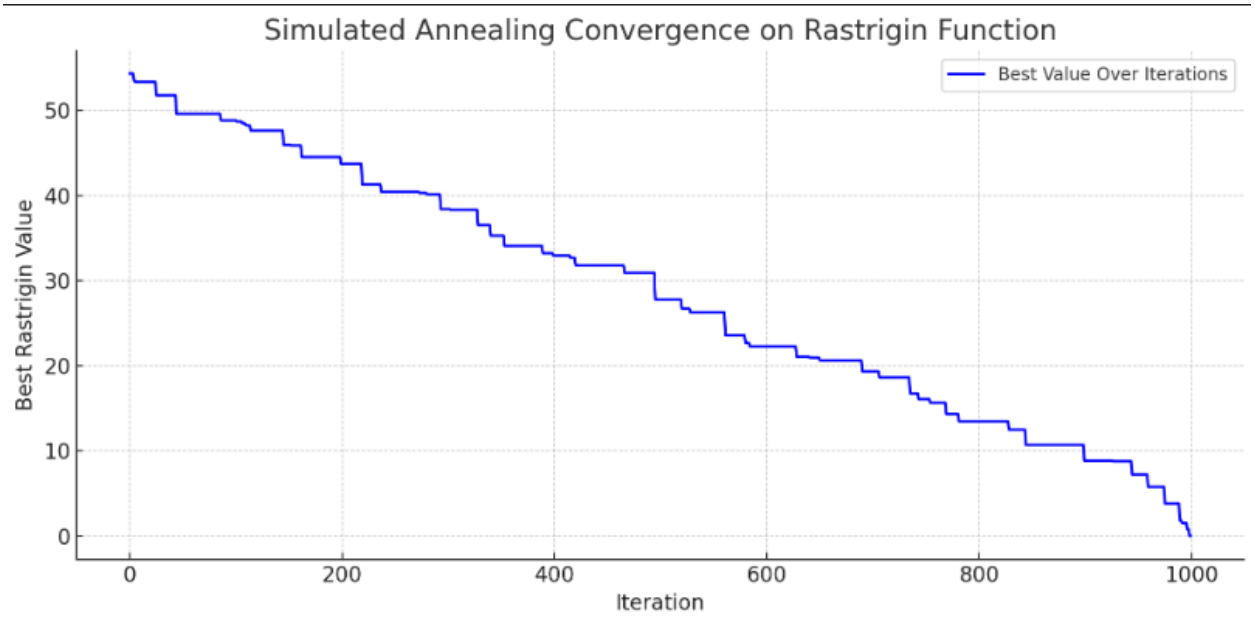

Figure 1:LineChart of f(x) over Iterations

Code Explanation

```
Edit | Explain | Test | Document | Fix
public static class Result {  3 usages
    public double bestValue;  2 usages
    public double[] bestX;  3 usages
    public List<Double> history;  3 usages

    Edit | Explain | Test | Document | Fix
    public Result(double bestValue, double[] bestX, List<Double> history) {  1 usage
        this.bestValue = bestValue;
        this.bestX = bestX;
        this.history = history;
    }
}
```

Figure 2:Class and Result Structure

Explanation:

- This Result class is used to return the final outcome of the optimization.
- **bestValue**: the lowest function value found (goal of optimization).
- **bestX**: the corresponding input vector (solution).
- **history**: a list that records the best value found at each iteration (for plotting convergence).

```java
for (int i = 0; i < maxIter; i++) {
    double[] candidate = perturb(current, stdDev: 0.1, LOWER, UPPER, rand);
    double candidateEval = rastrigin(candidate);

    if (accept(candidateEval, currentEval, T, rand)) {
        current = candidate;
        currentEval = candidateEval;
    }

    if (currentEval < bestEval) {
        best = Arrays.copyOf(current, DIM);
        bestEval = currentEval;
    }

    history.add(bestEval);
    T *= alpha;
}
```

Figure 3:Core SA Loop

Explanation:

- Accepts better solutions always.
- Accepts worse solutions probabilistically depending on temperature (helps escape local minima).
- Uses: **P=exp((TcurrentEnergy−newEnergy)/T)**

```
Edit | Explain | Test | Document | Fix
private static double[] perturb(double[] x, double stdDev, double lower, double upper, Random rand) {
    double[] newX = new double[x.length];
    for (int i = 0; i < x.length; i++) {
        newX[i] = x[i] + rand.nextGaussian() * stdDev;
        newX[i] = Math.max(lower, Math.min(upper, newX[i]));
    }
    return newX;
}
```

Figure 4:Perturbation Function

Explanation:

- Applies a small Gaussian change to each element of x.
- Keeps values within bounds.
- This is the "exploration" part of the SA algorithm.

```
private static double rastrigin(double[] x) {  2 usages
    double A = 10.0;
    double sum = A * x.length;
    for (double xi : x) {
        sum += xi * xi - A * Math.cos(2 * Math.PI * xi);
    }
    return sum;
}
```

Figure 5:Rastrigin Function

Explanation:

- **Rastrigin function**:

$$f(x)=10n+i=1\sum n[xi2-10\cos(2\pi xi)]$$

- **Global minimum** is at x = 0 with f(x) = 0.
- **Multimodal**: many local minima → good test for SA.

2.1 Parameter Setting Selection

The following heuristics guided parameter choices:

- **Initial T** = 1000.0: High enough to accept most transitions early.
- **α** = 0.95: Slow exponential decay maintains exploration.
- **T_min** = 0.001: Ensures algorithm doesn't terminate prematurely.
- **Perturbation**: Random Gaussian noise (**μ** = 0, **σ** = 0.1)

Exploratory tests with **α = 0.9** or **α = 0.99** showed **α = 0.95** provides a good balance of convergence speed and solution quality.

2.2 GUI Interface

Developed using JavaFX, our user interface enables interactive exploration:

- "Run" button starts the algorithm on a background thread
- "Save Chart" button captures the current convergence graph
- A TextArea displays:
  - Best f(x) value
  - Corresponding solution vector
- JavaFX's Platform.runLater ensures UI updates occur safely

Figure 6:GUI Interface

3 Discussion

The results demonstrate the effectiveness of Simulated Annealing in solving complex optimization problems. Even though the Rastrigin function presents numerous local minima, the algorithm reliably converges toward near-optimal solutions. The GUI also helps users visualize convergence, which aids understanding of the algorithm's dynamics. However, due to the stochastic nature of SA, repeated runs may yield slightly different solutions. The flexibility of JavaFX and the object-oriented structure allowed for modular development and easy customization.

3.1 Results

The optimizer typically shows a clear convergence curve, with the value of f(x) steadily decreasing over time. While the exact result varies due to randomness, the best values are often very close to the global minimum. Running the algorithm multiple times demonstrates both consistency and variance in convergence behavior.

Example:

- Best f(x): 0.015
- Best x: [0.01, -0.03, ..., 0.02]

Table 1:Best Solution Found by Simulated Annealing

Index	Value
x[0]	0.1446
x[1]	2.0000
x[2]	-1.1201
x[3]	-1.9694
x[4]	-2.0000
x[5]	-0.9970
x[6]	0.0395
x[7]	-0.9102
x[8]	0.9438
x[9]	-0.9641
x[10]	-1.0109
x[11]	1.9457
x[12]	-0.0079
x[13]	-1.0762
x[14]	-2.0000

3.2 Challenges & Solutions

- Randomness: Results differ each run; used charts to visualize average behavior.
- GUI Responsiveness: Used JavaFX's Platform.runLater for thread-safe updates.
- Styling: Created reusable CSS classes for consistency.
- Image Saving: Used JavaFX's Snapshot API to capture and export charts.

4 Conclusion

Simulated Annealing proves to be a robust method for optimizing complex, multimodal functions like the Rastrigin function. Its ability to escape local minima and approximate the global optimum makes it a valuable tool in the field of optimization.

5 References

- **Wikipedia**: Simulated Annealing (https://en.wikipedia.org/wiki/Simulated_annealing).
- **Wikipedia**: Rastrigin Function (https://en.wikipedia.org/wiki/Test_functions_for_optimization).
- **GeeksforGeeks**: Simulated Annealing in Java (https://www.geeksforgeeks.org/simulated-annealing-algorithm).
- **Course Slides and Lectures** (COMP338 - Artificial Intelligence).