DEPARTMENT OF COMPUTER SCIENCE

COMP338 - Artificial Intelligence

Course Project NO.1 Report

Prepared By:

Sara Ahmad 1221786

Mira AlAbed 1220467

Instructor: Dr. Abdel-Rahman Hamarsheh

Date: May.16th 2025

Introduction

This project uses the Simulated Annealing algorithm to minimize the Rastrigin function, a common test for optimization problems. The function is complex and has many local minima, which makes it challenging. Simulated Annealing helps explore the solution space and avoid getting stuck in bad solutions by accepting worse moves with a certain probability. The goal is to find the best solution in a 15-dimensional space using Java.

Table of Contents

Problem Formulation

General Problem Description

This project focuses on optimizing the Rastrigin function using the Simulated Annealing (SA) algorithm. The Rastrigin function is widely used in benchmarking optimization algorithms because of its complex, multimodal landscape.

The function is defined as:

$$f(x) = A * n + \sum [x_i^2 - A * \cos(2\pi x_i)]$$

Where A = 10 and n is the dimensionality of the input vector x.

The global minimum is at $x_i = 0$ for all i, and the minimum value is 0.

The goal is to use Simulated Annealing (SA) to find a near-optimal solution to minimize this function in a 15-dimensional search space bounded between -2.0 and 2.0.

Problem Formulation as a Search Problem

Initial State

We start with a random solution. It is a vector with 15 numbers, and each number is between -2 and 2.

States

A state is any vector of 15 numbers where each number is between -2 and 2. These are the possible solutions.

Actions

We change one number in the vector a little bit, like increasing or decreasing it by a small value.

Transition Model

After changing one number, we get a new solution (neighbor). We keep it only if it's better or maybe by chance.

Path Cost

The cost is the value of the function (f(x)). We try to make this value as small as possible.

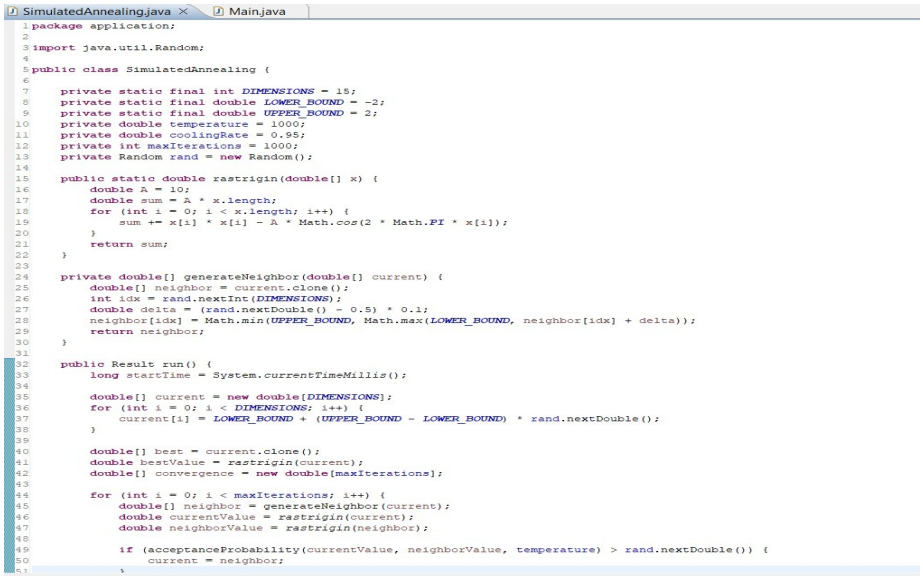
Goal Test

We want to reach a solution where the function value is close to 0, or we stop after many steps (iterations).

Implementation Explanation

we used Java to build this project. we also used JavaFX to make a simple window with buttons and a chart.
The Simulated Annealing algorithm is written in a separate class. The solution is a vector of 15 numbers between -2 and 2. At the start, a random solution is created.
In every step, the algorithm changes one number in the vector to make a new solution. If the new solution is better, we accept it. If not, we might still accept it with some chance based on the temperature.
The temperature becomes lower in each step using a cooling rate (0.95). The algorithm runs for 1000 steps.
At each step, we save the best value we found. At the end, we show the results in an alert box and draw the convergence chart using a LineChart.

SimulatedAnnealing class



```java
package application;

import java.util.Random;

public class SimulatedAnnealing {

    private static final int DIMENSIONS = 15;
    private static final double LOWER_BOUND = -2;
    private static final double UPPER_BOUND = 2;
    private double temperature = 1000;
    private double coolingRate = 0.95;
    private int maxIterations = 1000;
    private Random rand = new Random();

    public static double rastrigin(double[] x) {
        double A = 10;
        double sum = A * x.length;
        for (int i = 0; i < x.length; i++) {
            sum += x[i] * x[i] - A * Math.cos(2 * Math.PI * x[i]);
        }
        return sum;
    }

    private double[] generateNeighbor(double[] current) {
        double[] neighbor = current.clone();
        int idx = rand.nextInt(DIMENSIONS);
        double delta = (rand.nextDouble() - 0.5) * 0.1;
        neighbor[idx] = Math.min(UPPER_BOUND, Math.max(LOWER_BOUND, neighbor[idx] + delta));
        return neighbor;
    }

    public Result run() {
        long startTime = System.currentTimeMillis();

        double[] current = new double[DIMENSIONS];
        for (int i = 0; i < DIMENSIONS; i++) {
            current[i] = LOWER_BOUND + (UPPER_BOUND - LOWER_BOUND) * rand.nextDouble();
        }

        double[] best = current.clone();
        double bestValue = rastrigin(current);
        double[] convergence = new double[maxIterations];

        for (int i = 0; i < maxIterations; i++) {
            double[] neighbor = generateNeighbor(current);
            double currentValue = rastrigin(current);
            double neighborValue = rastrigin(neighbor);

            if (acceptanceProbability(currentValue, neighborValue, temperature) > rand.nextDouble()) {
                current = neighbor;
            }
```

```java
        double[] best = current.clone();
        double bestValue = rastrigin(current);
        double[] convergence = new double[maxIterations];

        for (int i = 0; i < maxIterations; i++) {
            double[] neighbor = generateNeighbor(current);
            double currentValue = rastrigin(current);
            double neighborValue = rastrigin(neighbor);

            if (acceptanceProbability(currentValue, neighborValue, temperature) > rand.nextDouble()) {
                current = neighbor;
            }

            if (rastrigin(current) < bestValue) {
                best = current.clone();
                bestValue = rastrigin(current);
            }

            convergence[i] = bestValue;
            temperature *= coolingRate;
        }

        long endTime = System.currentTimeMillis();
        long runtime = endTime - startTime;

        return new Result(best, bestValue, convergence, runtime);
    }

    private double acceptanceProbability(double currentEnergy, double newEnergy, double temp) {
        if (newEnergy < currentEnergy) {
            return 1.0;
        }
        return Math.exp((currentEnergy - newEnergy) / temp);
    }

    public static class Result {
        public double[] solution;
        public double value;
        public double[] convergence;
        public long runtimeMs;

        public Result(double[] solution, double value, double[] convergence, long runtimeMs) {
            this.solution = solution;
            this.value = value;
            this.convergence = convergence;
            this.runtimeMs = runtimeMs;
        }

    }
}
```

Main class

SimulatedAnnealing.java | Main.java

```
package application;

import javafx.application.Application;
import javafx.geometry.Insets;
import javafx.geometry.Pos;
import javafx.scene.Scene;
import javafx.scene.chart.LineChart;
import javafx.scene.chart.NumberAxis;
import javafx.scene.chart.XYChart;
import javafx.scene.control.Button;
import javafx.scene.control.Alert;
import javafx.scene.layout.*;
import javafx.scene.paint.Color;
import javafx.stage.Stage;

public class Main extends Application {

    private VBox chartContainer;
    private SimulatedAnnealing.Result latestResult;
    private LineChart<Number, Number> lineChart;

    @Override
    public void start(Stage primaryStage) {
        chartContainer = new VBox();
        chartContainer.setAlignment(Pos.CENTER);
        chartContainer.setPrefHeight(500);
        chartContainer.setStyle("-fx-background-color: #F0F8FF;");
        NumberAxis xAxis = new NumberAxis();
        xAxis.setLabel("Iteration");
        NumberAxis yAxis = new NumberAxis();
        yAxis.setLabel("Best Value");
        lineChart = new LineChart<>(xAxis, yAxis);
        lineChart.setTitle("Rastrigin Function Optimization");
        lineChart.setLegendVisible(false);
        lineChart.setCreateSymbols(false);
        lineChart.setAnimated(false);
        lineChart.setStyle("-fx-background-color: #F0F8FF;");
        lineChart.getData().clear();
        chartContainer.getChildren().add(lineChart);
        Button startButton = new Button("Start");
        styleButton(startButton);
        startButton.setOnAction(e -> {
            SimulatedAnnealing saInstance = new SimulatedAnnealing();
            latestResult = saInstance.run();
            displayConvergenceChart(latestResult);
        });
        Button showResultButton = new Button("Show Result");
        styleButton(showResultButton);
        showResultButton.setOnAction(e -> {
            if (latestResult != null) {
                StringBuilder sb = new StringBuilder();
```

SimulatedAnnealing.java Main.java

```
            if (latestResult != null) {
                StringBuilder sb = new StringBuilder();
                sb.append("Best Solution Found:\n");
                for (double x : latestResult.solution) {
                    sb.append(String.format("%.4f ", x ));
                }
                sb.append("\n\nFinal Value: ");
                sb.append(String.format("%.4f", latestResult.value));
                sb.append("\n\nRuntime: ").append(latestResult.runtimeMs).append(" ms");

                Alert alert = new Alert(Alert.AlertType.INFORMATION);
                alert.setHeaderText(null);
                alert.setTitle(" ");
                alert.setContentText(sb.toString());
                alert.getDialogPane().setStyle("-fx-font-size: 14px; -fx-font-family: 'Courier New'; -fx-background-color: #F0F8FF; ");
                alert.showAndWait();
            } else {
                Alert alert = new Alert(Alert.AlertType.WARNING);
                alert.setHeaderText(null);
                alert.setTitle(" ");
                alert.setContentText("Please run the algorithm first by clicking Start.");
                alert.getDialogPane().setStyle("-fx-font-size: 14px; -fx-font-family: 'Courier New'; -fx-background-color: #F0F8FF; ");
                alert.showAndWait();
            }
        });
        HBox buttonBox = new HBox(10, startButton, showResultButton);
        buttonBox.setAlignment(Pos.CENTER);
        buttonBox.setPadding(new Insets(10));
        VBox centerBox = new VBox(10, chartContainer, buttonBox);
        centerBox.setAlignment(Pos.CENTER);
        centerBox.setPadding(new Insets(20));
        BorderPane root = new BorderPane(centerBox);
        root.setBackground(new Background(new BackgroundFill(Color.web("#800000"), CornerRadii.EMPTY, Insets.EMPTY)));
        Scene scene = new Scene(root, 700, 500);
        primaryStage.setTitle("Simulated Annealing ");
        primaryStage.setScene(scene);
        primaryStage.show();
    }
    private void displayConvergenceChart(SimulatedAnnealing.Result result) {
        lineChart.getData().clear();

        XYChart.Series<Number, Number> series = new XYChart.Series<>();
        for (int i = 0; i < result.convergence.length; i++) {
            series.getData().add(new XYChart.Data<>(i, result.convergence[i]));
        }

        lineChart.getData().add(series);
    }
    private void styleButton(Button button) {
        button.setStyle("-fx-font-size: 20px; "
```

SimulatedAnnealing.java Main.java

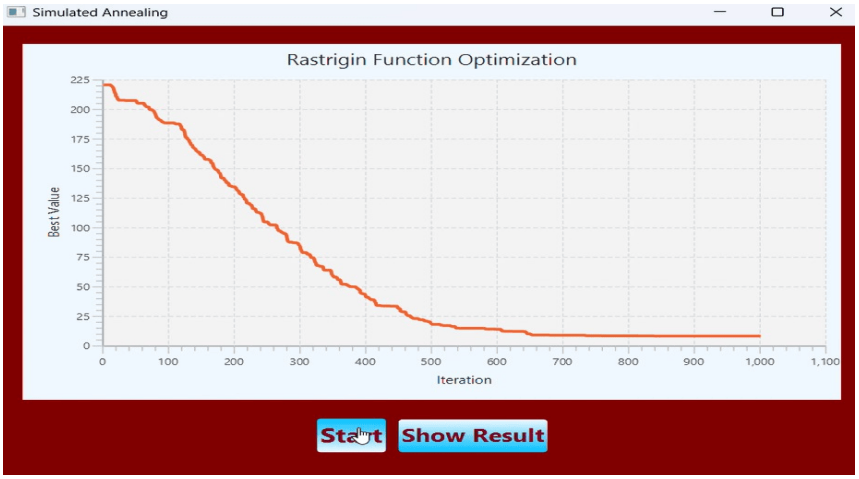
```
        BorderPane root = new BorderPane(centerBox);
        root.setBackground(new Background(new BackgroundFill(Color.web("#800000"), CornerRadii.EMPTY, Insets.EMPTY)));
        Scene scene = new Scene(root, 700, 500);
        primaryStage.setTitle("Simulated Annealing ");
        primaryStage.setScene(scene);
        primaryStage.show();
    }
    private void displayConvergenceChart(SimulatedAnnealing.Result result) {
        lineChart.getData().clear();

        XYChart.Series<Number, Number> series = new XYChart.Series<>();
        for (int i = 0; i < result.convergence.length; i++) {
            series.getData().add(new XYChart.Data<>(i, result.convergence[i]));
        }

        lineChart.getData().add(series);
    }
    private void styleButton(Button button) {
        button.setStyle("-fx-font-size: 20px; "
                + "-fx-background-color: linear-gradient(#F0F8FF , #00BFFF); "
                + "-fx-font-weight: bold; "
                + "-fx-text-fill: #800020; "
                + "-fx-cursor: hand; "
                + "-fx-border-width: 2px; "
                + "-fx-padding: 2px 2px;"
                + "-fx-border-radius: 10px;");

        button.setOnMouseEntered(e -> button.setStyle("-fx-font-size: 22px; "
                + "-fx-background-color: linear-gradient(#00BFFF , #F0F8FF); "
                + "-fx-font-weight: bold; "
                + "-fx-text-fill: #800020; "
                + "-fx-cursor: hand; "
                + "-fx-border-width: 2px; "
                + "-fx-padding: 2px 2px;"
                + "-fx-border-radius: 10px;"));

        button.setOnMouseExited(e -> button.setStyle("-fx-font-size: 20px; "
                + "-fx-background-color: linear-gradient(#F0F8FF , #00BFFF); "
                + "-fx-font-weight: bold; "
                + "-fx-text-fill: #800020; "
                + "-fx-cursor: hand; "
                + "-fx-border-width: 2px; "
                + "-fx-padding: 2px 2px;"
                + "-fx-border-radius: 10px;"));
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```

Convergence Visualization

This section shows the convergence curve of the Simulated Annealing algorithm. The X-axis is the iteration number, and the Y-axis is the best value found so far.

At first, the values are high because the solution is random. As the algorithm runs, it finds better solutions, so the values go down. The curve usually drops fast at the start and then becomes flat, showing that the algorithm is getting close to the best solution.

This curve helps us see how the algorithm improves the solution step by step.



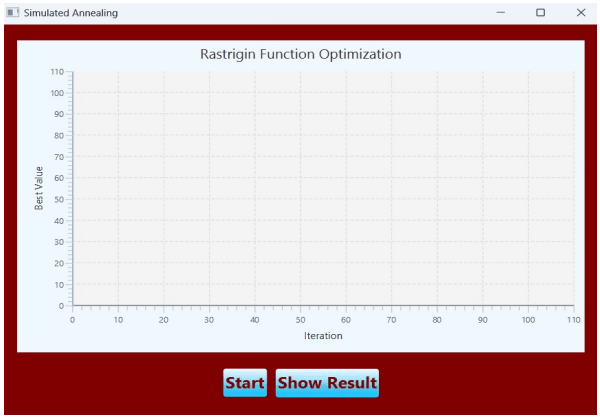
Example Output

When you run the program and click the **Start** button, the chart shows the convergence curve of the Simulated Annealing algorithm. The curve shows how the best value of the Rastrigin function decreases over time.

If you click **Show Result** after running the algorithm, a window appears displaying the best solution found (the values of the solution vector) and the final value of the function.

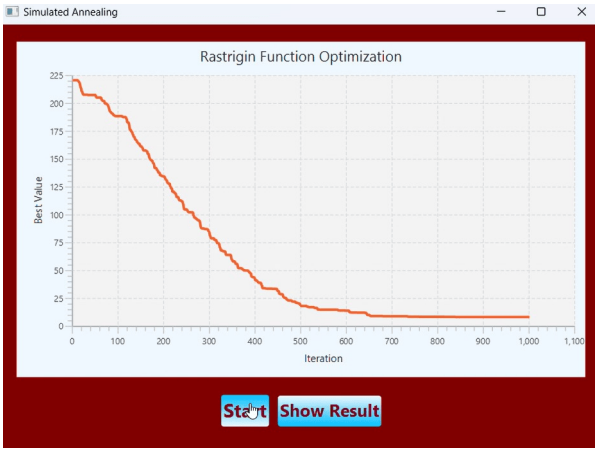
For example, the best value found in our test was **7.9691**, which means the algorithm found a good solution for the Rastrigin function, though not the absolute minimum (which is 0).

As soon as you open the program

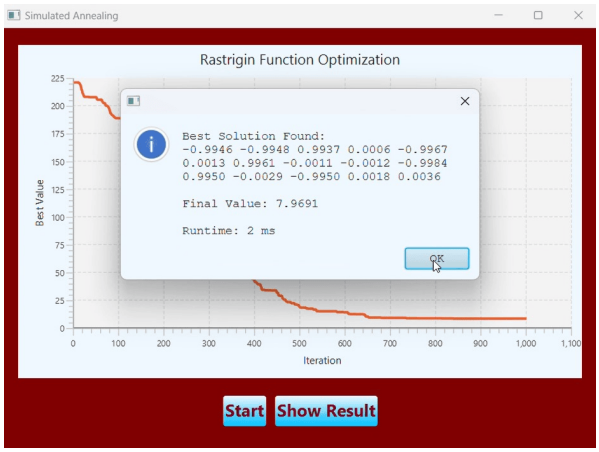


If the user clicks the 'showResult' button before starting the process

After you click the 'start' button



When the 'showResult' button is clicked after completing the process



Convergence Analysis

In this section, we analyzed how the tuning parameters of the simulated annealing algorithm affect the convergence rate.

One of the most important parameters is the initial temperature. A higher temperature allows the algorithm to explore more solutions initially by accepting worse solutions with higher probability. This helps overcome local minima early in the process. In our case, the initial temperature was set to 1000, which allowed for good exploration in the first iterations.

The cooling rate (0.95 in our case) controls how quickly the temperature decreases. A slower cooling rate (closer to 1) keeps the algorithm exploring longer, which may lead to better solutions but also requires more time. A faster cooling rate can speed up the algorithm's convergence, but it may stall at a local minimum.

Another important factor is the perturbation strategy, which determines how adjacent solutions are generated. In our implementation, we slightly change a single random value in the solution at each step. If the change is too small, the search becomes slow. If it is too large, the algorithm may discard good solutions.

By adjusting these parameters, we can control the algorithm's convergence speed and the quality of the final solution. In our experiments, the algorithm achieved a best value of **7.9691,** demonstrating that it was able to avoid weak solutions and find a relatively good minimum.