Lab 4: Solving Combinatorial and Creative Problems with Genetic Algorithms

Soham Naukudkar
Roll No: 202351135

Siddhikesh Gavit
Roll No: 202351040

Shreyash Borkar
Roll No: 202351132

***Abstract*—This report details the application of a Genetic Algorithm (GA), a powerful heuristic search method, to solve two distinct and complex problems: a visual jigsaw puzzle and the procedural generation of Indian classical music. For the jigsaw puzzle, we formulate the problem as a state-space search to find the optimal permutation of scrambled image tiles. For the music generation challenge, we design a fitness function to guide the GA in composing a melody conforming to the grammatical rules of Raag Bhairav. Our results demonstrate that the GA successfully solves the jigsaw puzzle by minimizing a cost function based on edge dissimilarities. Furthermore, the algorithm proves capable of generating musically coherent and aesthetically pleasing melodies, showcasing the versatility of evolutionary computation in both optimization and creative domains.**

***Index Terms*—Genetic Algorithm, State-Space Search, Jigsaw Puzzle, Algorithmic Composition, Raag Bhairav, Fitness Function, Heuristic Search, Permutation Problem.**

I. INTRODUCTION

Search algorithms are fundamental to artificial intelligence, providing the means to navigate vast problem spaces to find optimal solutions. While traditional algorithms are effective, heuristic search methods like Genetic Algorithms offer robust solutions to complex optimization problems where the search space is too large for exhaustive methods.

This report explores two such problems: the Jigsaw Puzzle Problem and the Raag Bhairav Melody Generation challenge.

A. Objectives

Our primary objectives are:

1) **Jigsaw Puzzle Problem:** To formulate the puzzle as a state-space search problem and implement a Genetic Algorithm to find the correct assembly of a scrambled image.
2) **Raag Bhairav Melody Generation:** To design a fitness function that encodes musical rules and use a Genetic Algorithm to procedurally generate a melody in a specific North Indian Classical Raag.

Our goal is to demonstrate how a single algorithmic paradigm can be adapted to solve problems in both logical optimization and creative domains.

II. BACKGROUND: GENETIC ALGORITHMS

A Genetic Algorithm is a search heuristic inspired by Charles Darwin's theory of natural evolution. It operates on a **population** of candidate solutions (**individuals**) and iteratively evolves them toward better solutions.

The core components are:

- **Individual:** A representation (or "chromosome") of a single solution in the state space.
- **Fitness Function:** A function that evaluates an individual and assigns it a score based on how close it is to the optimal solution.
- **Selection:** The process of choosing parent individuals for breeding, where fitter individuals have a higher chance of being selected. Our implementation uses **Tournament Selection**.
- **Crossover:** The process of creating a new "child" solution by combining the genetic material of two parent solutions.
- **Mutation:** The process of introducing small, random changes to a child's chromosome to maintain genetic diversity.
- **Elitism:** A mechanism that guarantees the best individuals from one generation are passed to the next, ensuring the best solution is never lost.

III. JIGSAW PUZZLE SOLVER

A. Problem Formulation

The jigsaw puzzle is formulated as a state-space search problem:

- **State:** A permutation of the N puzzle pieces. Each state represents one possible assembly of the puzzle.
- **Initial State:** The scrambled configuration provided in the `scrambled_lena.mat` file.
- **Operators:** Crossover and Mutation operators are used to generate new states (arrangements) from existing ones.
- **Cost Function:** The fitness of a state is determined by a cost function that measures the dissimilarity across the edges of all adjacent tiles. The cost is the *sum of squared differences* of pixel values along each internal seam. A lower cost signifies a better fit.
- **Goal State:** The state (permutation) with the minimum possible cost.

B. Algorithm Implementation

We implemented a Genetic Algorithm in Python. Key details include a custom file reader for the `.mat` format, Tournament Selection, Ordered Crossover (OX1), Swap Mutation, and Elitism.

Algorithm 1: Genetic Algorithm Loop for Jigsaw Puzzle

Input: Number of Generations G, Population Size P
Output: The best solution found

```
population ← create_initial_population(P)
best_solution ← None
for i ← 1 to G do
    fitness_scores ← evaluate_population(population)
    if best_in(fitness_scores) > fitness(best_solution) then
        best_solution ← best_in(population)
    new_population ← select_elites(population)
    while |new_population| < P do
        p1, p2 ← tournament_selection(population)
        child ← crossover(p1, p2)
        child ← mutate(child)
        add child to new_population
    population ← new_population
return best_solution
```

C. Experimental Setup

The algorithm was applied to a 512x512 pixel image from `scrambled_lena.mat`, which was treated as a 4x4 grid of 16 tiles, each 128x128 pixels. The GA was configured with a population size of 100 and run for approximately 3000 generations to ensure convergence.

D. Results

The Genetic Algorithm successfully converged on a solution with a minimal cost score. The resulting image was a perfectly assembled puzzle, but it was rotated by 90 degrees, as shown in Fig. 1.

Fig. 1. The final, correctly assembled (but rotated) 4x4 Lena puzzle solved by the Genetic Algorithm.

E. Discussion

The rotational offset in the final result is an expected and insightful finding. It occurs because the cost function, which only measures seam differences, is **rotationally invariant**. It has no knowledge of "up" or "down" and correctly identifies any of the four possible rotations as an optimal solution.

IV. RAAG BHAIRAV MELODY GENERATION

A. Problem Formulation

For the challenge problem, the GA was adapted to compose music.

- **State (Individual):** A sequence of musical notes, where each note is represented by a (pitch, duration) tuple.
- **Gene Pool:** The pitches are constrained to the notes of the **Raag Bhairav** scale (Sa, komal Re, Ga, Ma, Pa, komal Dha, Ni).

B. Algorithm and Fitness Function

The melody was generated using a Genetic Algorithm with elitism. The algorithm evolves a population of melodies using Roulette Wheel selection to choose parents. New child melodies are created using a standard one-point crossover, and variation is introduced via a mutation operator that can alter either the pitch or duration of a note.

Algorithm 2: GA for Melody Generation

Input: Generations G, Population Size P, Elite Size k
Output: The best melody found

```
population ← create_random_melodies(P)
best_melody ← None
for i ← 1 to G do
    scored_pop ← evaluate_fitness(population)
    sort scored_pop descending by fitness
    if fitness(best_in(scored_pop)) > fitness(best_melody) then
        best_melody ← best_in(scored_pop)
    new_pop ← top k melodies from scored_pop
    while |new_pop| < P do
        p1, p2 ← roulette_selection(scored_pop)
        c1, c2 ← crossover(p1, p2)
        add mutate(c1) to new_pop
        add mutate(c2) to new_pop
    population ← new_pop
return best_melody
```

The core of this creative process is the multi-part fitness function, designed to encode the rules and aesthetics of the Raag:

- **Motif Reward:** Increases score if specific, characteristic phrases of Raag Bhairav are present.
- **Smoothness Penalty:** Penalizes large, dissonant melodic leaps between notes.

- **Cadence Reward:** Increases score if the melody ends on the tonic note (Sa).
- **Variety Reward:** Increases score for using a variety of rhythms and pitches to avoid monotony.

C. Results and Discussion

The Genetic Algorithm was run for 250 generations with a population of 120. It successfully evolved a random sequence of notes into a structured and musically pleasing melody. The final output, saved as `raag_bhairav_best.wav`, contains recognizable melodic phrases that conform to the basic grammar of the Raag, serving as a successful proof of concept.

V. CONCLUSIONS

This report successfully demonstrates the power and flexibility of Genetic Algorithms for solving complex state-space search problems.

1) For the **jigsaw puzzle**, the GA proved effective at navigating a vast permutation space, with the final result highlighting the importance of understanding the properties of the chosen cost function.
2) For the **melody generation** problem, the GA showcased its potential as a tool for computational creativity, where success depends on the careful design of the fitness function.

In both cases, the GA provided a robust framework for finding high-quality solutions in problems that are intractable for exhaustive search methods.

ACKNOWLEDGMENT

The authors would like to thank the course instructors and teaching assistants for their guidance and support throughout this lab assignment.

Python Implementation : GitHub

REFERENCES

[1] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 4th ed. Pearson, 2020.
[2] J. H. Holland, *Adaptation in Natural and Artificial Systems*. MIT Press, 1992.
[3] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by Simulated Annealing," *Science*, vol. 220, no. 4598, pp. 671-680, 1983.
[4] D. Sholomon, O. E. David, and N. S. Netanyahu, "A genetic algorithm for solving jigsaw puzzles," in *2013 IEEE Congress on Evolutionary Computation*, pp. 3131-3138, 2013.
[5] D. Cope, *The Algorithmic Composer*. A-R Editions, Inc., 2000.