Aish Srinivas
20720876
ECE 457A

Problem 1: Simulated Annealing on Easom Function

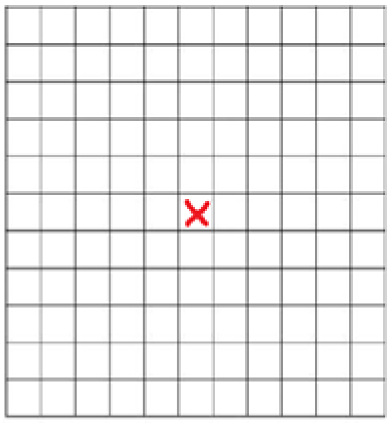
TABLE OF CONTENTS

1. Problem Formulation

- Initial state = a random point (x, y) with cost f(x, y) within the bounds [-100, 100]
- Goal state = a point (x, y) that gives min f(x, y) which is the global minimum of the function.
- State = a point (x, y) and its cost f(x, y)
- Actions = Evaluate function at neighboring random point and check if it is lower than current solution. If it is, accept it. Else, check against acceptance probability, and accept solutions. Iterate until the maximum number of iterations has been reached for that temperature. Reduce accepting worse solutions as temperature decreases so that near the end, we are mostly looking for better solutions.
- Cost = Easom function: f(x,y)= -cos(x1)cos(x1)e^(-((x1-π)^2-(x2-π)^2))
- Neighbourhood Function: N(x1,x2)=random(max(x1 - 5,-100),min(x1 + 5,100))

  A random point in the below neighborhood will be chosen and will be evaluated. To move across the [100, 100] grid quickly, a neighborhood value of 5 was chosen, so that we can diversify our search.

Aish Srinivas
20720876
ECE 457A



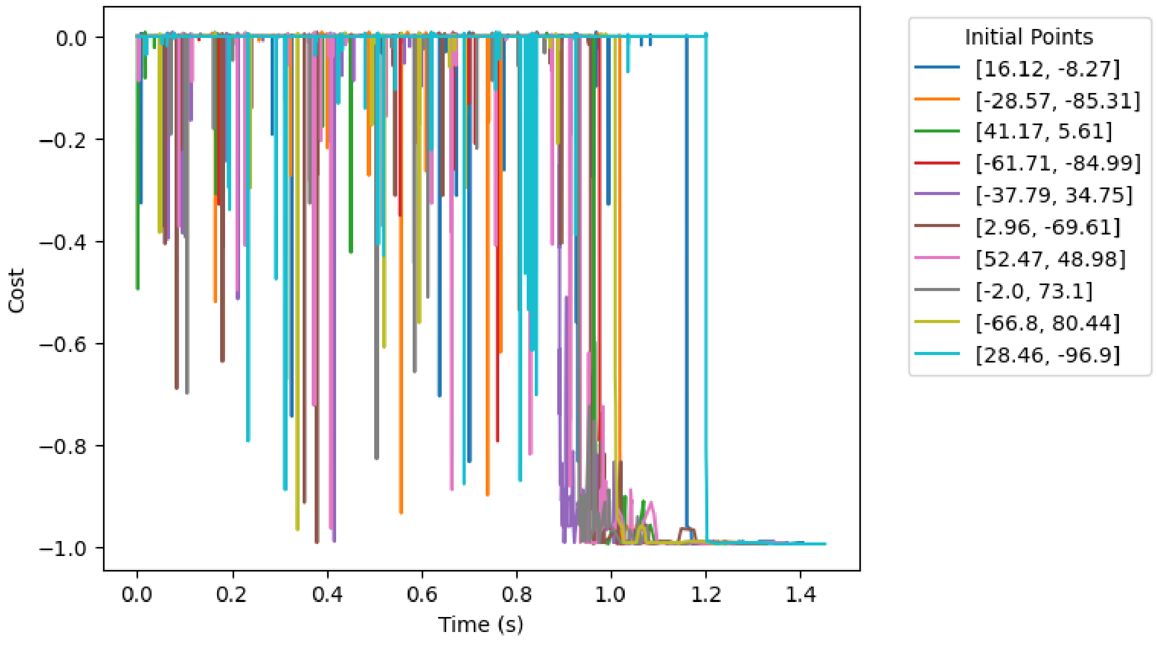
2. Algorithm / Pseudo-Code

```
// cost function
Def function(x1, x2):
        Return -cos(x1)*cos(x2)*exp(-(x1-pi)**2-(x2-pi)**2)

Def simulated_annealing(iteration, Initial_temperature, alpha, Initial_point):
        Bounds = [-100, 100]
        Number of iterations = 1000
        Step size = 0.01
        Initial_temperature = 500
        Alpha = 0.95

        Initial_point = [x, y]
        best_point = Initial_point
        best_solution = cost(Initial)
        Current_point, current_solution = best _point, best _solution

        While (temperature > 0.001):
        While (i < number of iterations):
                Neighbour_point = [rand (max(Current_point - 5, -100),
          min(Current_point + 5, 100))]
        Neighbour _solution = cost (Initial)
        Difference  = Neighbour_solution - best_solution
```

```
// check if value is smaller than minimum we found so far
If (Difference  < 0):
        best_solution = Neighbour _solution
curr_point, current_solution = Neighbor_point, Neighbour _solution
else:
probability_function = e^((difference)/(alpha*temperature))
if rand() < probability_function:
        Current_point, current_solution = Neighbour _point, Neighbour _solution
                temperature = temperature *alpha
        Return best_point, best_solution
```

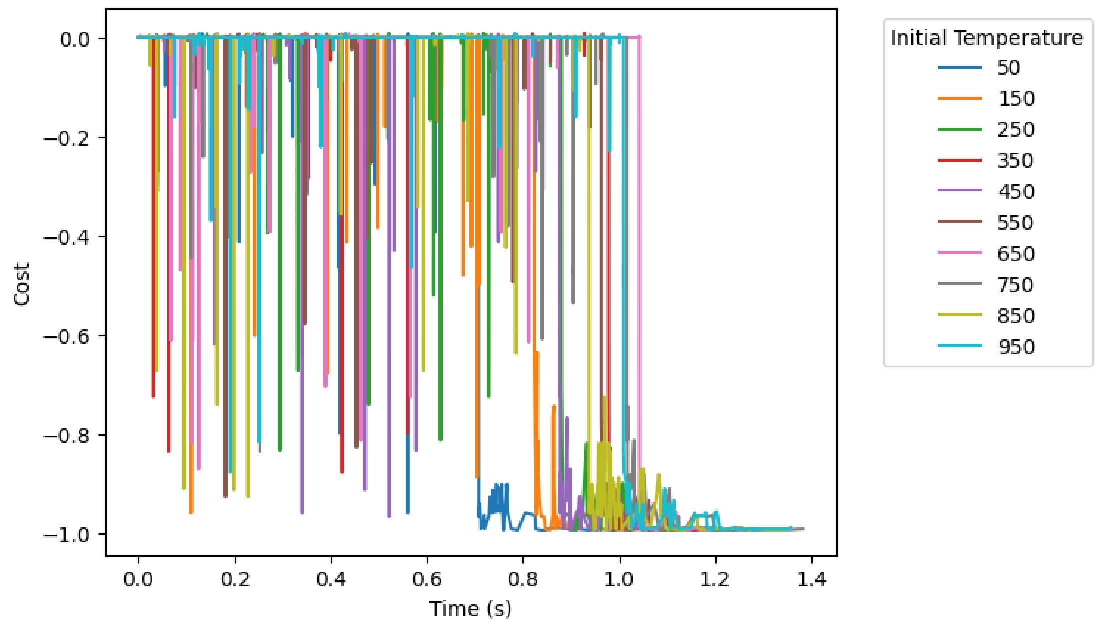
3.1 Experiment 1: Selecting 10 different initial points randomly in [-100, 100]

Fixed Parameters: Initial Temperature = 500, Annealing Rate = 0.95 (very slow)



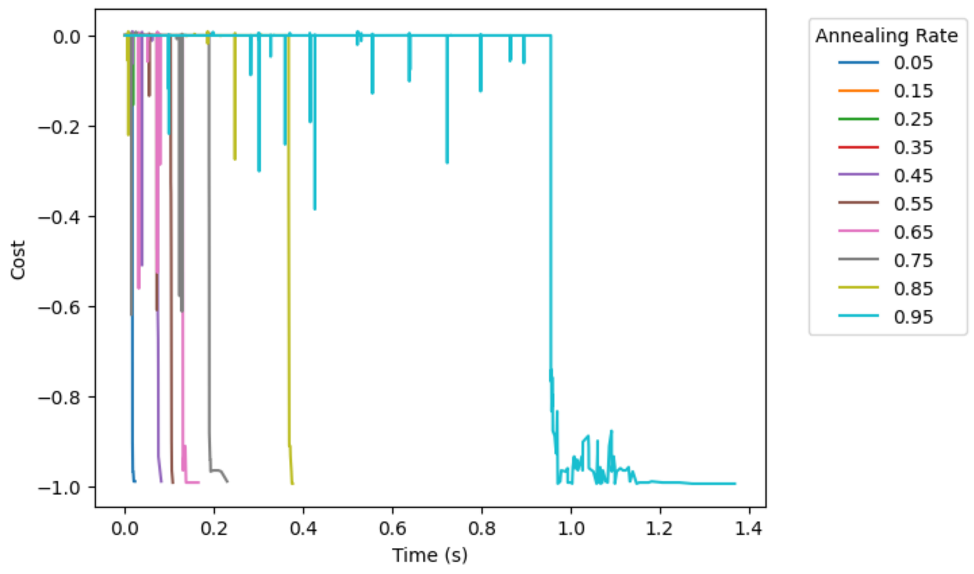
3.2. Experiment 2: Selecting 10 different initial temperatures in a reasonable range

Fixed Parameters: Initial Point = [-100, 100], Annealing Rate = 0.95 (very slow)



3.3. Experiment 3: Selecting 10 different annealing schedules.

Fixed Parameters: Initial Point = [-100, 100], Initial Temperature = 500



4. SA Performance

Given that the experiments were able to reach the goal state, below are the average times each experiment took:

Experiment 1 – 1.4 s
Experiment 2 – 1.4 s
Experiment 3 – A slowly decreasing temperature increases the execution time for the algorithm. In the above graph, an annealing rate of 0.99 took about 1.4 s to run, while the others either got stuck on local optima or took less than 0.4 s to run. If the temperature is decreased slowly, we have more time to explore different solutions and less chance of getting stuck in local optima.

The bounds on the two loops in the algorithms are dependent on initial temperature, the annealing rate, and the number of iterations that can happen at a particular temperature. A slow performance would be the result of:

- Very high temperature
- High number of iterations
- Very slow decrease in annealing rate

5. Best Solution and its parameters

Some solutions got stuck in local optima, meanwhile others were able to get as close as possible to the global minimum. Analyzing the graphs from above, the best solution had the following parameters:

Initial Temperature = 500
Final Temperature = 0.001
Initial Point = any point on within the bounds
Annealing Rate = 0.95
Neighborhood size = 5
Number of iterations = 1000

Output: Easom([3.1, 3.1]) = -0.994800

Comments

- Because this function has a very wide search space that decays to 0 and a small region where the global minimum lies, it is best if we can reach the area around the global minimum in time. Therefore the neighborhood function chooses a random solution from a large enough neighborhood to proceed forward. A neighborhood size of 5 works well in this instance.
- We need enough time to search, which is why the temperature should be set at a value below 500. Larger temperatures resulted in accepting worse solutions, so the search never progressed as seen from the graph.
- Difference = new solution's cost – current solution's cost. For the Easom function, this value was an extremely small number as most of the search space evaluates to values close to 0. To get a reasonable acceptance probability, the temperature should stay low.
- If the temperature decreases too quickly, we will not have enough time to explore the search space and reach our goal. Therefore, the annealing rate should be very small.
- Many iterations are required for the system to be stable and to give enough time to explore the search space. 1000 is an appropriate number to use.

5.1. Sample Output

```
Varying initial points ...
T = 500
Annealing Rate = 0.95
Initial Point = [75.82, -19.61], f([3.1, 3.1]) = -0.994800
Initial Point = [20.16, -87.11], f([3.1, 3.1]) = -0.994800
Initial Point = [-24.41, -89.22], f([3.1, 3.1]) = -0.994800
Initial Point = [35.73, -29.43], f([3.1, 3.1]) = -0.994800
Initial Point = [65.8, 59.45], f([3.1, 3.1]) = -0.994800
Initial Point = [-35.93, -22.34], f([3.1, 3.1]) = -0.994800
Initial Point = [-30.42, -88.55], f([3.1, 3.1]) = -0.994800
Initial Point = [43.54, 80.18], f([3.1, 3.1]) = -0.994800
Initial Point = [-94.62, -24.78], f([3.1, 3.1]) = -0.994800
Initial Point = [-62.36, 24.56], f([3.1, 3.1]) = -0.994800

Varying initial temperatures ...
Annealing Rate = 0.95
Initial Point = [-100, 100]
Initial Temperature = 50, f([3.1, 3.1]) = -0.994800
Initial Temperature = 150, f([3.1, 3.1]) = -0.994800
Initial Temperature = 250, f([3.1, 3.1]) = -0.994800
Initial Temperature = 350, f([3.1, 3.1]) = -0.994800
Initial Temperature = 450, f([3.1, 3.1]) = -0.994800
Initial Temperature = 550, f([3.1, 3.1]) = -0.994800
Initial Temperature = 650, f([3.1, 3.1]) = -0.994800
Initial Temperature = 750, f([3.1, 3.1]) = -0.994800
Initial Temperature = 850, f([3.1, 3.1]) = -0.994800
Initial Temperature = 950, f([3.1, 3.1]) = -0.994800

Varying initial annealing parameter ...
T = 500
Initial Point = [-100, 100]
Annealing Rate = 0.05, f([24.6, -12.2]) = -0.000000
Annealing Rate = 0.15, f([13.9, -17.5]) = -0.000000
Annealing Rate = 0.25, f([-23.1, -2.0]) = -0.000000
Annealing Rate = 0.35, f([-20.1, 13.3]) = -0.000000
Annealing Rate = 0.45, f([3.1, 3.1]) = -0.994800
Annealing Rate = 0.55, f([3.2, 3.1]) = -0.992300
Annealing Rate = 0.65, f([17.6, 26.2]) = -0.000000
Annealing Rate = 0.75, f([3.1, 3.1]) = -0.994800
Annealing Rate = 0.85, f([3.1, 3.1]) = -0.994800
Annealing Rate = 0.95, f([3.1, 3.1]) = -0.994800


Process finished with exit code 0
```

5.3. Time and Memory Complexity

Time Complexity

In Simulated Annealing, the temperature is decreased by a value α. α is chosen to be 0.01, so that the algorithm can run for a longer time and the temperature can be decreased slowly. The formula for decrease in temperature T is:

$T=T\cdot \alpha^x$

Since this is an exponential decaying function, it never approaches 0, but we want the program to terminate. As T -> 0, the algorithm will act purely like hill climbing, only searching for the global minimum. To make sure the algorithm does not run forever, we set the loop to terminate when $T \approx 0.001$.

Each temperature will run 1000 iterations, that is, it will look at 1000 potential solutions each loop.
To know how many times the outer while loop (while (T > 0.001)) will run:

$0.001 = T\cdot \alpha^x$
$\log(0.001) = \log(T) \cdot x \cdot \log(\alpha)$
$\log(0.001) - \log(T) = x \cdot \log(\alpha)$
$\log(0.001)/\log(\alpha) - \log(T)/\log(\alpha) = x$

As T -> ∞, the constant log(0.001)/log(α) disappears, and this function will represent log n. Therefore, the worst time complexity of this function is O(log n).

The number of iterations per temperature stays at 1000, so the inner loop has O(n) complexity. Together, the overall worst case time complexity of this algorithm is O(nlogn).

Memory Complexity

In the simulated annealing algorithm, the timestamp and cost of the accepted solutions are pushed into arrays so the data can be plotted like in question 3. So, the average and worst case space complexity will be equal to O(n).

Ignoring the data being used for the graphs, the SA algorithm will only have O(1) memory complexity as it uses a Point (x, y) data structure to represent a solution.