Soft Computing Project Report

The LNM Institute of Information Technology

Team Members: Shivang Negi 22UCS200, Anshit Mahajan 22UCS024, Pranav Shrivastava 22UCS151, Krishna Manchanda 22UCS111
Submission Date: 20/04/2025

Executive Summary

This project implements two key soft computing techniques to solve real-world problems: a Fuzzy Logic Controller (FLC) for determining washing machine wash times, and a Genetic Algorithm (GA) for solving the Traveling Salesman Problem (TSP). Both implementations demonstrate the power of computational intelligence approaches in handling uncertainty and optimization challenges. The fuzzy logic controller successfully maps imprecise inputs (dirt and grease levels) to appropriate wash times, while the genetic algorithm effectively finds near-optimal routes for the complex TSP problem.

1. Problem 1: Fuzzy Logic Controller for Washing Machine

1.1 Objective

Design a fuzzy logic controller (FLC) using Mamdani's approach to determine the wash time of a domestic washing machine based on two inputs:

- Dirt level (0–100)
- Grease level (0–50)

1.2 Methodology

1.2.1 Fuzzification

Membership Functions: Triangular functions were used for fuzzification due to their simplicity and computational efficiency.

Dirt (5 descriptors):

- Very Small Dirt (VSD): (0, 0, 25)
- Small Dirt (SD): (0, 25, 50)
- Medium Dirt (MD): (25, 50, 75)
- Heavy Dirt (HD): (50, 75, 100)
- Very Heavy Dirt (VHD): (75, 100, 100)

Grease (3 descriptors):

- Small Grease (SG): (0, 0, 25)
- Medium Grease (MG): (0, 25, 50)
- Heavy Grease (HG): (25, 50, 50)

Output (Wash Time):

- Very Short Time (VST): (0, 0, 15)
- Short Time (ST): (0, 15, 30)
- Medium Time (MT): (15, 30, 45)
- Long Time (HT): (30, 45, 60)
- Very Long Time (VHT): (45, 60, 60)

1.2.2 Fuzzy Rule Base

A rule table was implemented to map input combinations to output descriptors:

Dirt\Grease	SG	MG	HG
VSD	VST	VST	ST
SD	VST	ST	MT
MD	ST	MT	HT
HD	MT	HT	VHT
VHD	HT	VHT	VHT

1.2.3 Fuzzy Inference

Max-Min Composition: For each input pair (dirt, grease), the activation strength of each rule was computed as the minimum of the corresponding membership values. The output membership function was clipped at this strength, and the results were aggregated using the maximum operator.

This approach allows the system to handle multiple rules firing simultaneously and combines their effects based on the strength of each rule's activation.

1.2.4 Defuzzification

Center of Gravity (CoG): The crisp output (wash time) was calculated by integrating the aggregated output membership function over the range [0, 60] and dividing by the total area.

Formula:

$$\text{Wash Time} = \sum(x \cdot \mu_{aggregated}(x)) \;/\; \sum(\mu_{aggregated}(x))$$

This was implemented numerically by discretizing the output range into 1000 points for accurate approximation of the continuous mathematical formula.

1.3 Implementation Details

The fuzzy logic controller was implemented in Python. Key components include:

1. **Triangular membership function** - A function to calculate membership values using triangular functions defined by three points (a, b, c).
2. **Membership functions for inputs** - Functions to calculate degree of membership for dirt and grease levels across their linguistic variables.
3. **Rule base** - A dictionary mapping input combinations to output descriptors according to the rule table.
4. **Inference engine** - The `infer_output` function implementing max-min composition to determine output activation levels.
5. **Defuzzification** - The `defuzzify` function implementing the Center of Gravity method to convert the fuzzy output to a crisp value.

1.4 Results

Example: For dirt = 40, grease = 30, the output is MT with a crisp value of 27.50 minutes.

The system successfully maps ranges of input values to appropriate wash times, dealing effectively with the uncertainty inherent in describing dirt and grease levels.

```
PS C:\Users\91827> python -u "c:\Users\91827\OneDrive\Desktop\pythonRush\problem1_FLC.py"
Enter dirt level (0-100): 40
Enter grease level (0-50): 30

Wash Time: 27.50 minutes
PS C:\Users\91827> python -u "c:\Users\91827\OneDrive\Desktop\pythonRush\problem1_FLC.py"
Enter dirt level (0-100): 50
Enter grease level (0-50): 20

Wash Time: 26.38 minutes
PS C:\Users\91827> python -u "c:\Users\91827\OneDrive\Desktop\pythonRush\problem1_FLC.py"
Enter dirt level (0-100): 100
Enter grease level (0-50): 45

Wash Time: 54.85 minutes
PS C:\Users\91827> python -u "c:\Users\91827\OneDrive\Desktop\pythonRush\problem1_FLC.py"
Enter dirt level (0-100): 10
Enter grease level (0-50): 5

Wash Time: 10.53 minutes
PS C:\Users\91827> python -u "c:\Users\91827\OneDrive\Desktop\pythonRush\problem1_FLC.py"
Enter dirt level (0-100): 60
Enter grease level (0-50): 40

Wash Time: 39.89 minutes
PS C:\Users\91827>
```

2. Problem 2: Traveling Salesman Problem (TSP) using Genetic Algorithm

2.1 Objective

Find the shortest route for a salesman to visit 15 cities exactly once and return to the starting city, using a Genetic Algorithm (GA).

2.2 Methodology

2.2.1 Initialization

Population: 15 chromosomes (routes), each a random permutation of cities [0, 1, ..., 14].

Random permutations of cities were generated with the help of **random** module in Python.

Each chromosome represents a complete tour visiting all cities.

2.2.2 Fitness Calculation (Objective Function)

Objective: Minimize total distance.

Fitness Function:

```
Fitness = 1 / Total Distance
```

Where Total Distance includes the return trip to the starting city.

A higher fitness value indicates a better solution (shorter route).

2.2.3 Selection

Roulette Wheel Selection: Chromosomes were selected probabilistically based on fitness. Higher fitness = higher selection probability.

This approach gives better solutions a higher chance of being selected for reproduction while still maintaining diversity in the population.

2.2.4 Crossover

One-Point Crossover:

- Randomly select a crossover point (using random module).
- Offspring 1: First half of Parent 1 + second half of Parent 2.
- Offspring 2: First half of Parent 2 + second half of Parent 1.

Rectification: Duplicate cities in offspring were replaced with missing cities to ensure validity. This step is crucial because the TSP requires each city to be visited exactly once.

2.2.5 Mutation

Inorder Swap: Two random cities in a chromosome were swapped (e.g., swap cities at indices 3 and 10).

Mutation introduces diversity into the population and helps prevent premature convergence to local optima.

2.2.6 Elitism Model

The best solution from each generation was preserved by replacing the worst solution in the next generation. This ensures that the best solution found so far is never lost through the evolutionary process.

2.3 Implementation Details

The genetic algorithm for TSP was implemented in Python using object-oriented programming. Key components include:

1. **GeneticAlgorithmTSP class** - The main class that encapsulates the entire algorithm.
2. **Initialize population** - Creates random permutations of cities as the initial population.
3. **Fitness calculation** - Computes the total distance for each route (including return to start).
4. **Selection mechanism** - Implements roulette wheel selection based on fitness values.
5. **Crossover and mutation operators** - Implements one-point crossover and inorder swap mutation.
6. **Chromosome rectification** - Ensures each city appears exactly once in each solution.
7. **Algorithm execution** - The `run` method that orchestrates the evolutionary process over multiple generations.

2.4 Results

Input: User can manually pass a weight matrix (15×15) or can choose to pass a randomly generated weight matrix (15×15).

Output: Best route after 20 iterations with total distance of that route.

```
PS C:\Users\91827> python -u "c:\Users\91827\OneDrive\Desktop\pythonRush\problem2_TSP.py"
Do you want to use a random weight matrix? (yes/no): No
Enter your 15x15 matrix row by row (each row separated by newline):
Example for row 1: 0 10 15 20 ... (15 numbers separated by spaces)
Row 1: 0  29  20  21  16  31 100  12   4  31  18  23   5  48  27
Row 2: 29   0  15  29  28  40  72  21  29  41  12  23  23  23  31
Row 3: 20  15   0  15  14  25  81   9  23  27  13  16  17  15  28
Row 4: 21  29  15   0   4  12  92  12  25  13  25  24  15  19  22
Row 5: 16  28  14   4   0  16  94   9  20  16  22  20  13  18  24
Row 6: 31  40  25  12  16   0  95  24  36   3  37  29  26  20  30
Row 7: 100 72  81  92  94  95   0  90 101  99  84  78  89  93  87
Row 8: 12  21   9  12   9  24  90   0  15  25  16  17   6  20  19
Row 9:  4  29  23  25  20  36 101  15   0  35  22  25   9  27  21
Row 10: 31  41  27  13  16   3  99  25  35   0  38  30  27  21  31
Row 11: 18  12  13  25  22  37  84  16  22  38   0  15  20  26  24
Row 12: 23  23  16  24  20  29  78  17  25  30  15   0  21  18  27
Row 13:  5  23  17  15  13  26  89   6   9  27  20  21   0  25  20
Row 14: 48  23  15  19  18  20  93  20  27  21  26  18  25   0  22
Row 15: 27  31  28  22  24  30  87  19  21  31  24  27  20  22   0
Iteration 1: Best Distance = 409.00
Iteration 2: Best Distance = 401.00
Iteration 3: Best Distance = 386.00
Iteration 4: Best Distance = 366.00
Iteration 5: Best Distance = 366.00
Iteration 6: Best Distance = 366.00
Iteration 7: Best Distance = 361.00
Iteration 8: Best Distance = 361.00
Iteration 9: Best Distance = 361.00
Iteration 10: Best Distance = 361.00
Iteration 11: Best Distance = 361.00
Iteration 12: Best Distance = 361.00
Iteration 13: Best Distance = 361.00
Iteration 14: Best Distance = 361.00
Iteration 15: Best Distance = 361.00
Iteration 16: Best Distance = 361.00
Iteration 17: Best Distance = 361.00
Iteration 18: Best Distance = 361.00
Iteration 19: Best Distance = 361.00
Iteration 20: Best Distance = 354.00

Best route found: [3, 13, 11, 7, 10, 1, 6, 5, 9, 4, 12, 0, 8, 14, 2]
Total distance: 354
PS C:\Users\91827>
```

```
PS C:\Users\91827> python -u "c:\Users\91827\OneDrive\Desktop\pythonRush\problem2_TSP.py"
Do you want to use a random weight matrix? (yes/no): yes

Generated random weight matrix:
[[ 0 60 75 59 43 72 79 50 52 53 32 39 21 66 19]
 [60  0 68 19 58 56 31 69 64 40 27 56 68 67 31]
 [75 68  0 42 79 69 53 66 53 86 32 48 48 45 72]
 [59 19 42  0 79 47 58 59 59 17 65 53 58 37 35]
 [43 58 79 79  0 41 41 50 23 49 29 58 45 37 99]
 [72 56 69 47 41  0 37 49 79 57 46 71 82 50 66]
 [79 31 53 58 41 37  0 34 75 81 69 59 32 54 44]
 [50 69 66 59 50 49 34  0 44 19 65 54 38 73 61]
 [52 64 53 59 23 79 75 44  0 97 56 79 54 88 44]
 [53 40 86 17 49 57 81 19 97  0 90 79 52 64 38]
 [32 27 32 65 29 46 69 65 56 90  0 73 69 32 77]
 [39 56 48 53 58 71 59 54 79 79 73  0 52 41 75]
 [21 68 48 58 45 82 32 38 54 52 69 52  0 80 43]
 [66 67 45 37 37 50 54 73 88 64 32 41 80  0 28]
 [19 31 72 35 99 66 44 61 44 38 77 75 43 28  0]]
Iteration 1: Best Distance = 671.00
Iteration 2: Best Distance = 671.00
Iteration 3: Best Distance = 671.00
Iteration 4: Best Distance = 671.00
Iteration 5: Best Distance = 671.00
Iteration 6: Best Distance = 638.00
Iteration 7: Best Distance = 626.00
Iteration 8: Best Distance = 626.00
Iteration 9: Best Distance = 626.00
Iteration 10: Best Distance = 626.00
Iteration 11: Best Distance = 626.00
Iteration 12: Best Distance = 626.00
Iteration 13: Best Distance = 626.00
Iteration 14: Best Distance = 626.00
Iteration 15: Best Distance = 626.00
Iteration 16: Best Distance = 626.00
Iteration 17: Best Distance = 626.00
Iteration 18: Best Distance = 626.00
Iteration 19: Best Distance = 626.00
Iteration 20: Best Distance = 626.00

Best route found: [3, 1, 0, 7, 9, 14, 13, 11, 2, 10, 8, 12, 4, 5, 6]
Total distance: 626
PS C:\Users\91827>
```

3. Discussion

3.1 Fuzzy Logic Controller

The FLC implementation successfully demonstrates how fuzzy logic can model human reasoning in determining appropriate wash times based on subjective assessments of dirt and grease. Key advantages observed:

- Ability to handle imprecise inputs and produce reasonable outputs
- Intuitive rule-based approach that mimics human decision-making
- Smooth transitions between output categories

The triangular membership functions provided computational simplicity while still effectively capturing the linguistic variables. The Center of Gravity defuzzification method delivered consistent and stable results.

3.2 Genetic Algorithm for TSP

The GA implementation demonstrates the power of evolutionary algorithms in solving complex optimization problems. Key observations:

- Rapid convergence toward good solutions within 20 iterations
- Effectiveness of elitism in preserving good solutions
- Balance between exploration and exploitation through selection, crossover, and mutation

The algorithm successfully avoids invalid solutions through the rectification process, ensuring all constraints of the TSP are met.

4. Conclusion

Fuzzy Logic Controller: Successfully modeled wash time using Mamdani inference and CoG defuzzification, demonstrating how computational intelligence can effectively handle uncertainty in control systems.

Genetic Algorithm: Efficiently solved TSP with elitism, ensuring convergence to a near-optimal solution in a limited number of iterations, showcasing the power of evolutionary approaches for complex combinatorial optimization problems.

Both implementations highlight the effectiveness of soft computing techniques in addressing real-world problems where traditional computing approaches might struggle due to uncertainty, complexity, or the need for approximate solutions.

5. Team

- Shivang Negi: 22UCS200
- Anshit Mahajan: 22UCS024
- Pranav Shrivastava: 22UCS151
- Krishna Manchanda: 22UCS111

6. Submitted Files

- problem1_FLC.py: Fuzzy logic implementation
- problem2_TSP.py: Genetic algorithm for TSP
- SC_report.pdf: This document