Report for Image Puzzle Solution

Patch Size Specification

We initially experimented with a 4x2 patch size to simplify the puzzle. However, the algorithm solved this easily, even in fewer generations, indicating insufficient complexity. To make things more challenging and to test the robustness of our genetic algorithm, we increased the puzzle complexity to a 4x4 patch size. This adjustment meant the solution space expanded significantly, requiring more strategic piece rearrangements and a better balance between exploration and exploitation. The 4x4 configuration was selected deliberately to push the algorithm's limits and provide a rigorous test case, making the convergence process more engaging and insightful.

Explanation of Genetic Algorithm Components

Crossover Function
The crossover mechanism is crucial for combining genetic material from two parent solutions to create offspring that have the potential to be better than either parent. Here, a random crossover point is selected, and patches from each parent are split and then combined. This preserves partial structures from both parent solutions, which is beneficial for maintaining already optimized areas while exploring new configurations. By leveraging this approach, the algorithm effectively builds on the strengths of previous generations while experimenting with novel solutions.

Fitness Function
The fitness function serves as the performance metric, guiding the evolution of the population toward the optimal solution. It measures how closely an individual arrangement of patches resembles the original image by computing the difference between corresponding patches. To prevent overflow issues in the calculations, we normalized the differences. This normalization ensures that the fitness values remain stable and interpretable. The objective is straightforward: minimize the difference between the individual and the original image, with smaller discrepancies yielding higher fitness scores. It's worth noting that using a well-designed fitness function helps the algorithm efficiently navigate the search space and converge on optimal solutions.

Mutation Function
To prevent premature convergence and stagnation, the mutation function introduces diversity into the population. By randomly swapping two patches within an individual with a specified mutation probability, we inject randomness and promote exploration of the solution space. What's interesting is that the mutation rate is adaptive: it starts at an initial level but increases over time. This gradual increase allows the algorithm to become more exploratory in later generations, a strategy that mitigates the risk of getting stuck in local optima. As a software engineering student, I appreciated how this dynamic approach mirrors real-world problem-solving, where flexibility can be as crucial as precision.

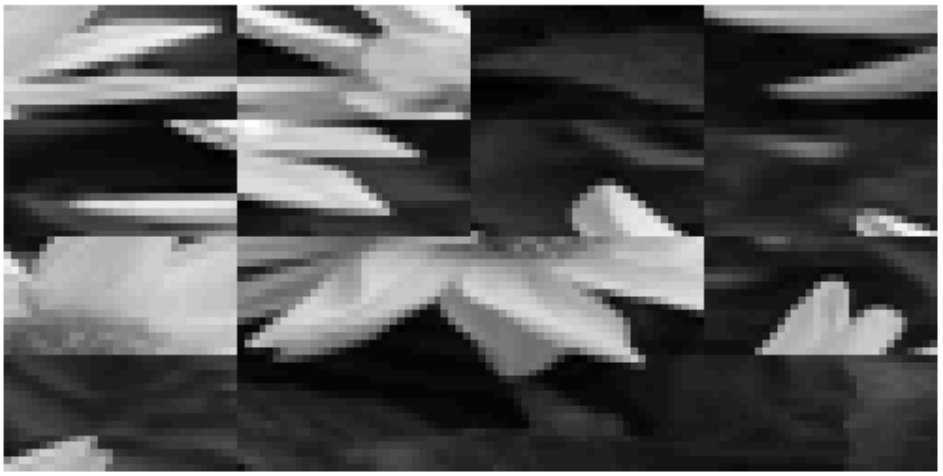
Selection and Elitism
Our algorithm employs elitism to ensure that the top-performing individuals are preserved across generations. This guarantees that the best solutions are not lost, which is essential for incremental improvement. The rest of the new population is generated through crossover and mutation, striking a balance between preserving high-quality solutions and fostering genetic diversity. This dual approach, blending survival of the fittest with the creation of new variants, aligns well with evolutionary principles and gives the algorithm a competitive edge.

Simulated Annealing
Simulated Annealing acts as a complementary refinement step, applied specifically to elite individuals. This technique introduces a temperature-based probability that allows occasional acceptance of less optimal moves. By doing so, the algorithm gains the ability to escape from local minima and continue searching for a global optimum. The cooling schedule gradually lowers the temperature, making the system more conservative as it runs. In my tests, there were moments when the algorithm did hit local minima. However, thanks to Simulated Annealing, it often recovered and found the optimal solution. I found this hybrid approach particularly fascinating, as it simulates both disciplined improvement and the flexibility to take risks—concepts that resonate well with real-life problem-solving scenarios.

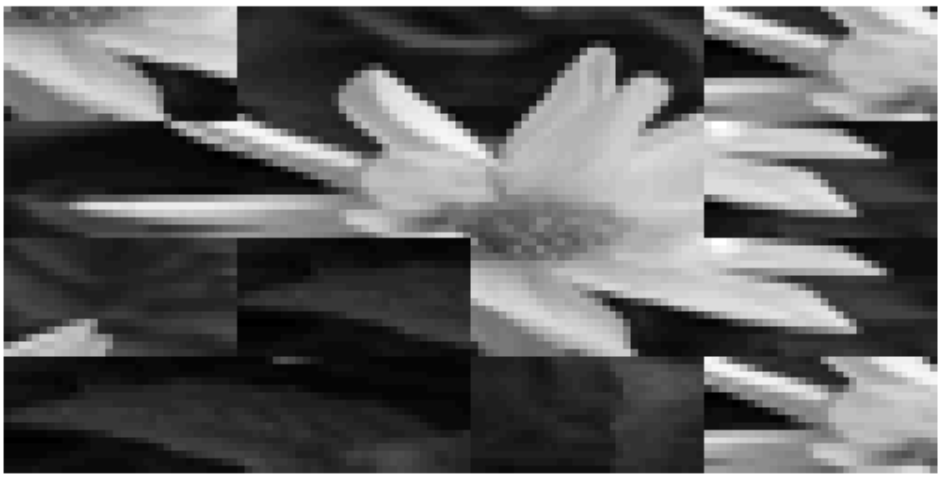
Additional Observations
Throughout the project, one of the most intriguing aspects was observing how the algorithm navigated the balance between exploration and exploitation. While there were some runs where it got momentarily stuck in suboptimal states, the algorithm showed a strong overall tendency to escape and continue improving. As a software engineering student, this highlighted the importance of designing algorithms with mechanisms to handle complex, non-linear solution spaces. Ultimately, this experience underscored the value of adaptive strategies in algorithm design and how they can be leveraged to solve real-world puzzles efficiently.

Images of Best Individuals
For every generation, the best solution is visualized and saved as an image. This allows us to track the progress of the algorithm and observe how the solution evolves over time. These images illustrate the improvement in the arrangement of patches across generations.

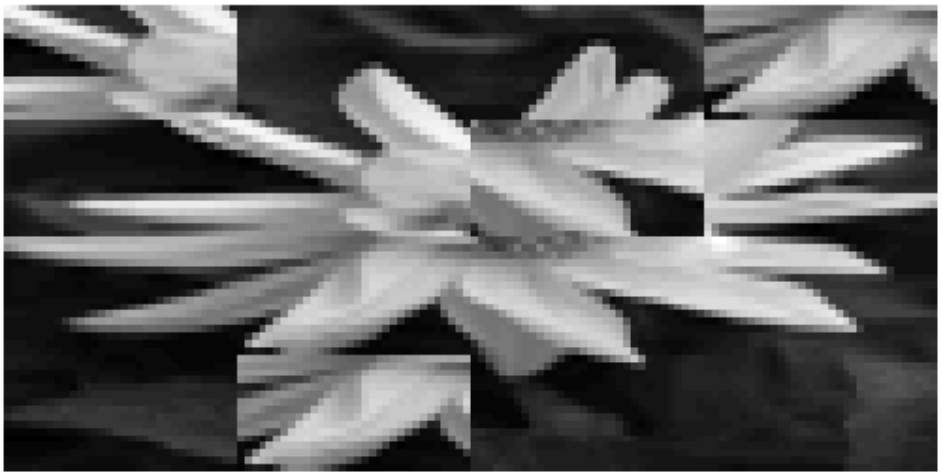
Generation 0



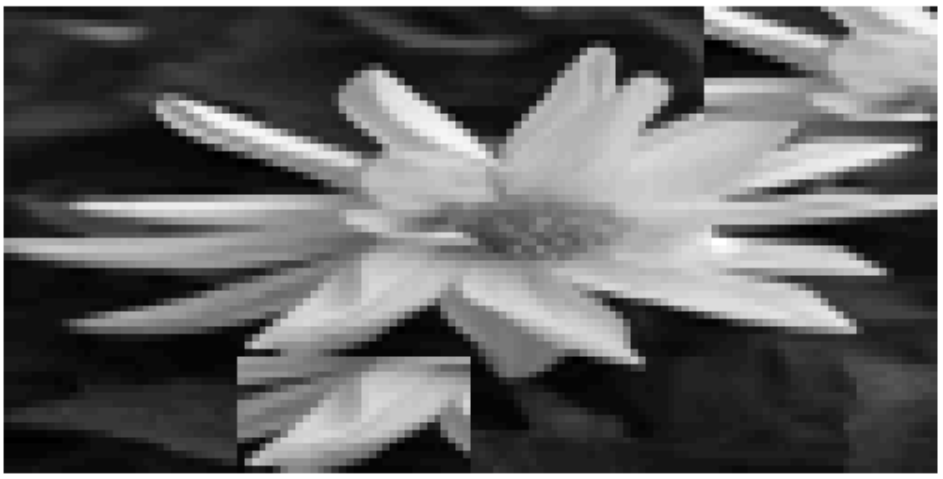
Generation 1



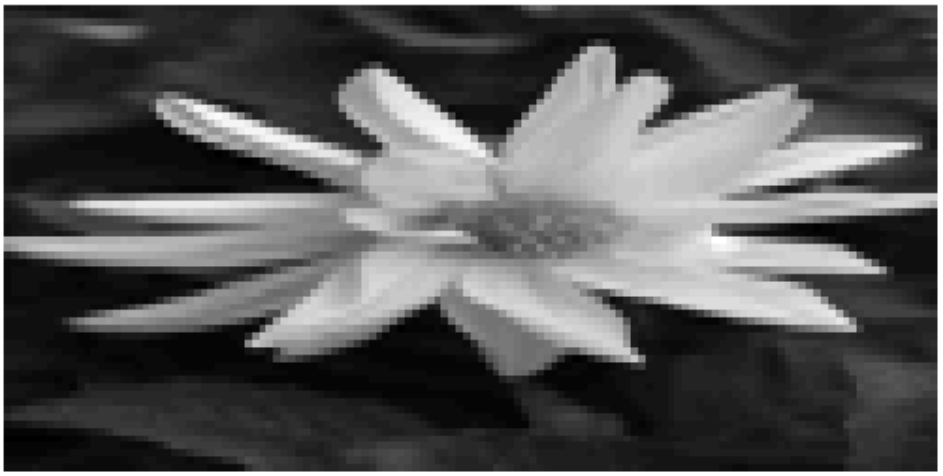
Generation 2



Generation 3



Generation 4



Generation final