METU EE449 Homework 3: Evolutionary Algorithms

1 Introduction

This report details the findings of the experiments completed as part of Homework 3 for METU EE 449 Computational Intelligence. The goal of the assignment was to create an evolutionary algorithm in Python, utilizing OpenCV to produce an image of filled triangles that resemble a given RGB source image (Figure 1). The fitness score was calculated using the Structural Similarity Index Measure (SSIM). The experiments focused on analyzing seven hyperparameters and three new modifications were enforced in an attempt to improve convergence. In this document, I cover the effectiveness of the hyperparameters and their optimal values, relevant plots and images, as well as the evaluation of my proposed changes.

Figure 1: Source image (painting.png)

2 Hyperparameter Analysis

For each hyperparameter, experiments used the values in Table 1 of the assignment, with default values (num_inds=20, num_genes=50, tm_size=5, frac_elites=0.2, frac_parents=0.6, mutation_prob=0.2, mutation_type=guided) for non-tested parameters. The algorithm ran for 10,000 generations, and the best SSIM fitness scores are reported. Results are described within the report.

Parameter	**Values**
<num inds>	5, 10, **20**, 50, 75
<num genes>	10, 25, **50**, 100, 150
<tm size>	2, **5**, 10, 20
<frac elites>	0.05, **0.2**, 0.4
<frac parents>	0.2, 0.4, **0.6**, 0.8
<mutation prob>	0.1, **0.2**, 0.5, 0.8
<mutation type>	**guided**, unguided

Table 1: parameters to be tested

2.1 Number of Individuals (num_inds)

The number of individuals determines the population variance. A smaller population size (5) constricts exploration leading to faster convergence, captured at a low SSIM of 0.62 displayed in Figure 2. Noticeable exploration and steady improvement in fitness score were observed in larger populations (10-75), resulting in higher SSIM. The peak SSIM of 0.825 is observed at 75 individuals, marking stronger diversity but at an expense of computational cost. Figures from 1,000 to 10,000 iterations (Figure 3) depict improved approximations when the number of individuals is 75. The impact of varying the number of individuals on SSIM performance is summarized in Table 2, showing a steady improvement as the population size increases.

Parameter	**5**	**10**	**20**	**50**	**75**
SSIM	0.62	0.765	0.79	0.805	0.825

Table 2: SSIM values for different numbers of individuals

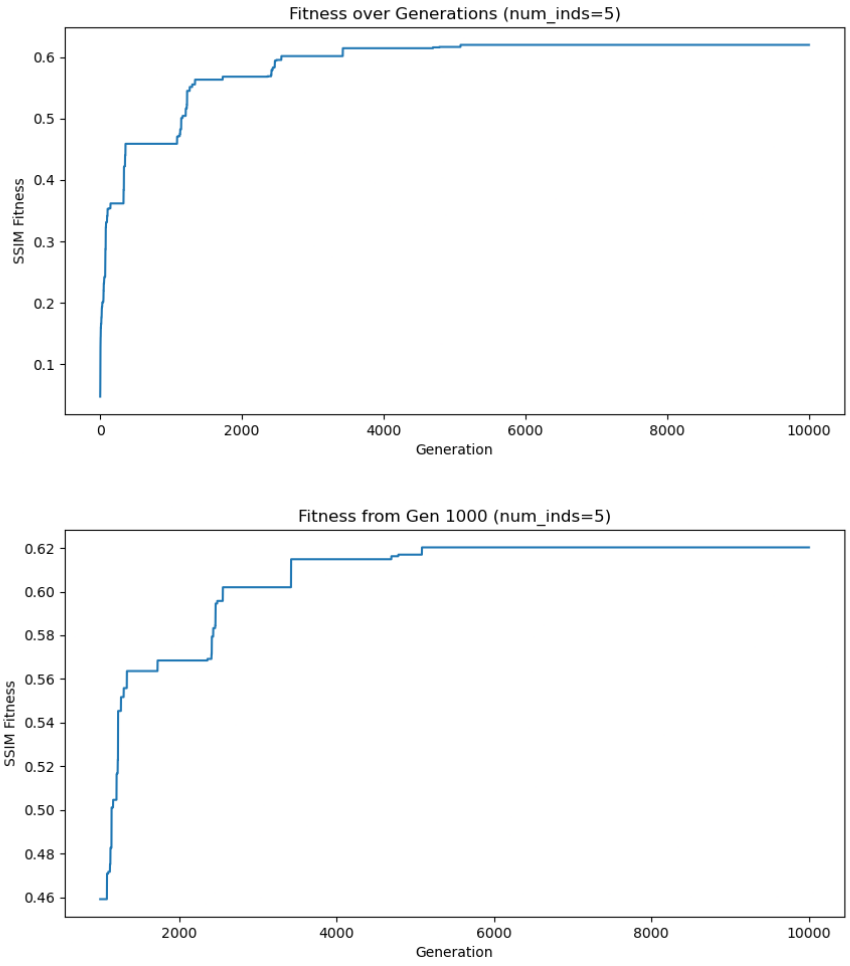


Figure 2: SSIM for num_inds = 5

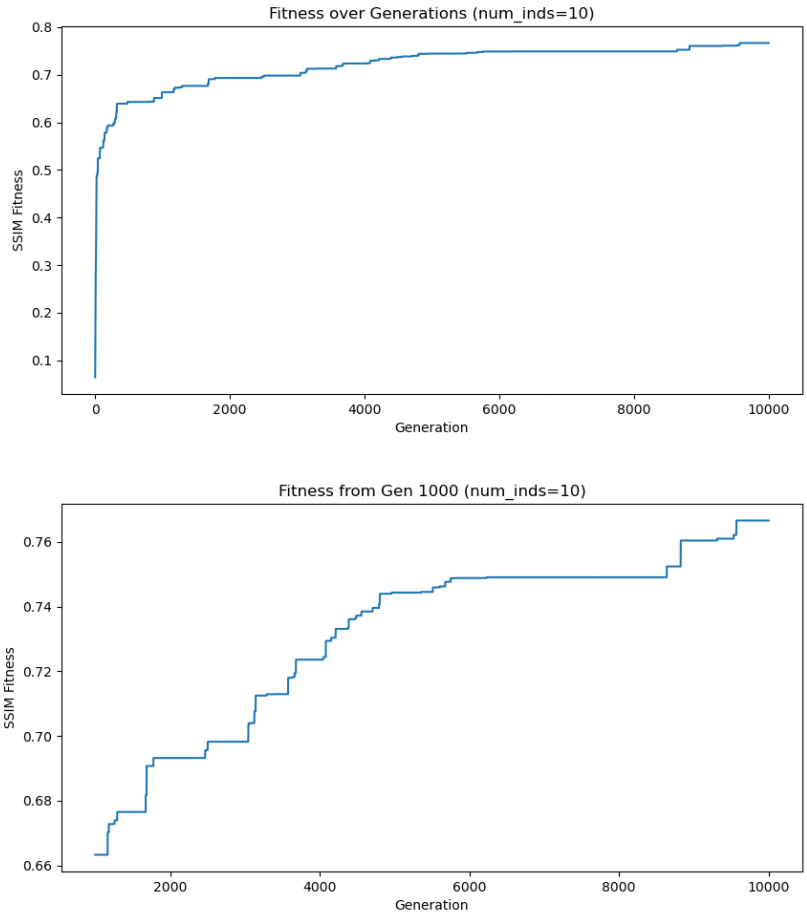


Figure 3: SSIM for num_inds = 75

2.2 Number of Genes (num_genes)

The complexity of the generated image is based on the number (of triangles) genes. Too few triangles (10) limit detail, causing a plateau at 0.742 SSIM (Figure 4). The peak SSIM (0.83) at 25 genes suggests an optimal balance, as shown in Figure 5. Higher values (100, 150) lead to overfitting or redundant triangles, reducing SSIM (0.635 at 150), with fitness plateauing (Figure 6). The impact of varying the number of individuals on SSIM performance is summarized in Table 3.

Number of Genes	**10**	**25**	**50**	**100**	**150**
SSIM	0.742	0.83	0.79	0.725	0.635

Table 3: SSIM values for different numbers of genes

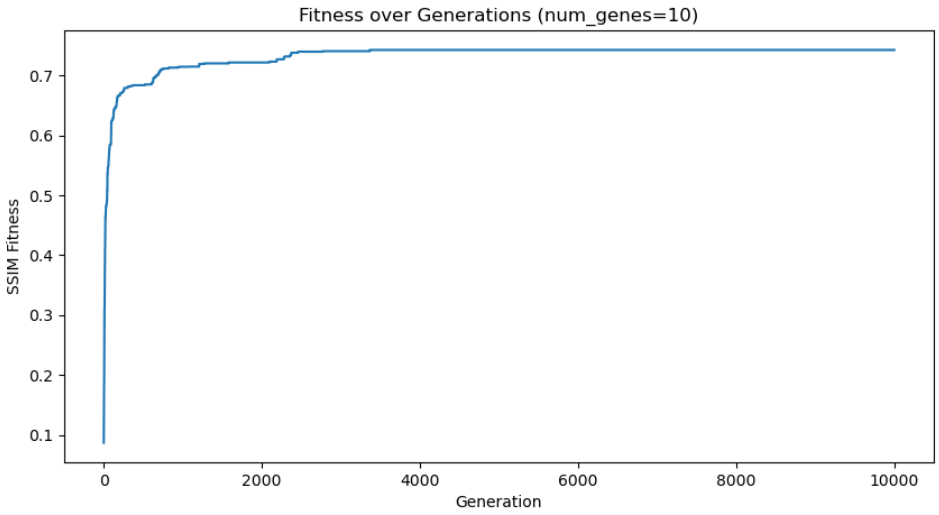


Figure 4: SSIM for num_genes = 10

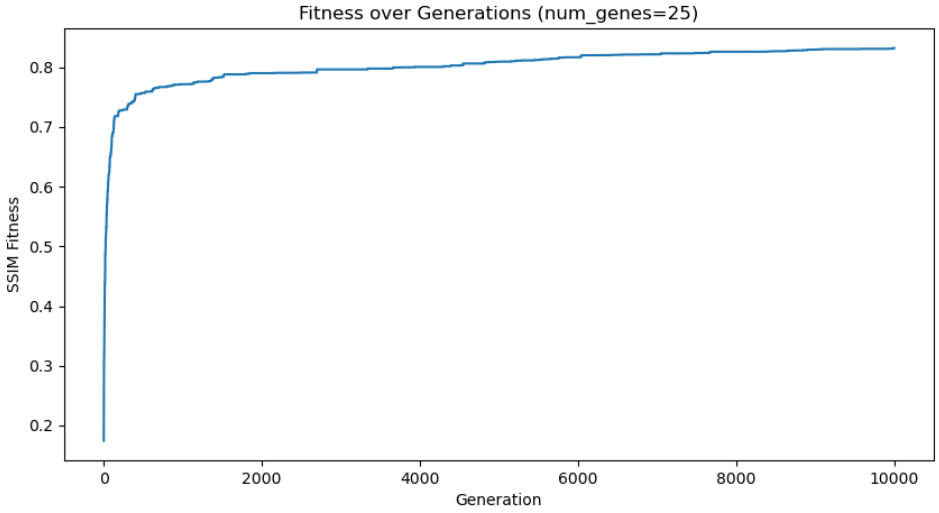


Figure 5: SSIM for num_genes = 25

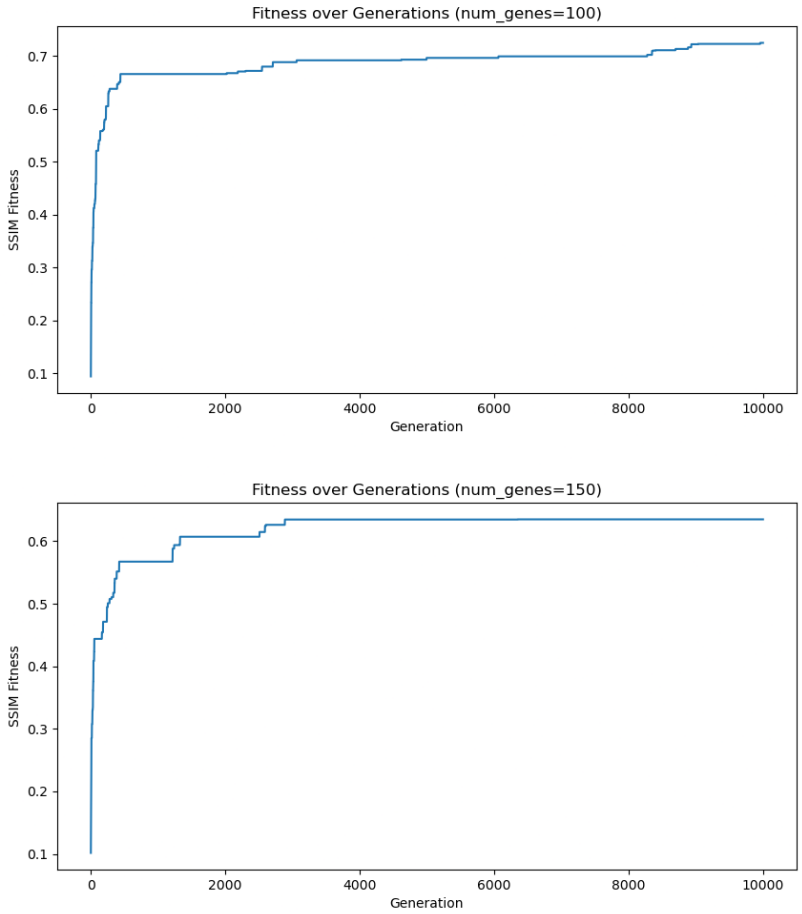


Figure 6: SSIM for num_genes = 100 and num_genes = 150

2.3 Tournament Size (tm_size)

Effect: The selection pressure is influenced by the tournament size. Smaller sizes (2, 5) tend to yield higher SSIM (0.79, 0.785) alongside steady fitness growth (Figure 7) due to optimal balance between exploration and exploitation. Larger sizes (10, 20) impose more pressure which slows convergence, noted in results, and reduces SSIM (0.74 at 20), as seen in Figure 8. The best approximations were recorded for smaller tournament sizes (Figure 9). The impact of varying the tournament size on SSIM performance is summarized in Table 3.

Tournament Size	**2**	**5**	**10**	**20**
SSIM	0.79	0.785	0.78	0.74

Table 4: SSIM values for different tm_size

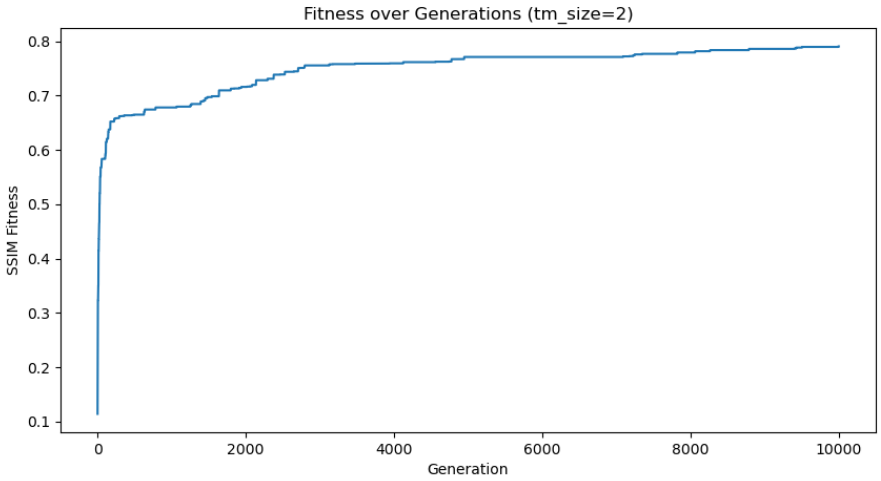


Figure 7: SSIM for tm_size = 2

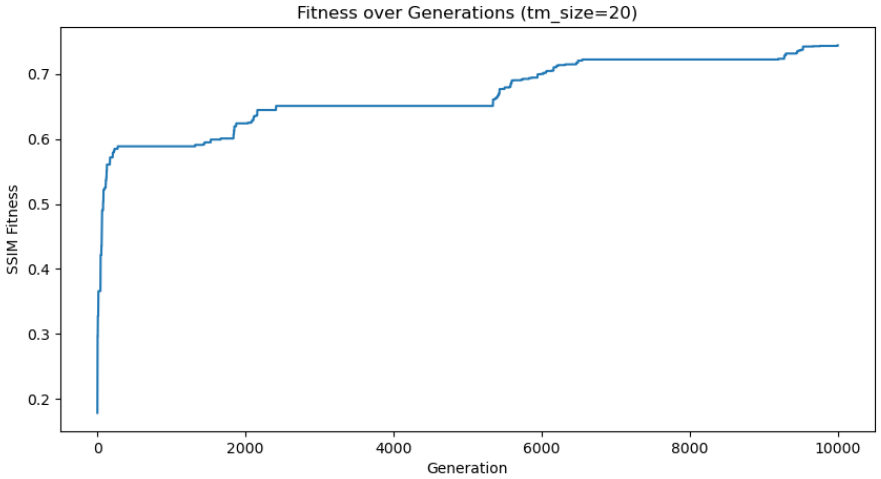


Figure 8: SSIM for tm_size = 20

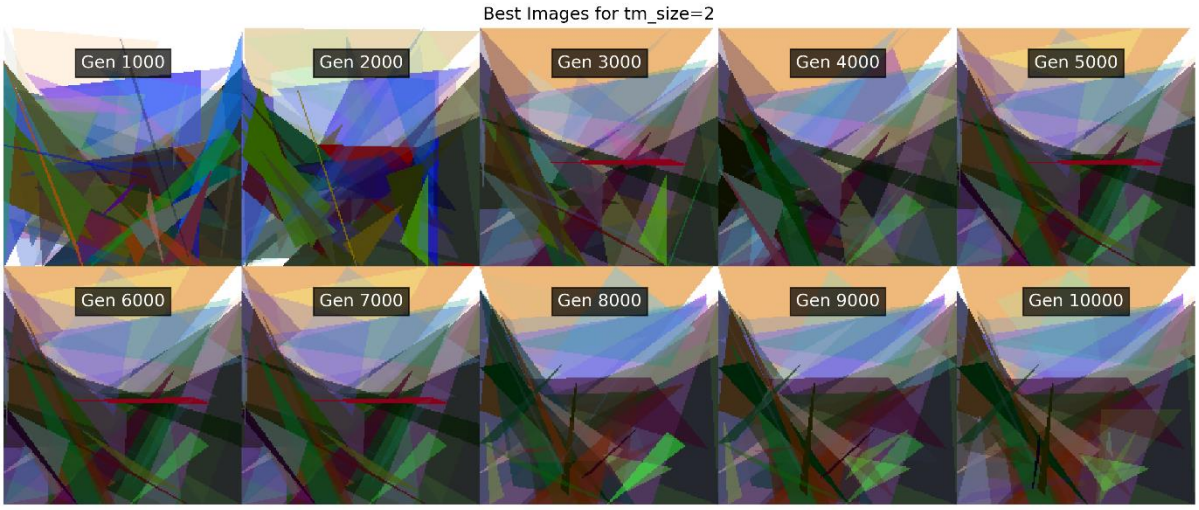


Figure 9: Best images per 1000 generation for tm_size = 2

2.4 Fraction of Elites (frac_elites)

The elite fraction retaining roughly the best individuals results in them being top capped. A low fraction (0.05) is capped for retention of useful solutions causing a plateau at 0.687 SSIM (fitnessfullfracelites0.05.png) stricter (0.2, 0.4) caps where good solutions are retained leads to a sustained increase in SSIM (0.8). Images (bestimagegen__fracelites_0.2.png) demonstrate a consistent improvement. No significant difference exists between 0.2 and 0.4, hence the preferable is set to 0.2. Less computational time is needed with this default setting. The impact of varying the frac_elites values on SSIM performance is summarized in Table 5.

Fraction of Elites	0.05	0.2	0.4
SSIM	0.687	0.8	0.8

Table 5: SSIM values for different frac_elites

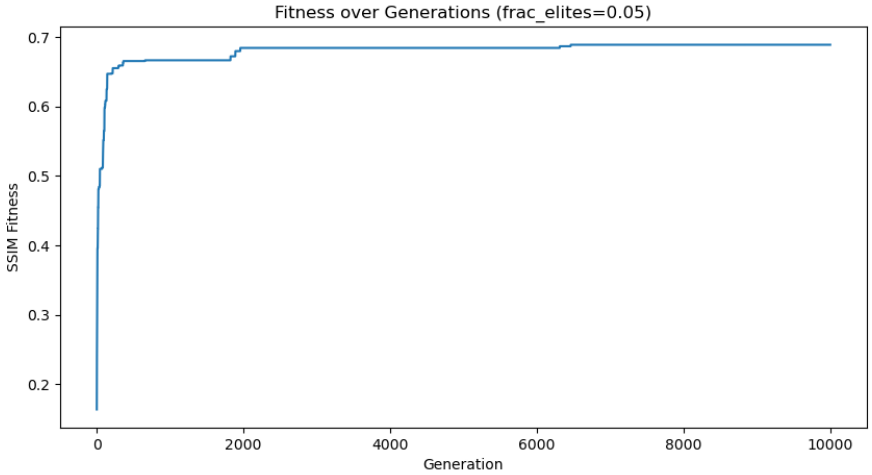


Figure 10: SSIM for frac_elites = 0.05

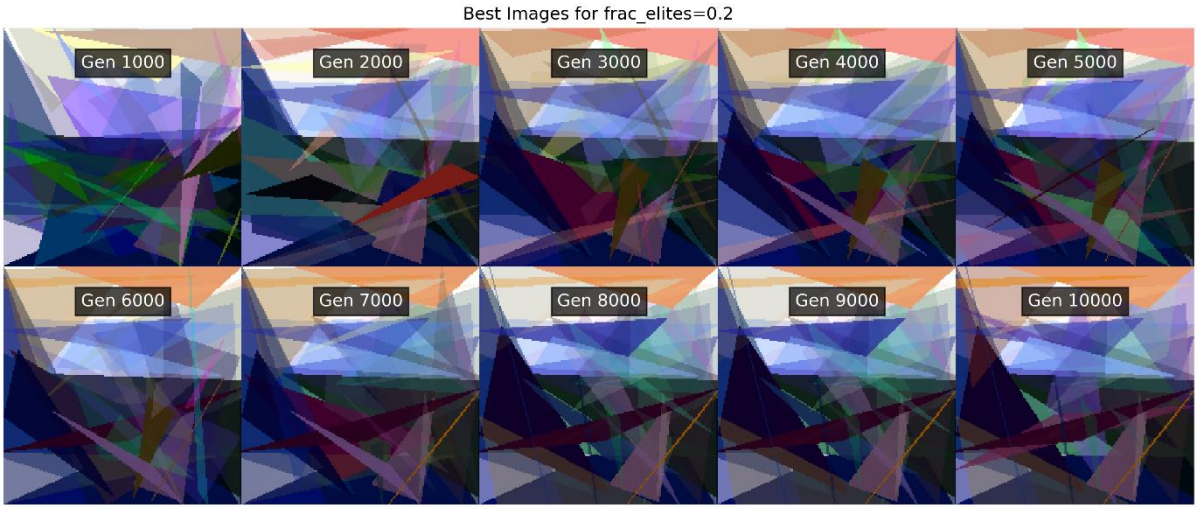


Figure 11: Best images per 1000 generation for frac_elites = 0.2

2.5 Fraction of Parents (frac_parents)

Defining the fraction of crossover parents determines the SSIM value produced by the model. Mid-range fractions (0.2, 0.4, 0.6) provide consistent improvements over generations. However, these mid-fraction (0.2, 0.4, and 0.6) yield lower SSIM (0.755–0.79). With a high fraction (0.8), the model yields the best SSIM (0.795). This is likely due to more parents adding genetic diversity to the population (Figure 12). The impact of varying the frac_parents values on SSIM performance is summarized in Table 6.

Fraction of Parents	0.2	0.4	0.6	0.8
SSIM	0.79	0.777	0.755	0.795

Table 6: SSIM values for different frac_parents

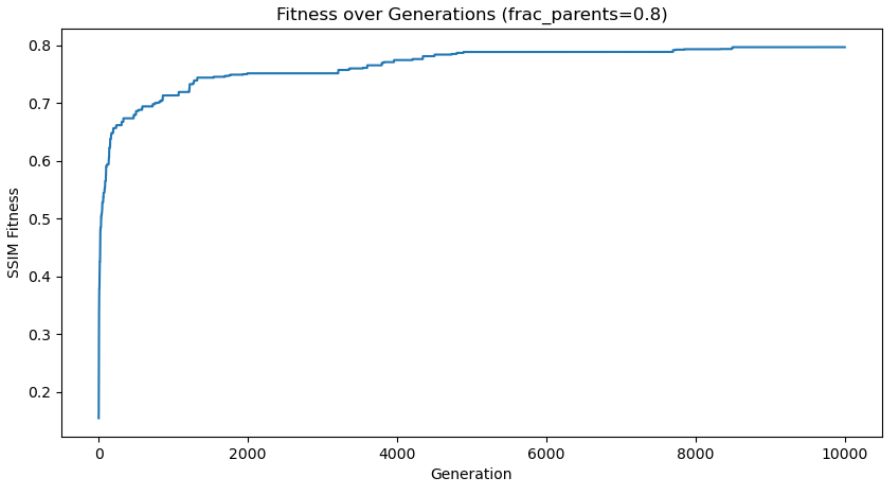


Figure 12: SSIM for frac_parents = 0.8

2.6 Mutation Probability (mutation_prob)

Effect: Mutation probabilities govern the SSIM value produced by the model. Low probabilities (0.1, 0.2) allow steady fitness improvements, with 0.1 achieving the highest SSIM (0.84) (Figure 13). High probabilities (0.5, 0.8) add too much chaos, causing the disruption of good solutions and plateaus at lower SSIM (seen value of 0.635 at 0.8) as demonstrated in (Figure 14). The impact of varying the mutation probability values on SSIM performance is summarized in Table 7.

Mutation Probability	**0.1**	**0.2**	**0.5**	**0.8**
SSIM	0.84	0.77	0.72	0.635

Table 7: SSIM values for different mutation_prob

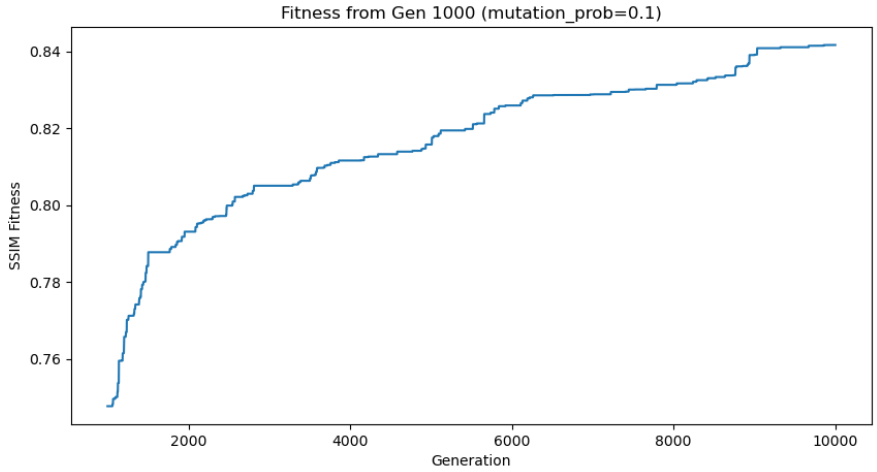


Figure 13: SSIM for mutation_prob = 0.1

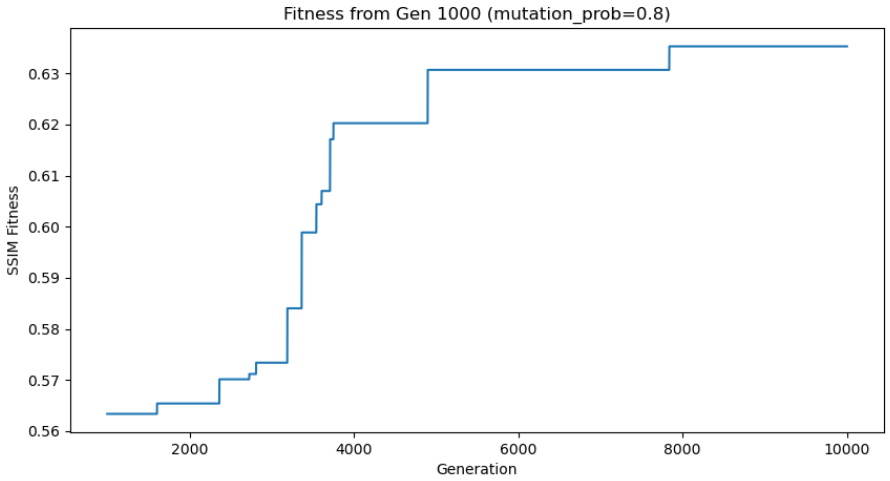


Figure 14: SSIM for mutation_prob = 0.8

2.7 Mutation Type (mutation_type)

The guided mutation allows for local adjustment of gene values while maintaining favorable characteristics, resulting in a high SSIM of appearance model (0.8) with consistent growth (Figure 15). With disruption of convergence and randomization of genes, unguided mutation shows a low SSIM score of 0.55 (Figure 16). Guided images show significantly better approximations than unguided images (Figure 17).

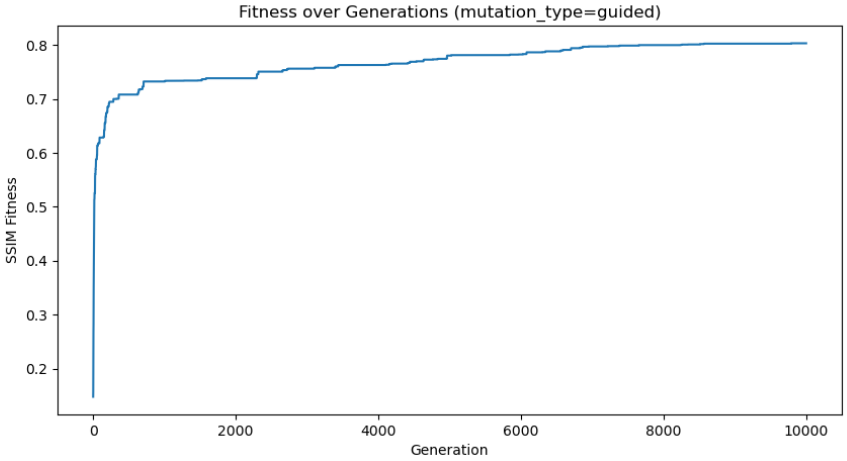


Figure 15: SSIM for mutation_type = guided

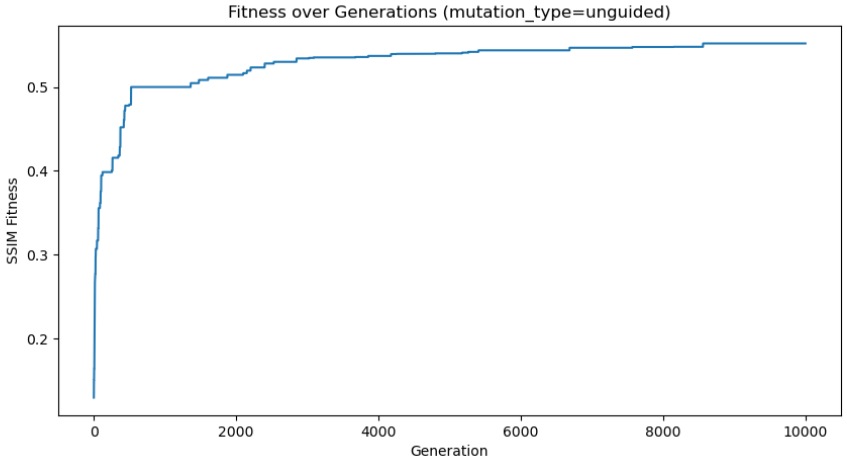


Figure 16: SSIM for mutation_type = unguided

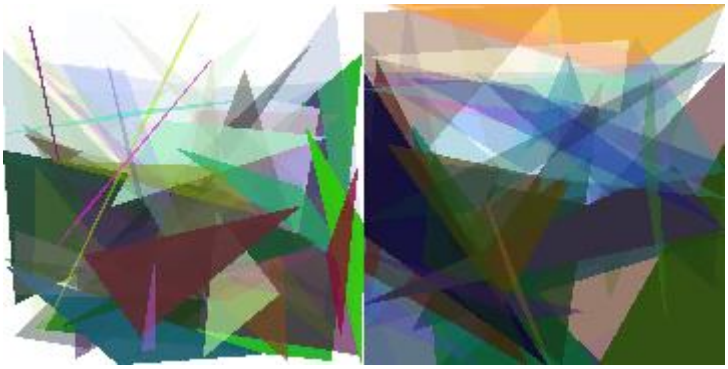


Figure 17: unguided (left), guided (right)

3 Proposed Improvements

Three modifications were implemented to improve convergence: adaptive mutation, dynamic triangles, and fitness sharing. Experiments used default parameters, with results compared to a baseline with default hyperparameters (Figure 18). Output images for baseline experiment per 1000 generations can be seen in Figure 19.

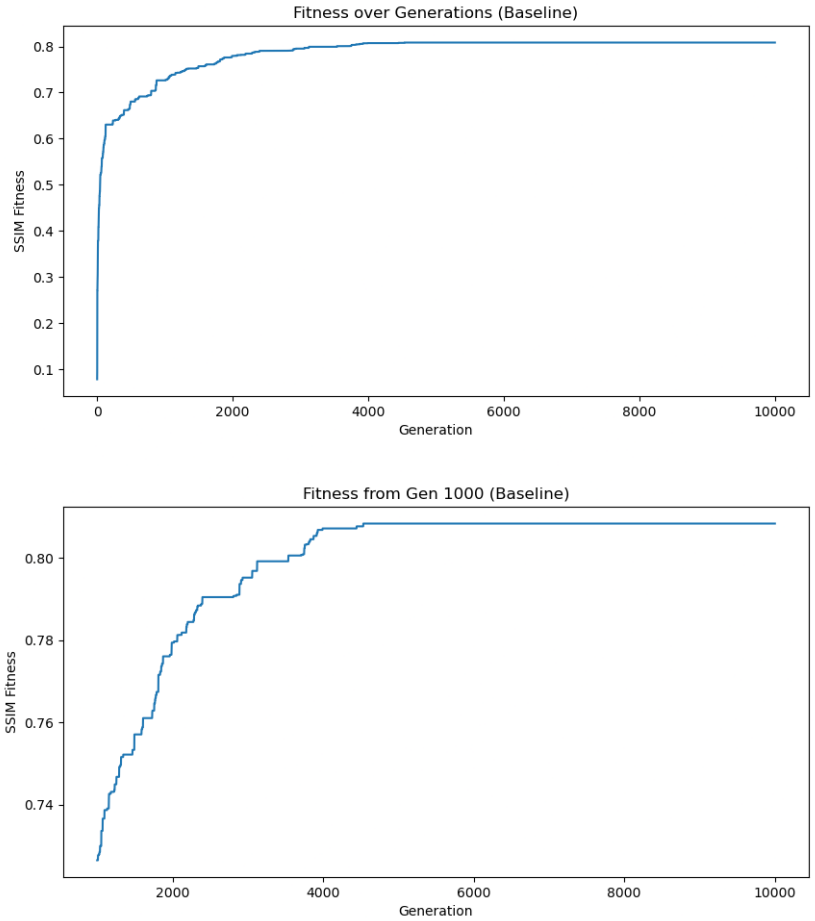


Figure 18: SSIM for baseline experiment

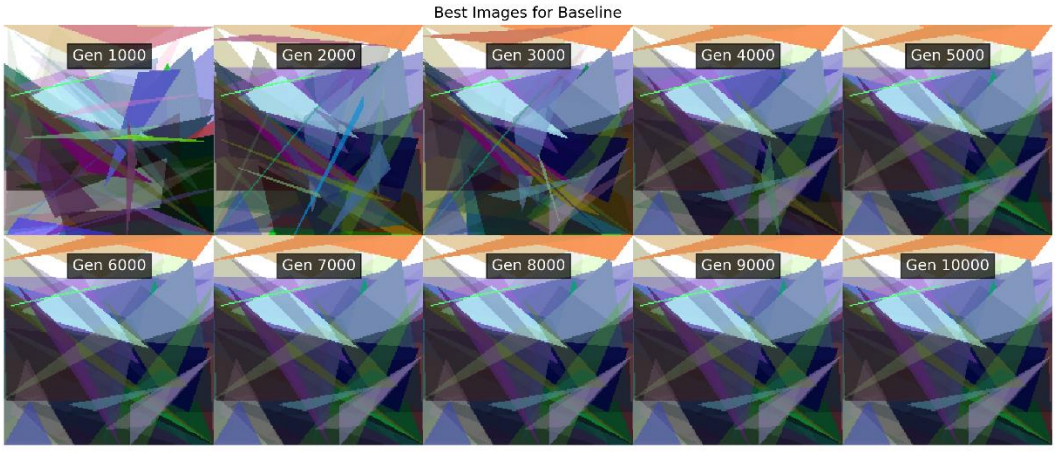


Figure 19: Output images for baseline experiment

3.1 Adaptive Mutation

Adaptive mutation changes the mutation rate due to a stagnation level in fitness. (somewhat raised to 0.8 max, lowered to 0.1 min). In the case of SSIM (0.82) and the baseline, adaptive mutation seems to provide a small advantage over the heuristic baseline (0.805). As observed in Figure 20, the convergence appears to be slower, indicating the adjustment mechanism may restrict exploration. Output images per 1000 generations can be seen in Figure 21.

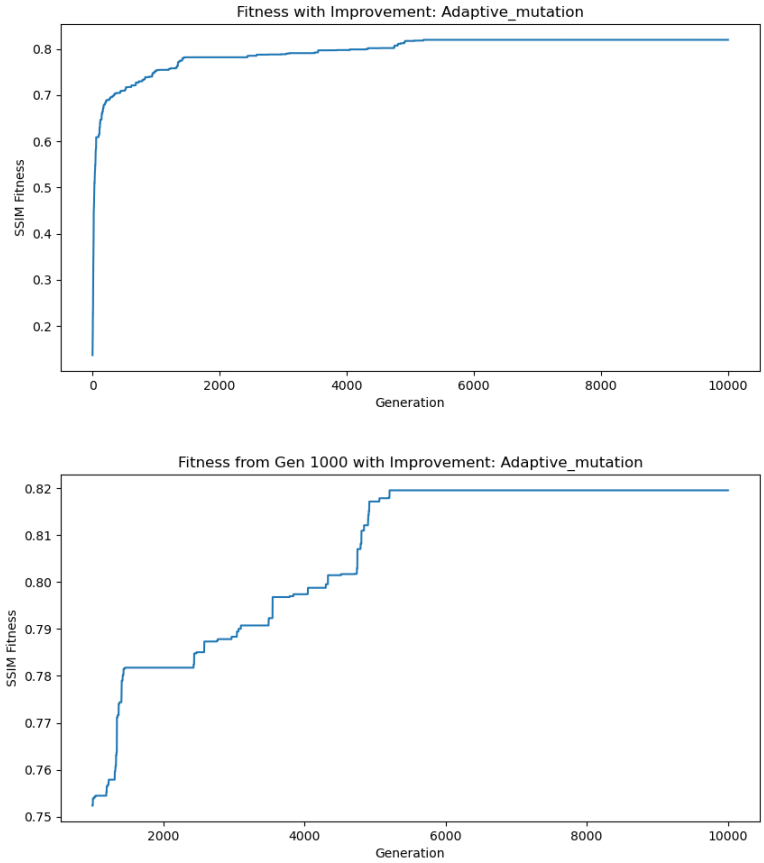


Figure 20: SSIM for adaptive mutation

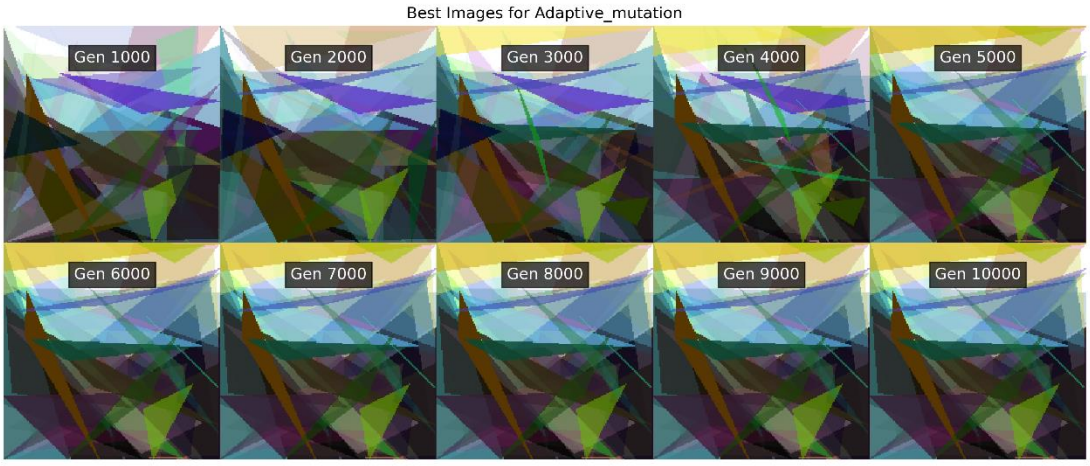


Figure 21: Output images for adaptive mutation

3.2 Dynamic Triangles

Dynamic triangles increase the number of genes linearly from 10 to 50 over generations. The estimated peak SSIM (0.84) at generation 4,500 (Figure 22) indicates that adding detail in the form of triangle early on significantly improves detail quite rapidly; however, the drop to 0.82 by 7000 indicates that these excess triangles might be causing disruption or overfitting. Improves early convergence but may need adjustment to sustain late degradation. Output images per 1000 generations can be seen in Figure 23.

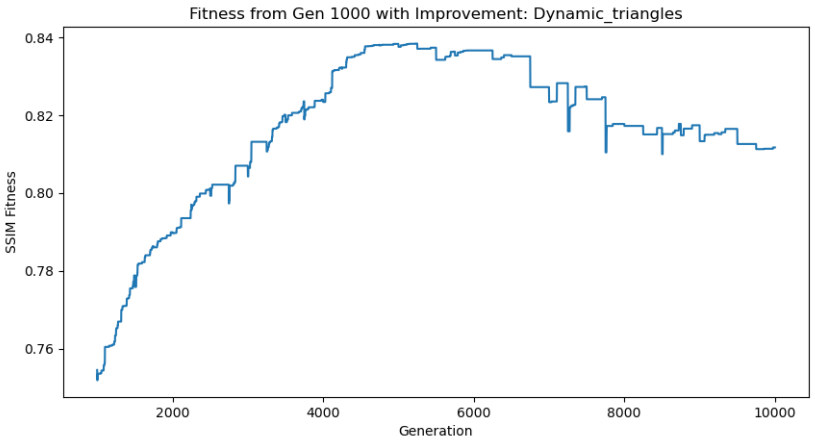


Figure 22: SSIM for dynamic triangles

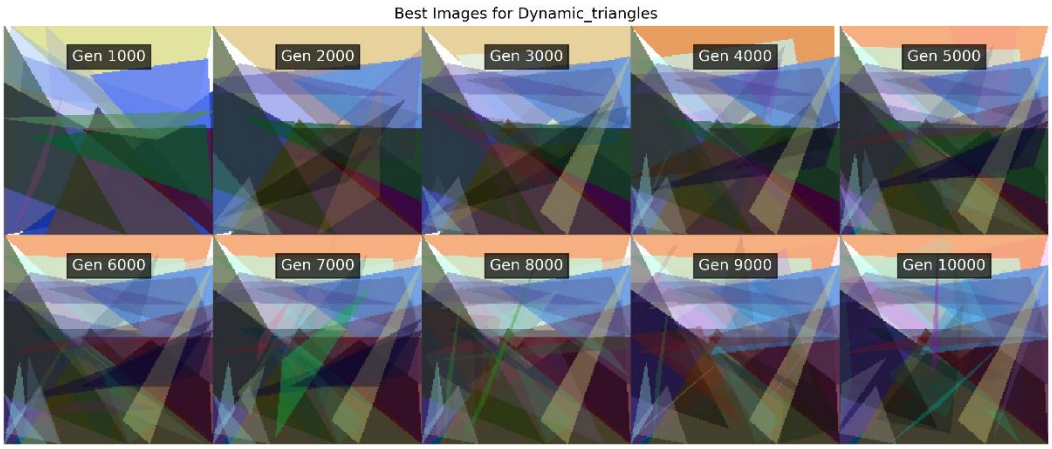


Figure 23: Output images for dynamic triangles

3.3 Fitness Sharing

Fitness sharing reduces fitness based on population similarity to promote diversity. Based on the very low value of SSIM (0.6) and oscillations (Figure 24), it can be said that including too much diversity prevents focus on optimal solutions, disrupting convergence. Images (Figure 25) clearly show the lack of resemblance for the approximations which proves the method was ineffective for this particular task. Perhaps the sigma parameter (100.0) requires some modification to strike the appropriate balance between diversity and convergence.

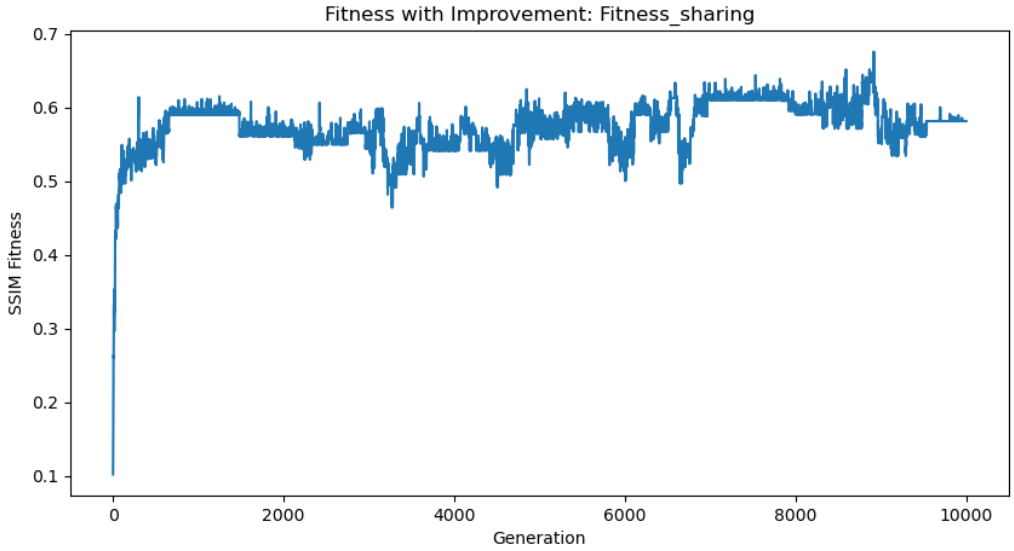


Figure 24: SSIM for fitness sharing

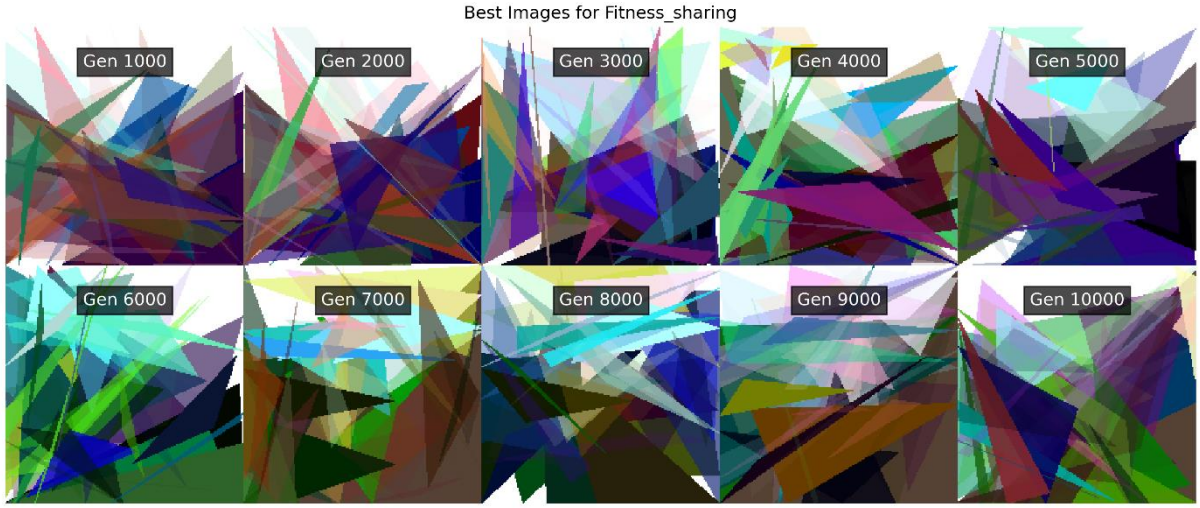


Figure 25: Best images for fitness sharing

4 Conclusion

The experiments prove that the performance of the evolutionary algorithm is sensitive to hyperparametric configurations. The most optimal configuration was set as follows: numinds=75, numgenes=25, tmsize=2, fracelites=0.2, fracparents=0.8, mutationprob=0.1, guided being the type of mutation used, which provides the highest SSIM (0.84 with mutation_prob=0.1). Adapting mutation and dynamic triangles improves SSIM slightly over the baseline measure (0.805) with both achieving 0.82, while dynamic triangles showcased quicker early convergence. SSIM declines considerably when fitness sharing is included, reaching a dismal 0.6. Future work may look to refine dynamic triangle limits alongside adaptive mutation schedules to sustain improvements achieved through these adjustments.

Appendix

A. CODE

```
import cv2
import numpy as np
from copy import deepcopy
import matplotlib.pyplot as plt
import os
from scipy.spatial.distance import euclidean

# Gene class to represent a triangle
class Gene:
    def __init__(self, width, height):
        if width <= 0 or height <= 0:
            raise ValueError(f"Invalid dimensions: width={width}, height={height}")
        # Initialize vertices and color
        self.vertices = [(np.random.randint(0, width), np.random.randint(0, height)) for _ in range(3)]
        self.color = [int(np.random.randint(0, 256)) for _ in range(3)] + [np.random.uniform(0, 1)]
        self.area = self.calculate_area()

        # Ensure triangle is within image bounds
        while not self.is_within_bounds(width, height):
            self.vertices = [(np.random.randint(0, width), np.random.randint(0, height)) for _ in range(3)]
            self.color = [int(np.random.randint(0, 256)) for _ in range(3)] + [np.random.uniform(0, 1)]
            self.area = self.calculate_area()

    def calculate_area(self):
        # Calculate triangle area using the shoelace formula
        x1, y1 = self.vertices[0]
        x2, y2 = self.vertices[1]
        x3, y3 = self.vertices[2]
        return 0.5 * abs(x1 * (y2 - y3) + x2 * (y3 - y1) + x3 * (y1 - y2))

    def is_within_bounds(self, width, height):
        # Check if at least one vertex is within image bounds
        for x, y in self.vertices:
```

```
            if 0 <= x < width and 0 <= y < height:
                return True
        return False

    def mutate(self, width, height, mutation_type, mutation_prob):
        if np.random.random() < mutation_prob:
            if mutation_type == 'unguided':
                self.vertices = [(np.random.randint(0, width), np.random.randint(0, height)) for _ in range(3)]
                self.color = [int(np.random.randint(0, 256)) for _ in range(3)] + [np.random.uniform(0, 1)]
            elif mutation_type == 'guided':
                for i in range(3):
                    self.vertices[i] = (
                        np.clip(self.vertices[i][0] + np.random.randint(-width//4, width//4), 0, width-1),
                        np.clip(self.vertices[i][1] + np.random.randint(-height//4, height//4), 0, height-1)
                    )
                self.color = [
                    int(np.clip(self.color[0] + np.random.randint(-64, 65), 0, 255)),
                    int(np.clip(self.color[1] + np.random.randint(-64, 65), 0, 255)),
                    int(np.clip(self.color[2] + np.random.randint(-64, 65), 0, 255)),
                    np.clip(self.color[3] + np.random.uniform(-0.25, 0.25), 0, 1)
                ]
            self.area = self.calculate_area()

# Individual class to represent a chromosome
class Individual:
    def __init__(self, num_genes, width, height):
        self.genes = [Gene(width, height) for _ in range(num_genes)]
        self.genes.sort(key=lambda x: x.area, reverse=True)
        self.fitness = None
        self.raw_fitness = None
        self.shared_fitness = None

    def draw_image(self, shape):
        # Initialize white image
        image = np.ones(shape, dtype=np.uint8) * 255
        for gene in self.genes:
            overlay = image.copy()
```

```
            pts = np.array(gene.vertices, np.int32).reshape((-1, 1, 2))
            color = tuple(int(c) for c in gene.color[:3])
            try:
                cv2.fillPoly(overlay, [pts], color)
                image = cv2.addWeighted(overlay, gene.color[3], image, 1 - gene.color[3], 0)
            except Exception as e:
                print(f"Error in cv2.fillPoly: {e}")
                print(f"Vertices: {pts}, Color: {color}")
                raise
        return image

# Population class
class Population:
    def __init__(self, num_inds, num_genes, width, height):
        self.individuals = [Individual(num_genes, width, height) for _ in range(num_inds)]
        self.current_num_genes = num_genes
        self.best_fitness = 0
        self.stagnation_count = 0
        self.mutation_prob = 0.2

    def evaluate(self, source_image, use_fitness_sharing=False):
        for ind in self.individuals:
            generated_image = ind.draw_image(source_image.shape)
            ind.raw_fitness = calculate_ssim(source_image, generated_image)
            ind.fitness = ind.raw_fitness
        if use_fitness_sharing:
            self.apply_fitness_sharing()

    def apply_fitness_sharing(self, sigma=100.0):
        for i, ind1 in enumerate(self.individuals):
            sharing_sum = 0
            for j, ind2 in enumerate(self.individuals):
                if i != j:
                    distance = self.compute_individual_distance(ind1, ind2)
                    sharing_sum += np.exp(-distance / sigma)
            ind1.shared_fitness = ind1.raw_fitness / (1 + sharing_sum)
            ind1.fitness = ind1.shared_fitness
```

```
    def compute_individual_distance(self, ind1, ind2):
        total_distance = 0
        min_genes = min(len(ind1.genes), len(ind2.genes))
        for g1, g2 in zip(ind1.genes[:min_genes], ind2.genes[:min_genes]):
            vertex_distance = sum((v1[0] - v2[0])**2 + (v1[1] - v2[1])**2
                                  for v1, v2 in zip(g1.vertices, g2.vertices))
            color_distance = sum((c1 - c2)**2 for c1, c2 in zip(g1.color, g2.color))
            total_distance += np.sqrt(vertex_distance + color_distance)
        return total_distance / min_genes if min_genes > 0 else 0

    def adjust_triangle_count(self, gen, max_gen, max_genes):
        progress = gen / max_gen
        target_genes = int(10 + (max_genes - 10) * progress)
        if target_genes > self.current_num_genes:
            for ind in self.individuals:
                ind.genes.extend([Gene(source_image.shape[1], source_image.shape[0])
                                  for _ in range(target_genes - len(ind.genes))])
                ind.genes.sort(key=lambda x: x.area, reverse=True)
        self.current_num_genes = target_genes

    def update_mutation_prob(self, new_best_fitness, stagnation_threshold=0.001, improvement_threshold=0.01):
        if abs(new_best_fitness - self.best_fitness) < stagnation_threshold:
            self.stagnation_count += 1
            if self.stagnation_count >= 500:
                self.mutation_prob = min(self.mutation_prob * 1.5, 0.8)
                self.stagnation_count = 0
        else:
            if new_best_fitness - self.best_fitness > improvement_threshold:
                self.mutation_prob = max(self.mutation_prob * 0.5, 0.1)
            self.stagnation_count = 0
        self.best_fitness = new_best_fitness

# SSIM implementation
def calculate_ssim(source, generated):
    c1, c2 = 6.5025, 58.5225
    ssim_sum = 0
```

```
    for k in range(3):
        S = source[:, :, k].astype(np.float32)
        G = generated[:, :, k].astype(np.float32)
        mu_s = np.mean(S)
        mu_g = np.mean(G)
        sigma_s = np.mean((S - mu_s) ** 2)
        sigma_g = np.mean((G - mu_g) ** 2)
        sigma_sg = np.mean((S - mu_s) * (G - mu_g))
        numerator = (2 * mu_s * mu_g + c1) * (2 * sigma_sg + c2)
        denominator = (mu_s ** 2 + mu_g ** 2 + c1) * (sigma_s + sigma_g + c2)
        ssim_sum += numerator / denominator
    return ssim_sum / 3

# Crossover function
def crossover(parent1, parent2, num_genes, width, height):
    child1, child2 = Individual(num_genes, width, height), Individual(num_genes, width, height)
    child1.genes, child2.genes = [], []
    for i in range(num_genes):
        if np.random.random() < 0.5:
            child1.genes.append(deepcopy(parent1.genes[i]))
            child2.genes.append(deepcopy(parent2.genes[i]))
        else:
            child1.genes.append(deepcopy(parent2.genes[i]))
            child2.genes.append(deepcopy(parent1.genes[i]))
    child1.genes.sort(key=lambda x: x.area, reverse=True)
    child2.genes.sort(key=lambda x: x.area, reverse=True)
    return child1, child2

# Tournament selection
def tournament_selection(self, tm_size, use_fitness_sharing=False):
    selected = []
    fitness_key = 'shared_fitness' if use_fitness_sharing else 'fitness'
    for _ in range(len(self.individuals)):
        tournament = np.random.choice(self.individuals, tm_size)
        best = max(tournament, key=lambda x: getattr(x, fitness_key))
        selected.append(deepcopy(best))
    return selected
```

```
# Attach method to Population class
Population.tournament_selection = tournament_selection


# Evolutionary algorithm
def evolutionary_algorithm(source_image, num_inds=20, num_genes=50, tm_size=5, frac_elites=0.2, frac_parents=0.6, mutation_prob=0.2, mutation_type='guided', num_generations=10000, use_adaptive_mutation=False, use_dynamic_triangles=False, use_fitness_sharing=False):
    global source_image  # Ensure source_image is accessible in Population methods
    width, height = source_image.shape[1], source_image.shape[0]
    if width <= 0 or height <= 0:
        raise ValueError(f"Invalid source image dimensions: width={width}, height={height}")
    population = Population(num_inds, num_genes, width, height)
    fitness_history = []
    best_images = []


    for gen in range(num_generations):
        if use_dynamic_triangles:
            population.adjust_triangle_count(gen, num_generations, num_genes)
        population.evaluate(source_image, use_fitness_sharing)
        best_ind = max(population.individuals, key=lambda x: x.raw_fitness)
        fitness_history.append(best_ind.raw_fitness)
        if use_adaptive_mutation:
            population.update_mutation_prob(best_ind.raw_fitness)
        if (gen + 1) % 1000 == 0:
            best_images.append(best_ind.draw_image(source_image.shape))
        num_elites = int(frac_elites * num_inds)
        elites = sorted(population.individuals, key=lambda x: x.fitness, reverse=True)[:num_elites]
        non_elites = population.tournament_selection(tm_size, use_fitness_sharing)
        num_parents = int(frac_parents * num_inds)
        parents = sorted(non_elites, key=lambda x: x.fitness, reverse=True)[:num_parents]
        offspring = []
        for i in range(0, num_parents, 2):
            if i + 1 < num_parents:
                child1, child2 = crossover(parents[i], parents[i+1], population.current_num_genes if use_dynamic_triangles else num_genes, width, height)
                offspring.extend([child1, child2])
        for ind in offspring:
```

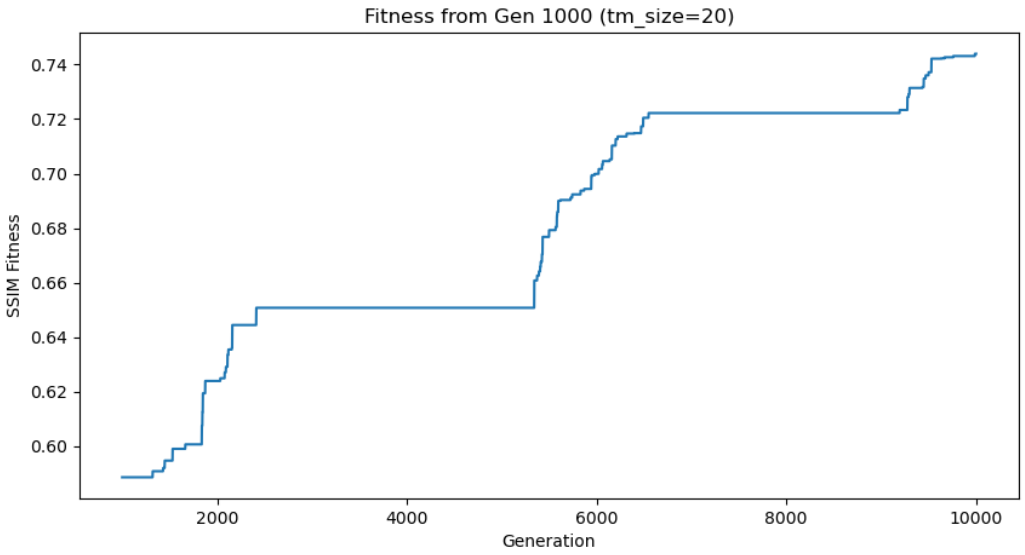
```
            for gene in ind.genes:
                gene.mutate(width, height, mutation_type, population.mutation_prob if use_adaptive_mutation else mutation_prob)
        population.individuals = elites + offspring[:num_inds - num_elites]
    return fitness_history, best_images


# Run all experiments
def run_experiments(source_image):
    output_dir = "output"
    output_improvements_dir = "output_improvements"
    os.makedirs(output_dir, exist_ok=True)
    os.makedirs(output_improvements_dir, exist_ok=True)

    default_params = {
        'num_inds': 20,
        'num_genes': 50,
        'tm_size': 5,
        'frac_elites': 0.2,
        'frac_parents': 0.6,
        'mutation_prob': 0.2,
        'mutation_type': 'guided',
        'num_generations': 10000
    }

    # Hyperparameter experiments
    param_configs = {
        'num_inds': [5, 10, 20, 50, 75],
        'num_genes': [10, 25, 50, 100, 150],
        'tm_size': [2, 5, 10, 20],
        'frac_elites': [0.05, 0.2, 0.4],
        'frac_parents': [0.2, 0.4, 0.6, 0.8],
        'mutation_prob': [0.1, 0.2, 0.5, 0.8],
        'mutation_type': ['guided', 'unguided']
    }

    for param, values in param_configs.items():
        for value in values:
```

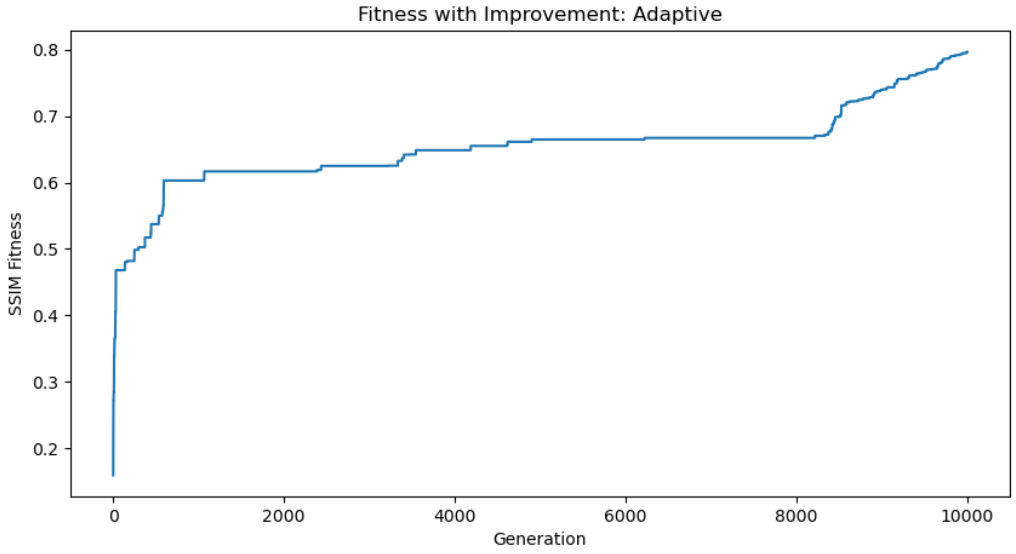
```
        print(f"Running baseline experiment with {param}={value}")
        params = default_params.copy()
        params[param] = value
        fitness_history, best_images = evolutionary_algorithm(source_image, **params)
        plt.figure(figsize=(10, 5))
        plt.plot(range(1, len(fitness_history) + 1), fitness_history)
        plt.title(f"Fitness over Generations ({param}={value})")
        plt.xlabel("Generation")
        plt.ylabel("SSIM Fitness")
        plt.savefig(os.path.join(output_dir, f"fitness_full_{param}_{value}.png"))
        plt.close()
        plt.figure(figsize=(10, 5))
        plt.plot(range(1000, len(fitness_history) + 1), fitness_history[999:])
        plt.title(f"Fitness from Gen 1000 ({param}={value})")
        plt.xlabel("Generation")
        plt.ylabel("SSIM Fitness")
        plt.savefig(os.path.join(output_dir, f"fitness_1000_{param}_{value}.png"))
        plt.close()
        for i, img in enumerate(best_images):
            cv2.imwrite(os.path.join(output_dir, f"best_image_gen_{(i+1)*1000}_{param}_{value}.png"), img)

# Baseline experiment for improvements
print("Running baseline experiment for improvements")
fitness_history, best_images = evolutionary_algorithm(source_image, **default_params)
plt.figure(figsize=(10, 5))
plt.plot(range(1, len(fitness_history) + 1), fitness_history)
plt.title("Fitness over Generations (Baseline)")
plt.xlabel("Generation")
plt.ylabel("SSIM Fitness")
plt.savefig(os.path.join(output_improvements_dir, "fitness_full_baseline.png"))
plt.close()
plt.figure(figsize=(10, 5))
plt.plot(range(1000, len(fitness_history) + 1), fitness_history[999:])
plt.title("Fitness from Gen 1000 (Baseline)")
plt.xlabel("Generation")
plt.ylabel("SSIM Fitness")
plt.savefig(os.path.join(output_improvements_dir, "fitness_1000_baseline.png"))
```

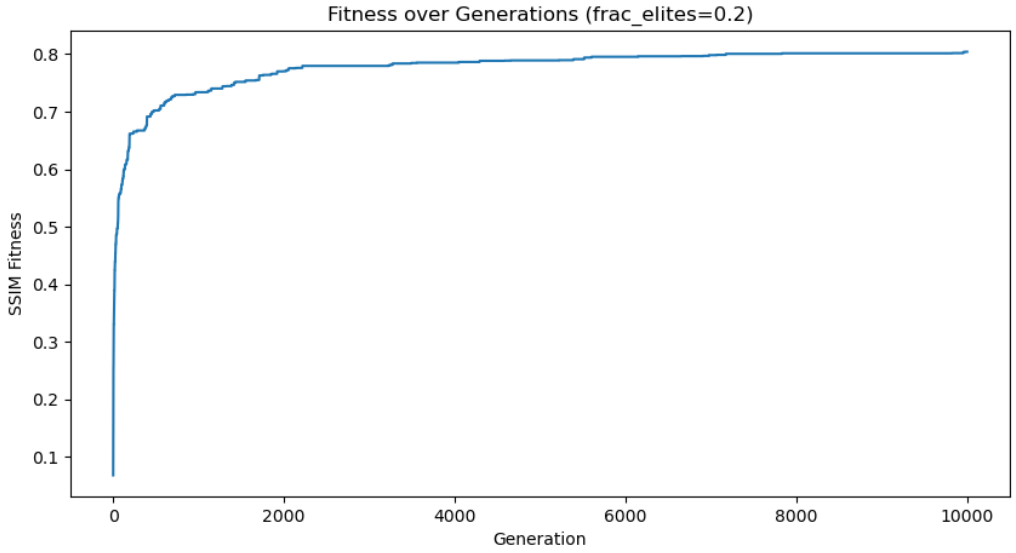
```
    plt.close()
    for i, img in enumerate(best_images):
        cv2.imwrite(os.path.join(output_improvements_dir, f"best_image_gen_{(i+1)*1000}_baseline.png"), img)

    # Improvement experiments
    improvements = [
        ('adaptive_mutation', {'use_adaptive_mutation': True, 'use_dynamic_triangles': False, 'use_fitness_sharing': False}),
        ('dynamic_triangles', {'use_adaptive_mutation': False, 'use_dynamic_triangles': True, 'use_fitness_sharing': False}),
        ('fitness_sharing', {'use_adaptive_mutation': False, 'use_dynamic_triangles': False, 'use_fitness_sharing': True})
    ]

    for improvement_name, improvement_params in improvements:
        print(f"Running experiment with improvement: {improvement_name}")
        params = default_params.copy()
        params.update(improvement_params)
        fitness_history, best_images = evolutionary_algorithm(source_image, **params)
        plt.figure(figsize=(10, 5))
        plt.plot(range(1, len(fitness_history) + 1), fitness_history)
        plt.title(f"Fitness with Improvement: {improvement_name.capitalize()}")
        plt.xlabel("Generation")
        plt.ylabel("SSIM Fitness")
        plt.savefig(os.path.join(output_improvements_dir, f"fitness_full_{improvement_name}.png"))
        plt.close()
        plt.figure(figsize=(10, 5))
        plt.plot(range(1000, len(fitness_history) + 1), fitness_history[999:])
        plt.title(f"Fitness from Gen 1000 with Improvement: {improvement_name.capitalize()}")
        plt.xlabel("Generation")
        plt.ylabel("SSIM Fitness")
        plt.savefig(os.path.join(output_improvements_dir, f"fitness_1000_{improvement_name}.png"))
        plt.close()
        for i, img in enumerate(best_images):
            cv2.imwrite(os.path.join(output_improvements_dir, f"best_image_gen_{(i+1)*1000}_{improvement_name}.png"),
img)

# Main execution
if __name__ == "__main__":
    try:
```

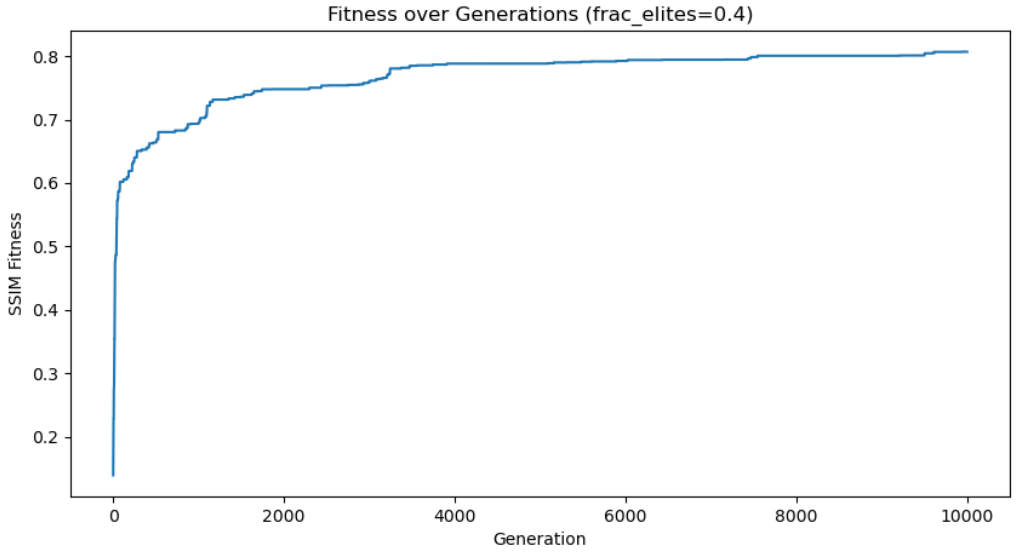
```
    source_image = cv2.imread('painting.png')
    if source_image is None:
        raise FileNotFoundError
except FileNotFoundError:
    print("painting.png not found. Creating a dummy image.")
    source_image = np.random.randint(0, 256, (100, 100, 3), dtype=np.uint8)
    cv2.imwrite('painting.png', source_image)
run_experiments(source_image)
print("Experiments complete. Check 'output' and 'output_improvements' directories for results.")
```

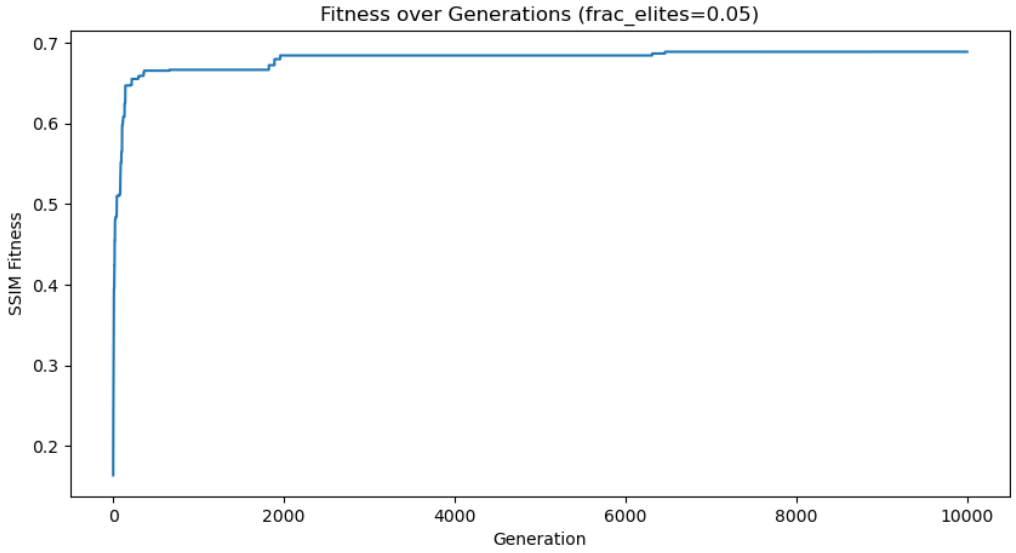
B. SSIM PLOTS

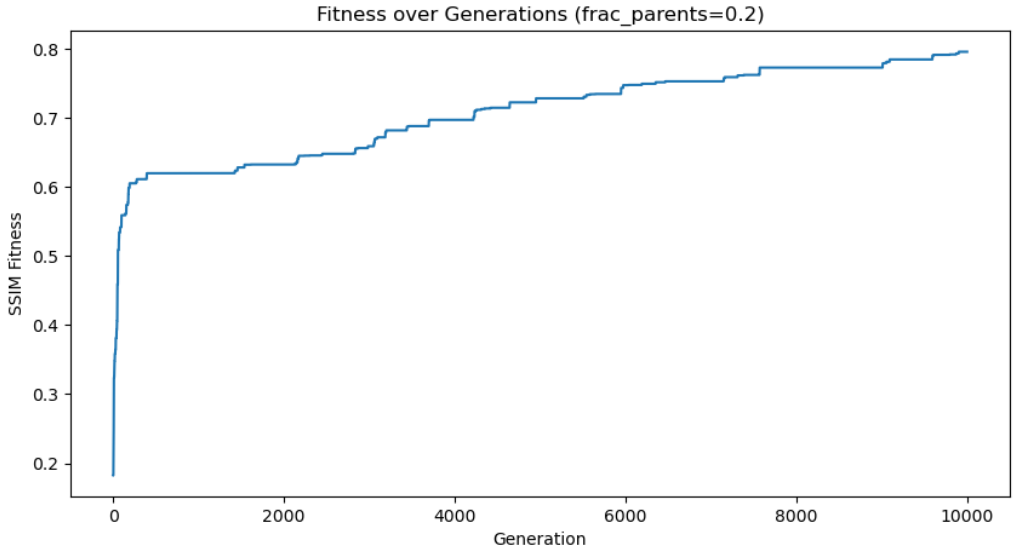


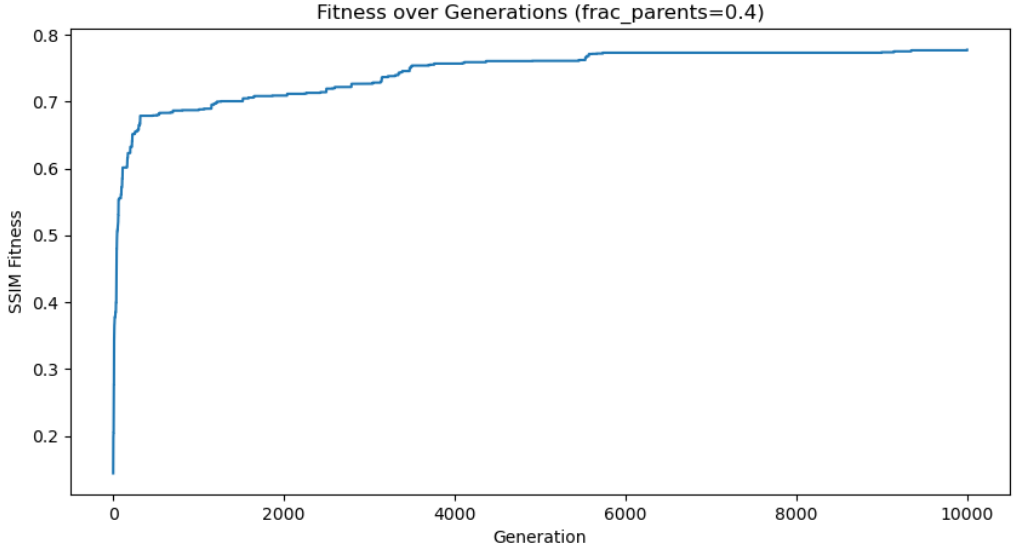


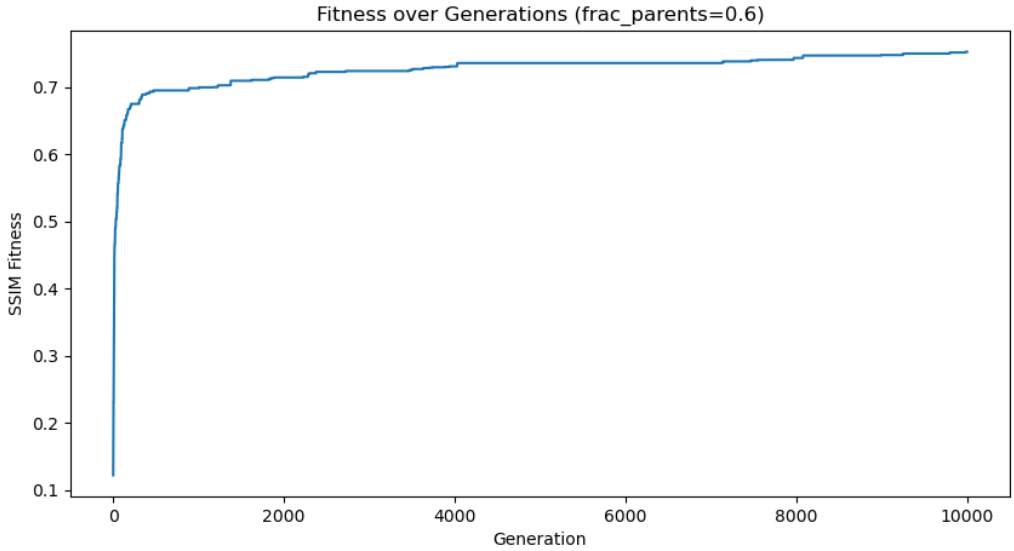

Fitness over Generations (frac_elites=0.2)
SSIM Fitness
Generation
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
2000
4000
6000
8000
10000

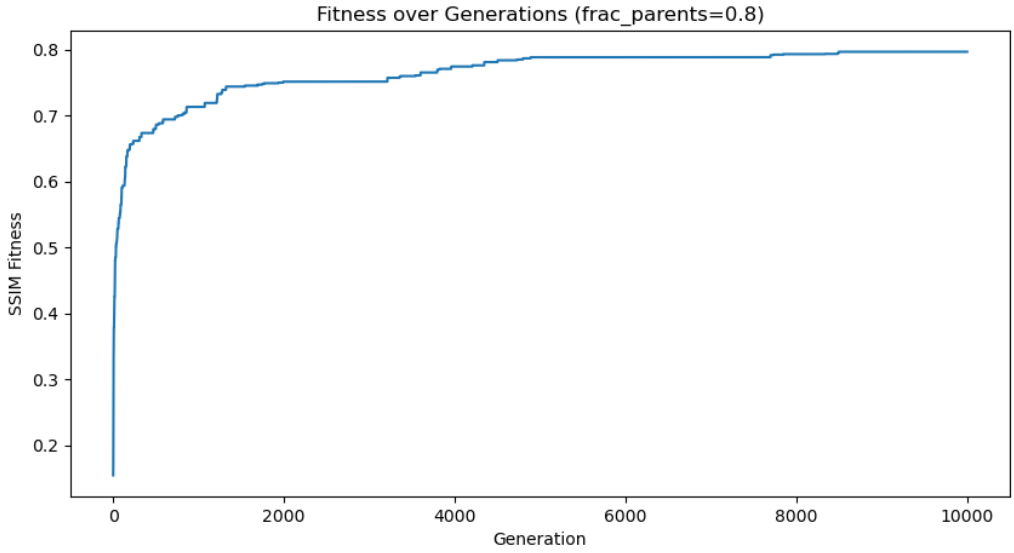

Fitness over Generations (frac_elites=0.4)
SSIM Fitness
Generation
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0
2000
4000
6000
8000
10000

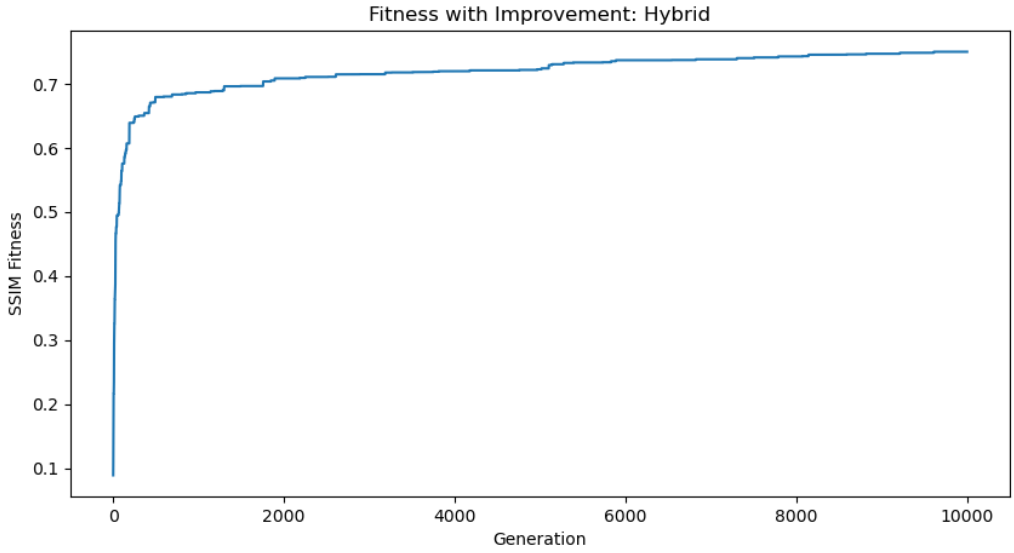

Fitness over Generations (frac_elites=0.05)
SSIM Fitness
Generation
0.7
0.6
0.5
0.4
0.3
0.2
0
2000
4000
6000
8000
10000

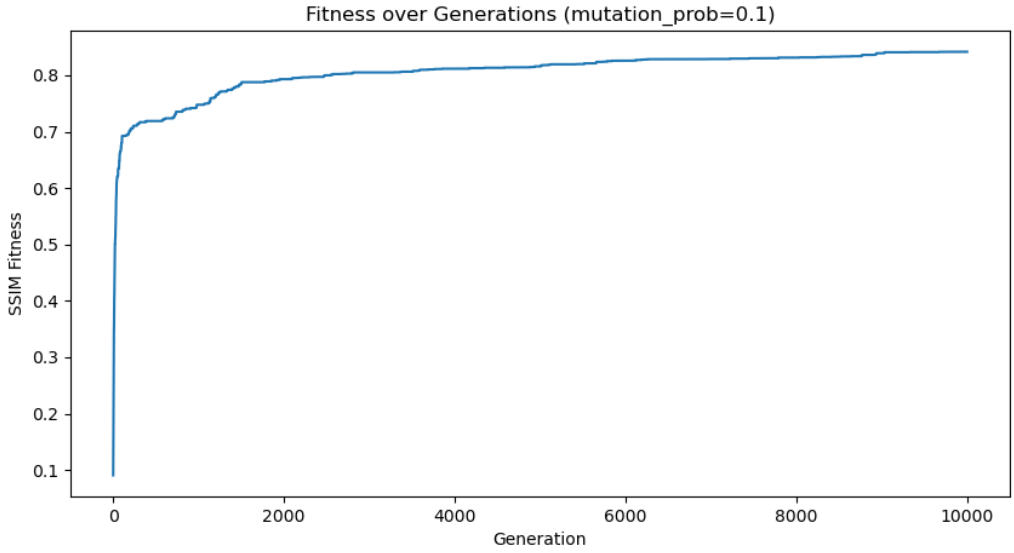

Fitness over Generations (frac_parents=0.2)
SSIM Fitness
Generation
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0
2000
4000
6000
8000
10000

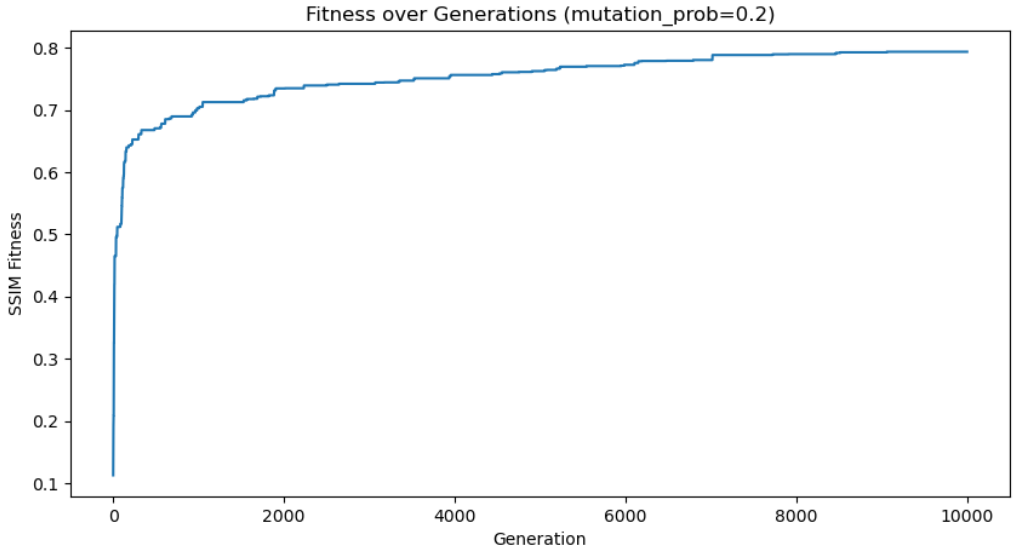

Fitness over Generations (frac_parents=0.4)
SSIM Fitness
Generation
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0
2000
4000
6000
8000
10000

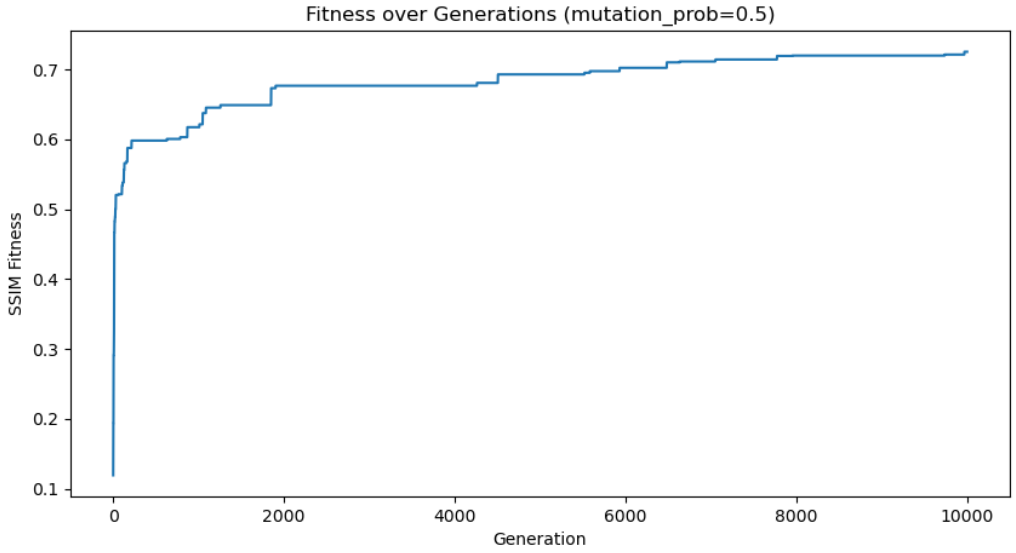

Fitness over Generations (frac_parents=0.6)
SSIM Fitness
Generation
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0
2000
4000
6000
8000
10000

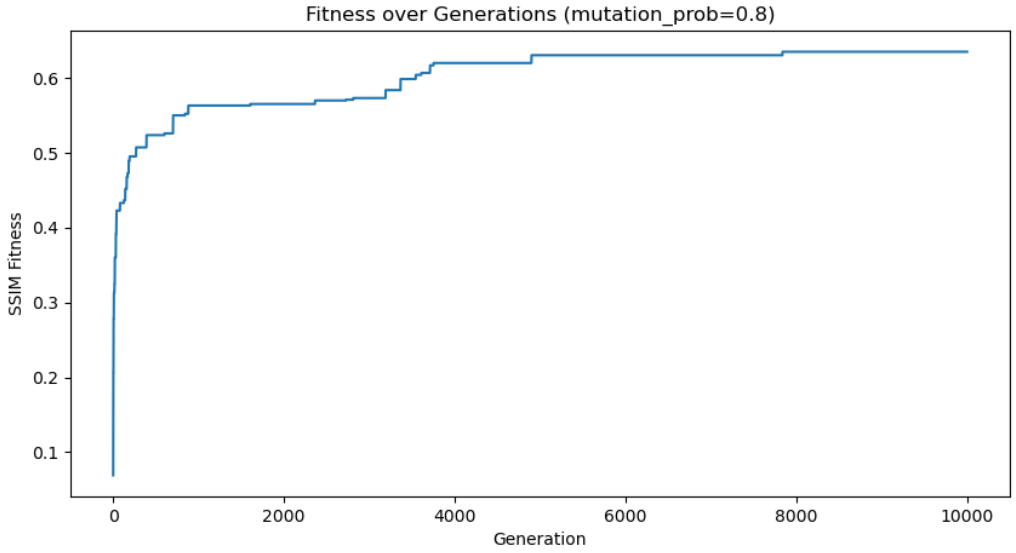

Fitness over Generations (frac_parents=0.8)
SSIM Fitness
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0
2000
4000
6000
8000
10000
Generation

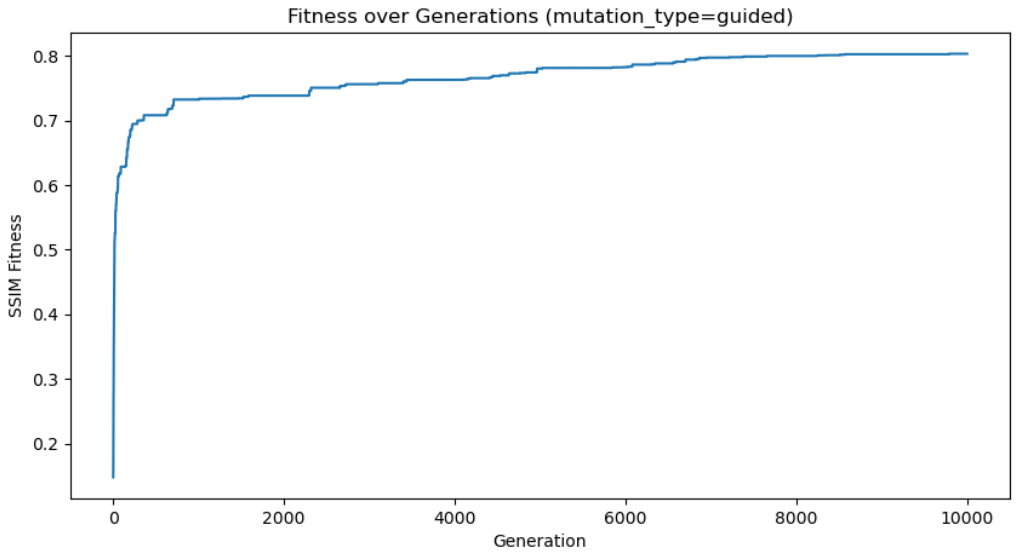

Fitness with Improvement: Hybrid
SSIM Fitness
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
2000
4000
6000
8000
10000
Generation

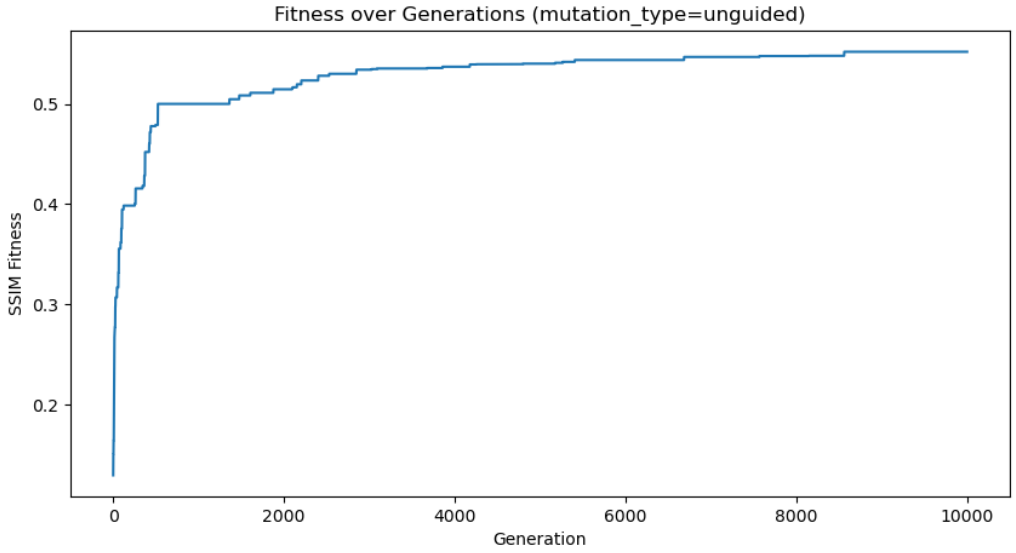

Fitness over Generations (mutation_prob=0.1)
SSIM Fitness
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
2000
4000
6000
8000
10000
Generation

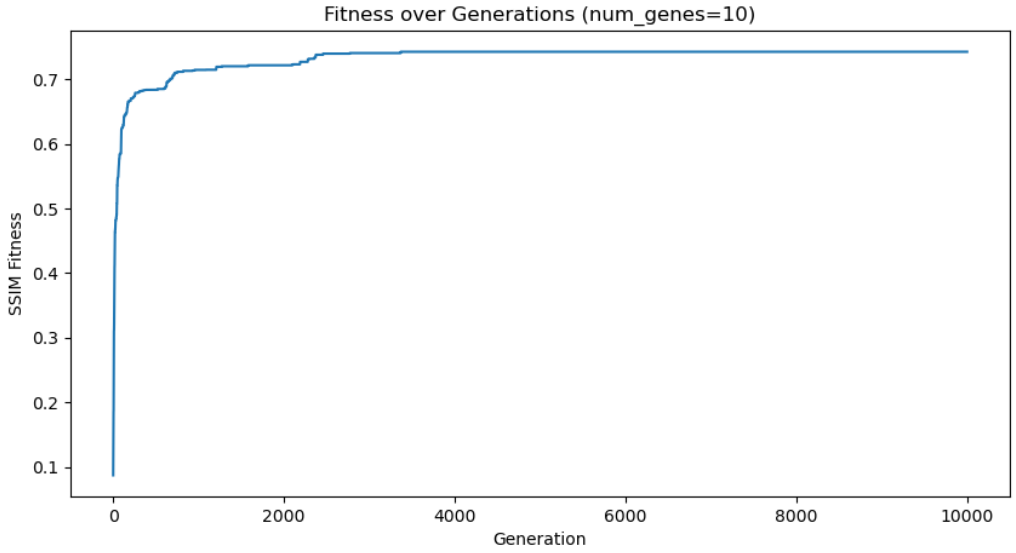

Fitness over Generations (mutation_prob=0.2)
SSIM Fitness
Generation
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0
2000
4000
6000
8000
10000

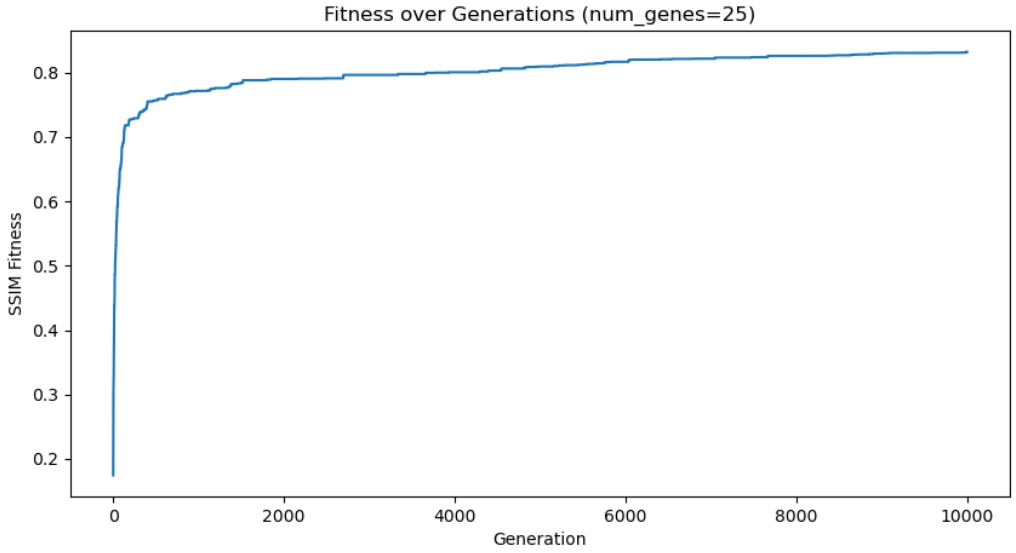

Fitness over Generations (mutation_prob=0.5)
SSIM Fitness
Generation
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0
2000
4000
6000
8000
10000

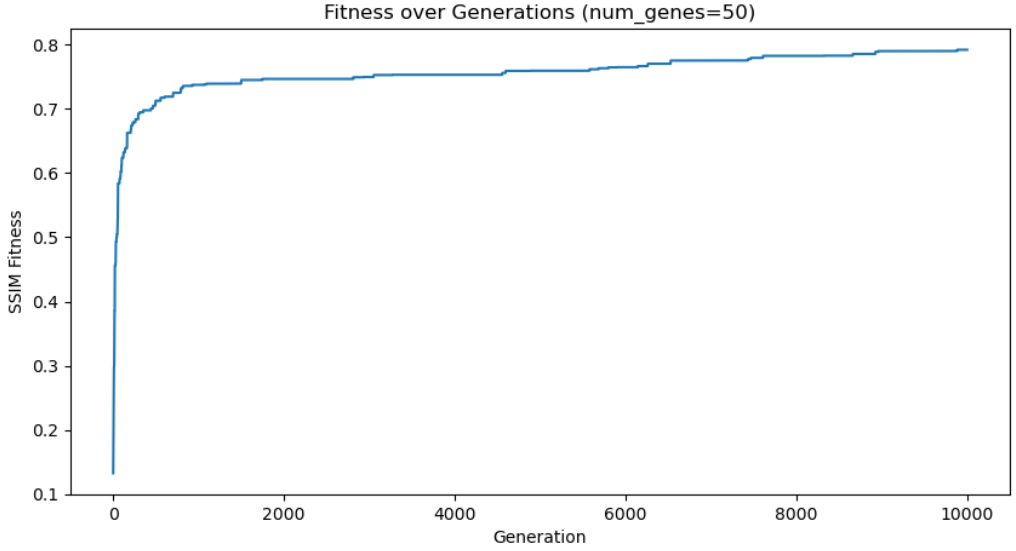

Fitness over Generations (mutation_prob=0.8)
SSIM Fitness
Generation
0.1
0.2
0.3
0.4
0.5
0.6
0
2000
4000
6000
8000
10000

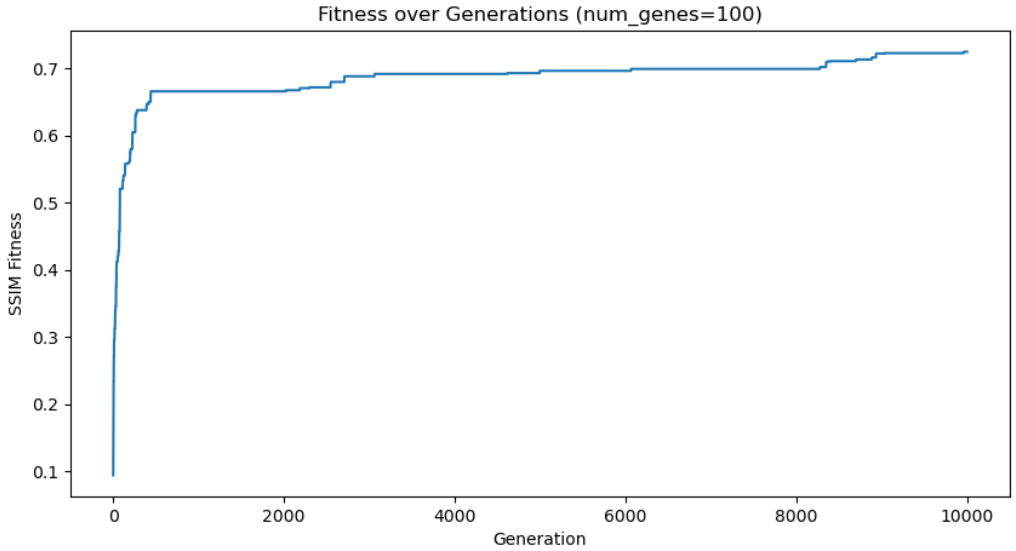

Fitness over Generations (mutation_type=guided)
SSIM Fitness
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0
2000
4000
6000
8000
10000
Generation

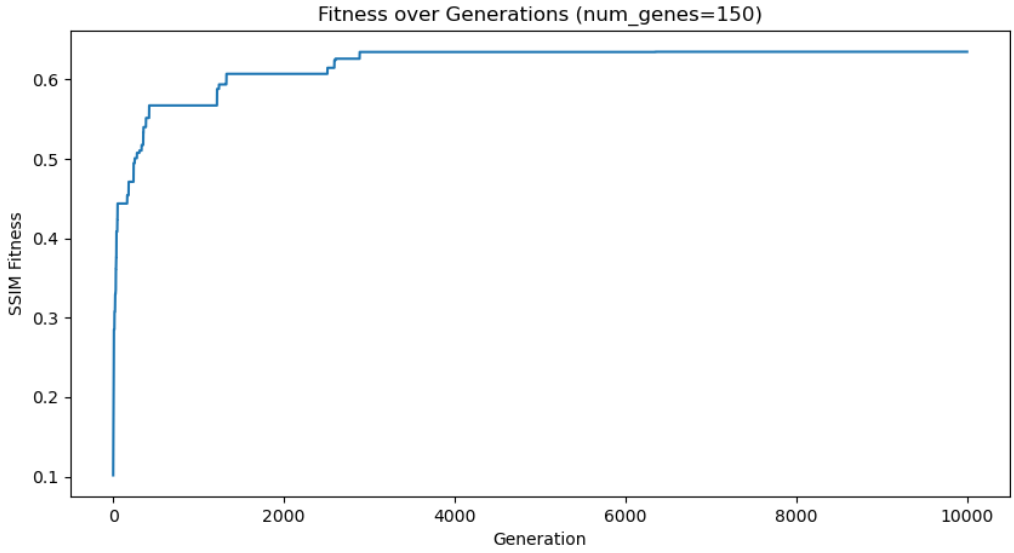

Fitness over Generations (mutation_type=unguided)
SSIM Fitness
0.5
0.4
0.3
0.2
0
2000
4000
6000
8000
10000
Generation

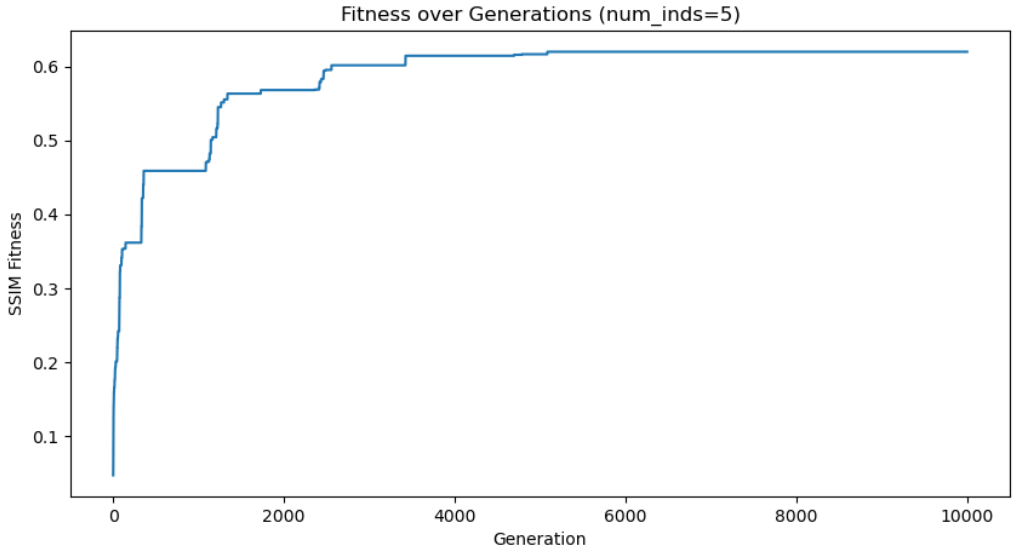

Fitness over Generations (num_genes=10)
SSIM Fitness
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
2000
4000
6000
8000
10000
Generation

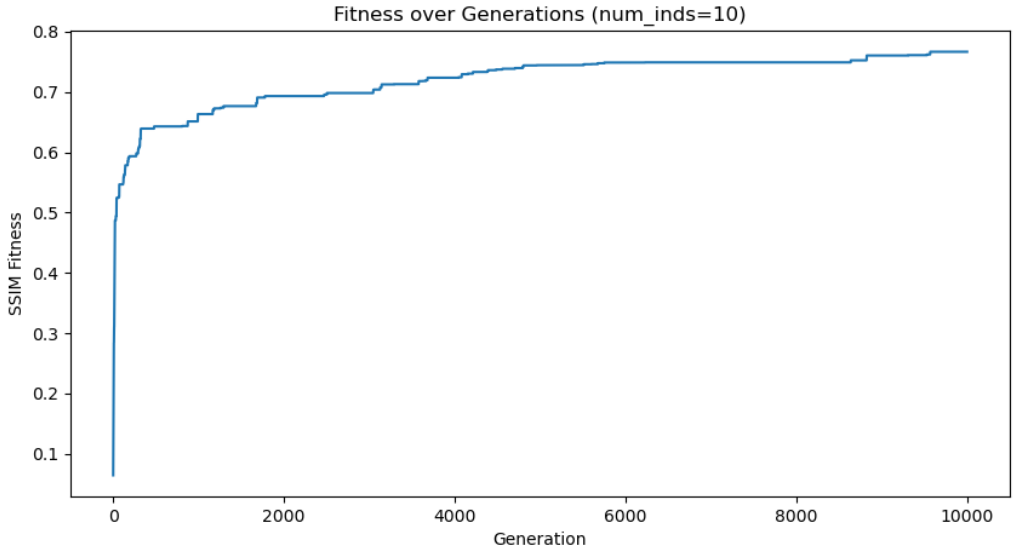

Fitness over Generations (num_genes=25)
SSIM Fitness
Generation
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0
2000
4000
6000
8000
10000

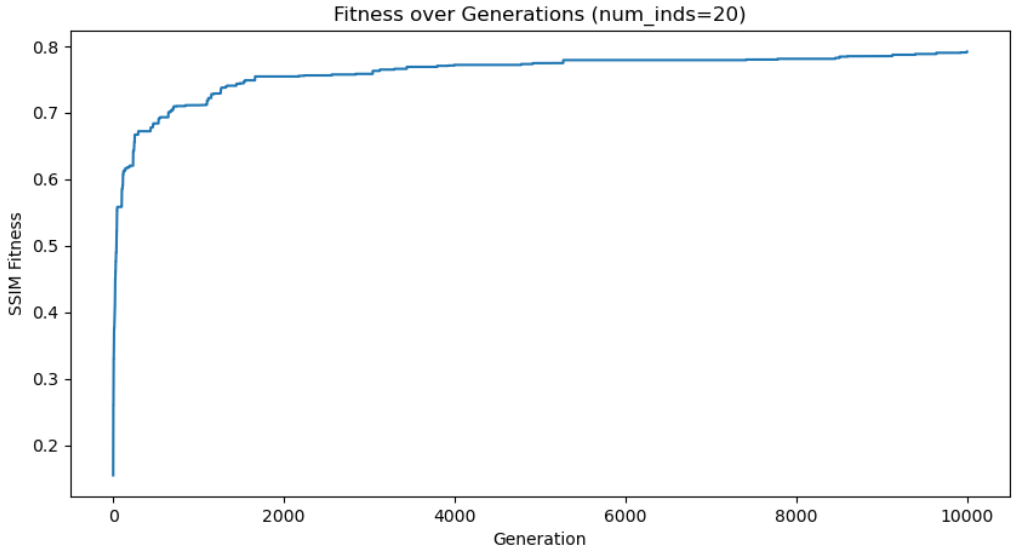

Fitness over Generations (num_genes=50)
SSIM Fitness
Generation
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0
2000
4000
6000
8000
10000

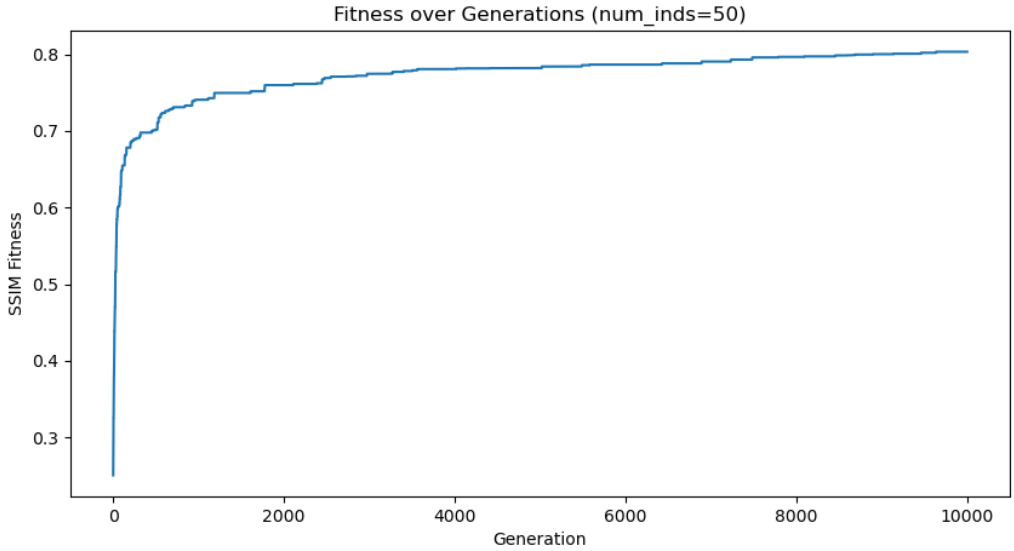

Fitness over Generations (num_genes=100)
SSIM Fitness
Generation
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0
2000
4000
6000
8000
10000

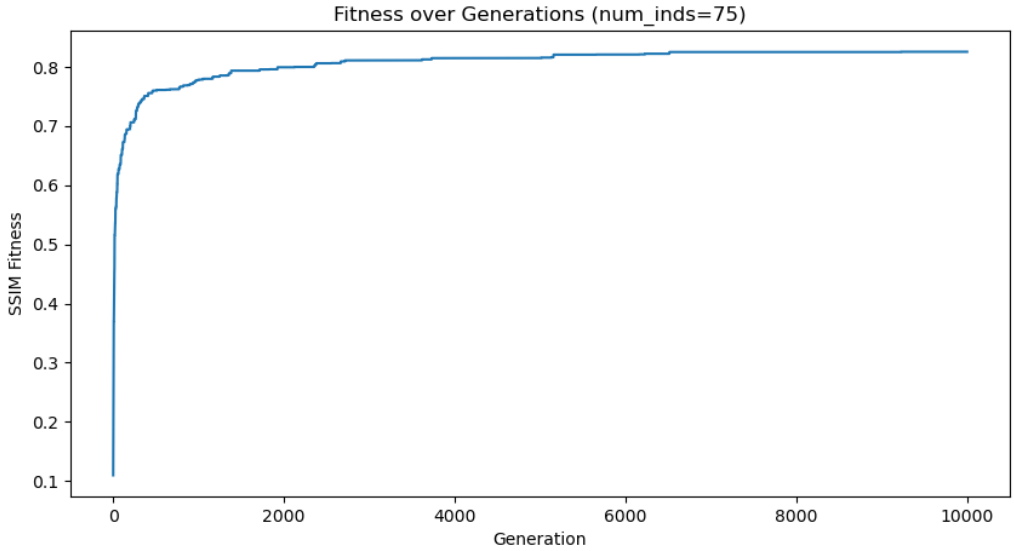

Fitness over Generations (num_genes=150)
SSIM Fitness
Generation
0.1
0.2
0.3
0.4
0.5
0.6
0
2000
4000
6000
8000
10000

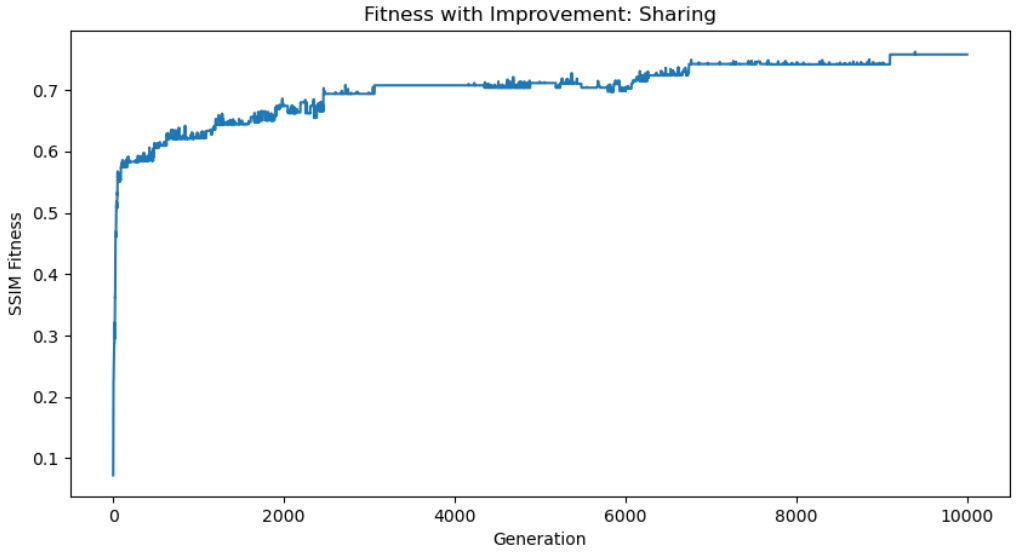

Fitness over Generations (num_inds=5)
SSIM Fitness
Generation
0.1
0.2
0.3
0.4
0.5
0.6
0
2000
4000
6000
8000
10000

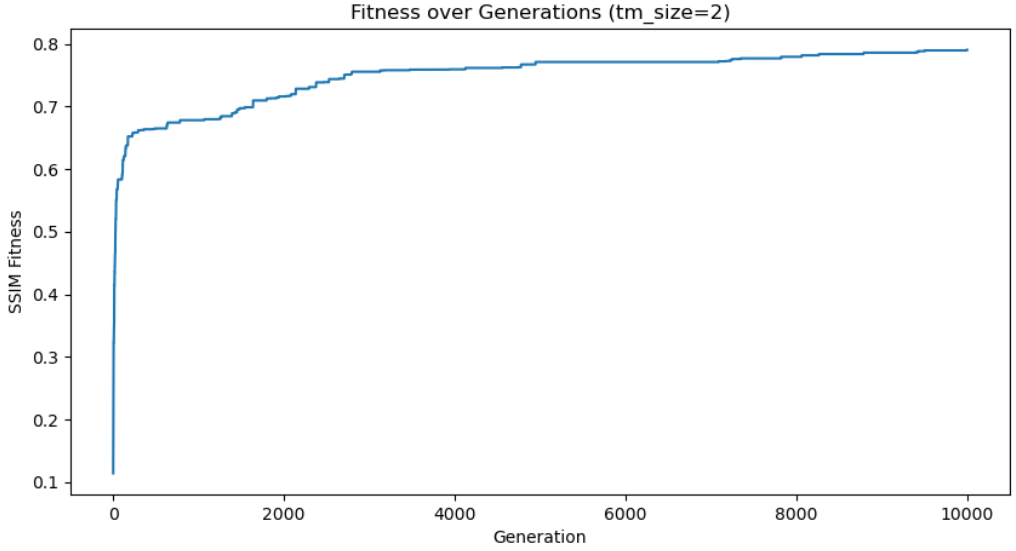

Fitness over Generations (num_inds=10)
SSIM Fitness
Generation
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0
2000
4000
6000
8000
10000

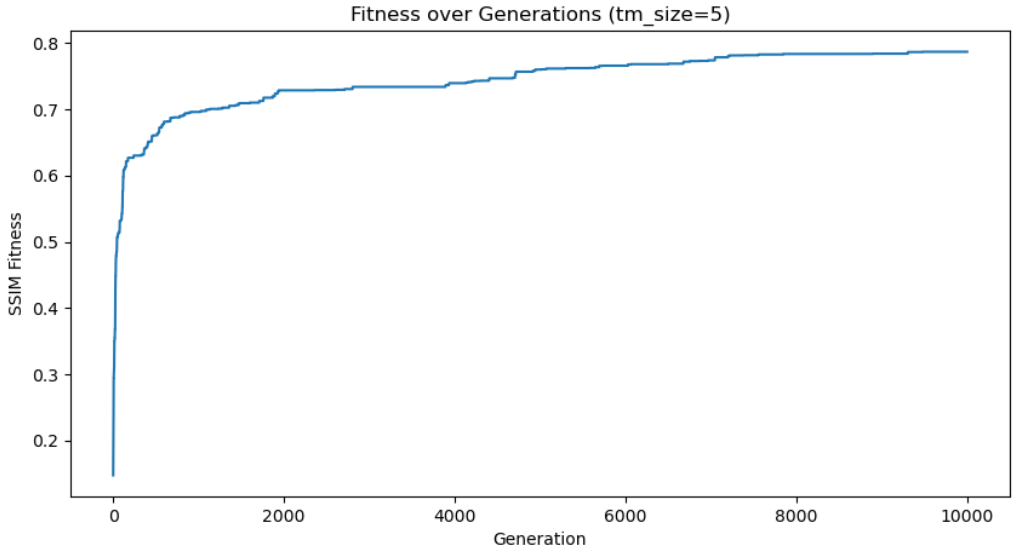

Fitness over Generations (num_inds=20)
SSIM Fitness
Generation
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0
2000
4000
6000
8000
10000

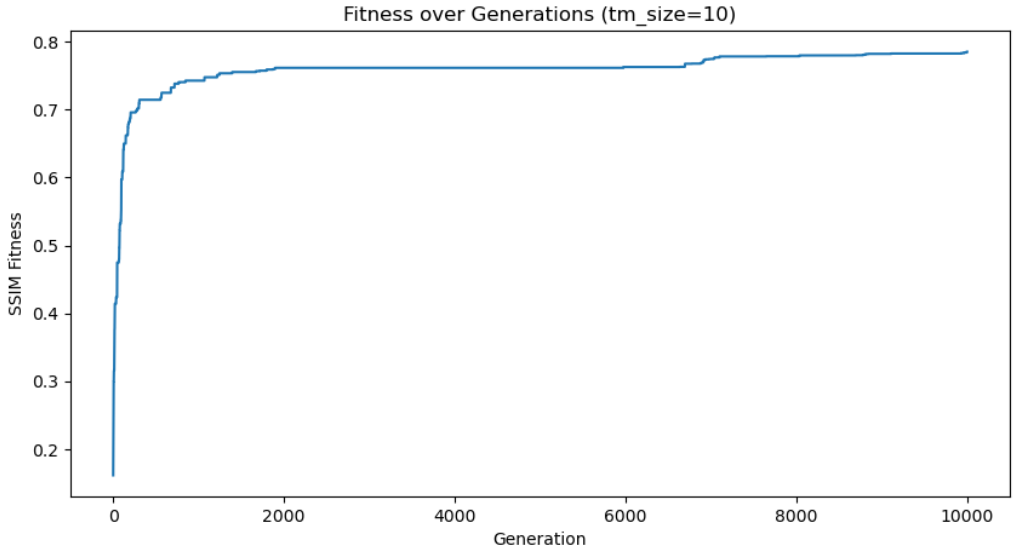

Fitness over Generations (num_inds=50)
SSIM Fitness
Generation
0.3
0.4
0.5
0.6
0.7
0.8
0
2000
4000
6000
8000
10000

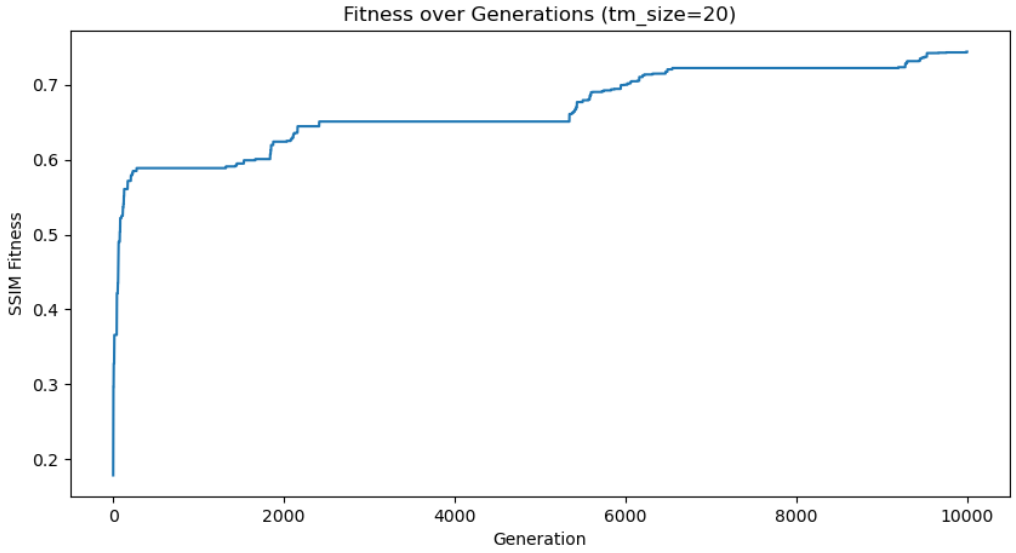

Fitness over Generations (num_inds=75)
SSIM Fitness
Generation
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0
2000
4000
6000
8000
10000

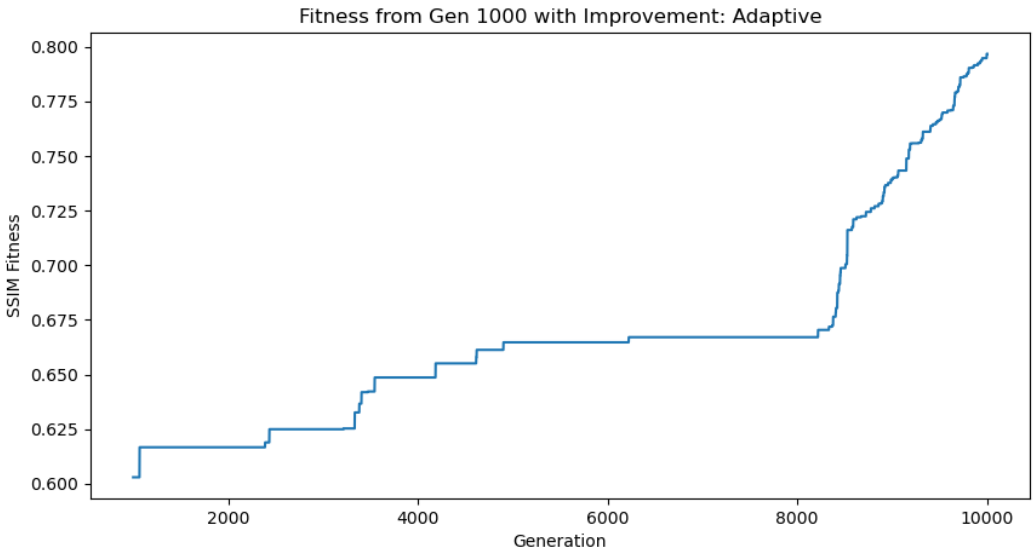

Fitness with Improvement: Sharing
SSIM Fitness
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0
2000
4000
6000
8000
10000
Generation

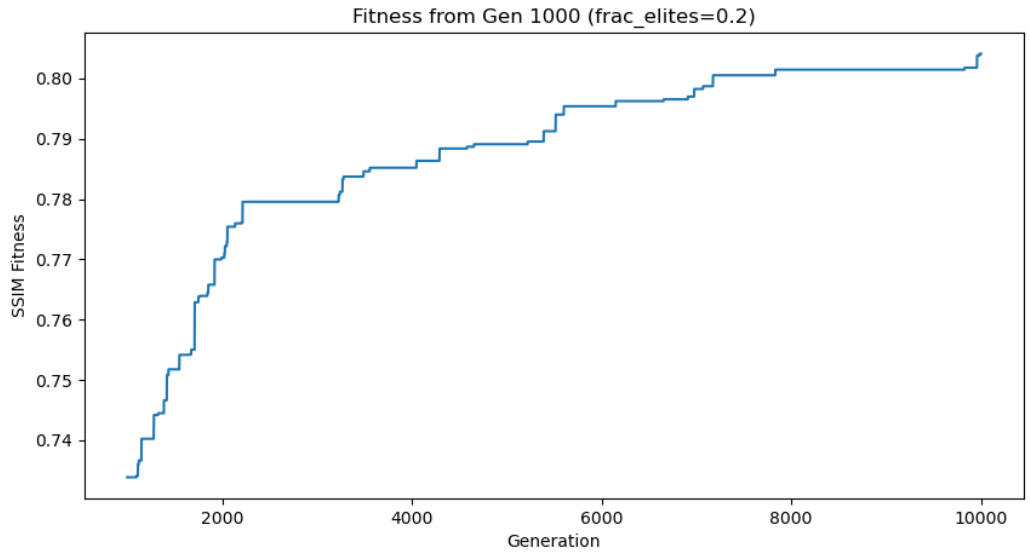

Fitness over Generations (tm_size=2)
SSIM Fitness
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0
2000
4000
6000
8000
10000
Generation

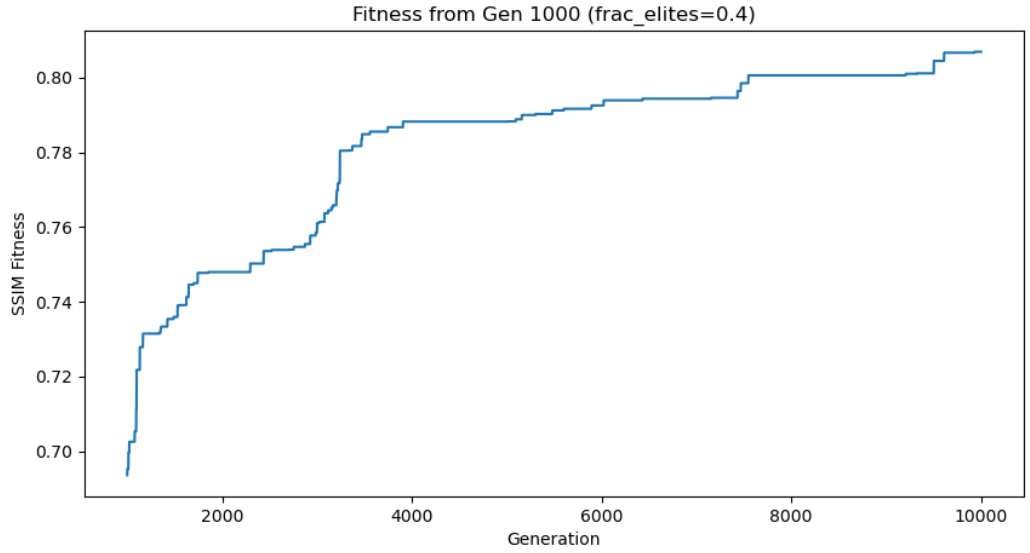

Fitness over Generations (tm_size=5)
SSIM Fitness
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0
2000
4000
6000
8000
10000
Generation

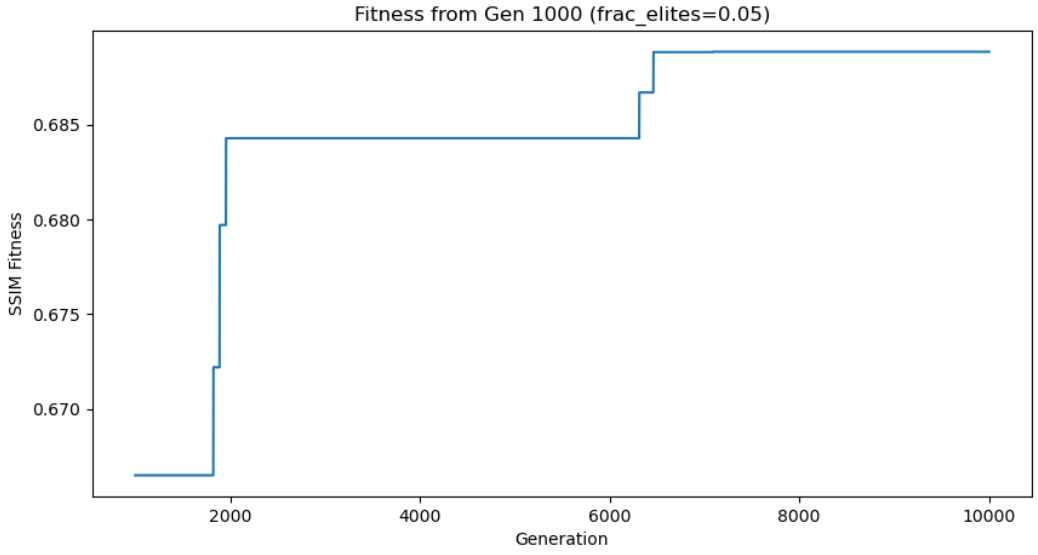

Fitness over Generations (tm_size=10)
SSIM Fitness
Generation
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0
2000
4000
6000
8000
10000

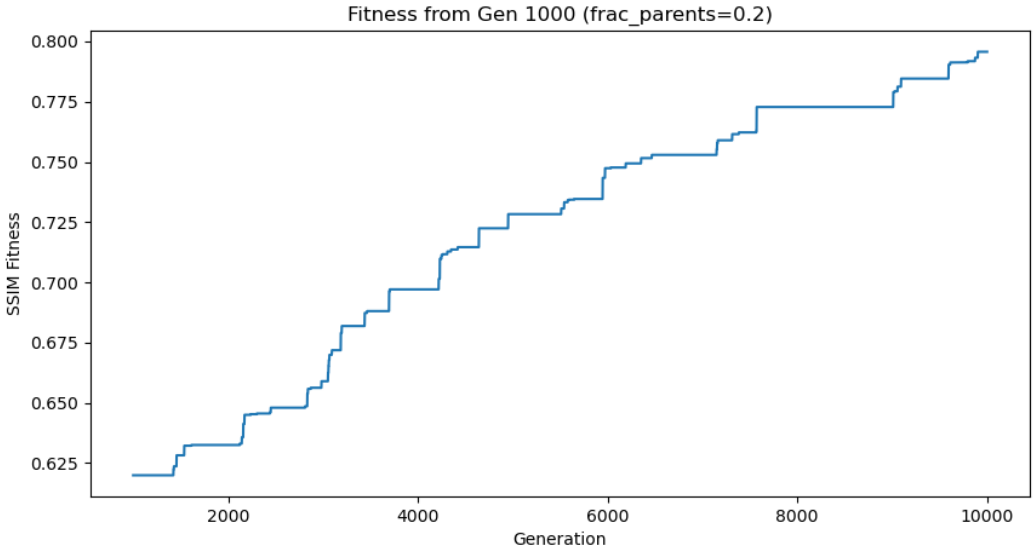

Fitness over Generations (tm_size=20)
SSIM Fitness
Generation
0.2
0.3
0.4
0.5
0.6
0.7
0
2000
4000
6000
8000
10000

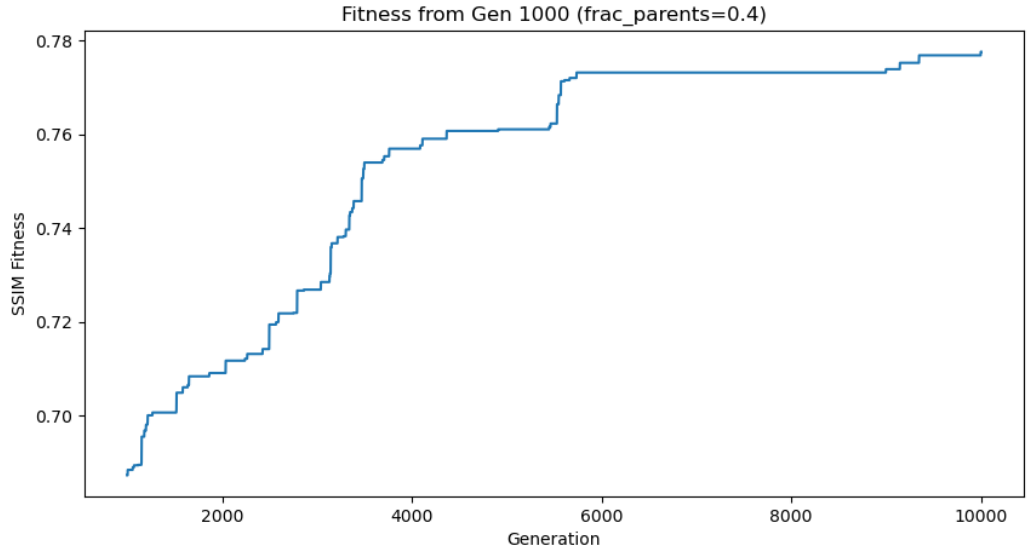

Fitness from Gen 1000 with Improvement: Adaptive
SSIM Fitness
Generation
0.600
0.625
0.650
0.675
0.700
0.725
0.750
0.775
0.800
2000
4000
6000
8000
10000

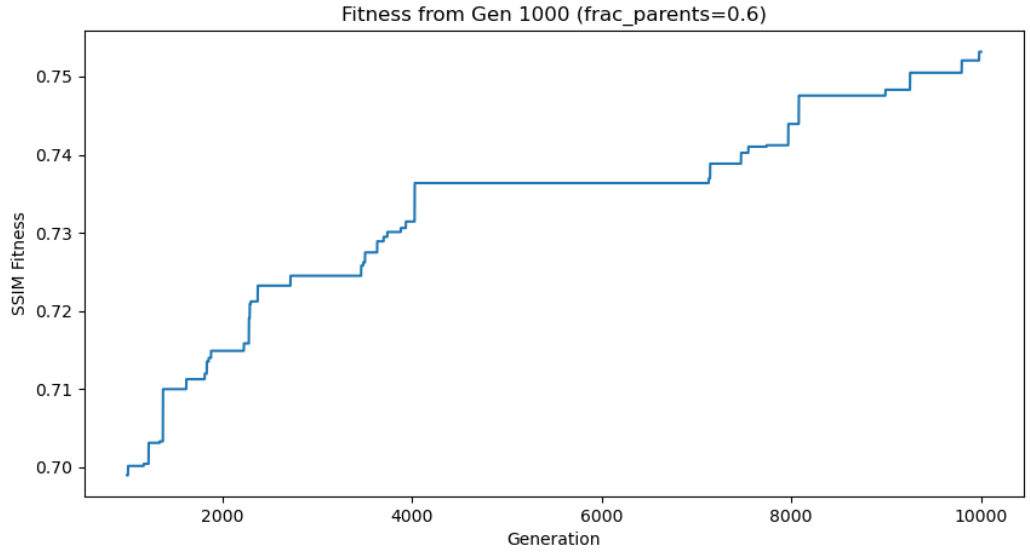

Fitness from Gen 1000 (frac_elites=0.2)
SSIM Fitness
0.80
0.79
0.78
0.77
0.76
0.75
0.74
2000
4000
6000
8000
10000
Generation

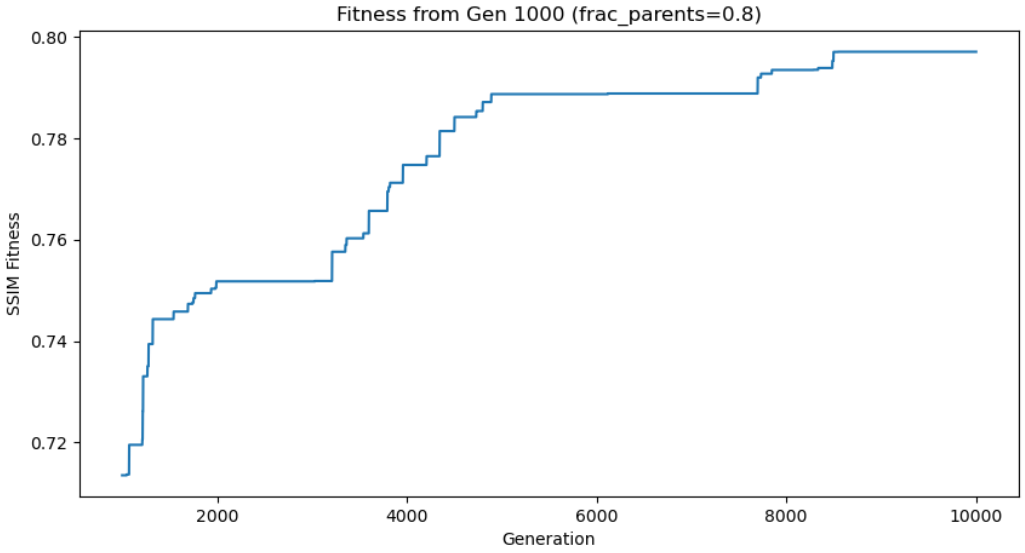

Fitness from Gen 1000 (frac_elites=0.4)
SSIM Fitness
0.80
0.78
0.76
0.74
0.72
0.70
2000
4000
6000
8000
10000
Generation

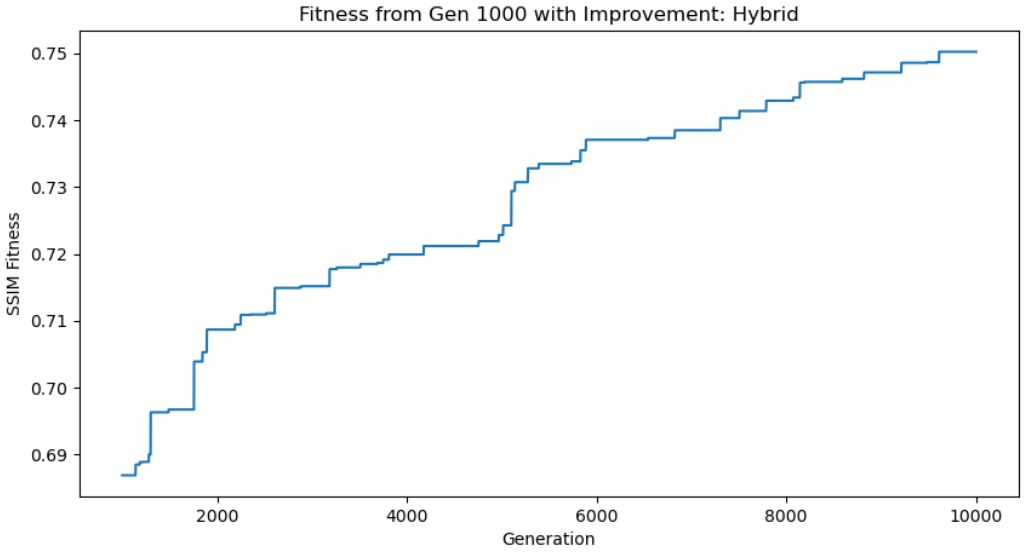

Fitness from Gen 1000 (frac_elites=0.05)
SSIM Fitness
0.685
0.680
0.675
0.670
2000
4000
6000
8000
10000
Generation


Fitness from Gen 1000 (frac_parents=0.2)
SSIM Fitness
0.800
0.775
0.750
0.725
0.700
0.675
0.650
0.625
2000
4000
6000
8000
10000
Generation

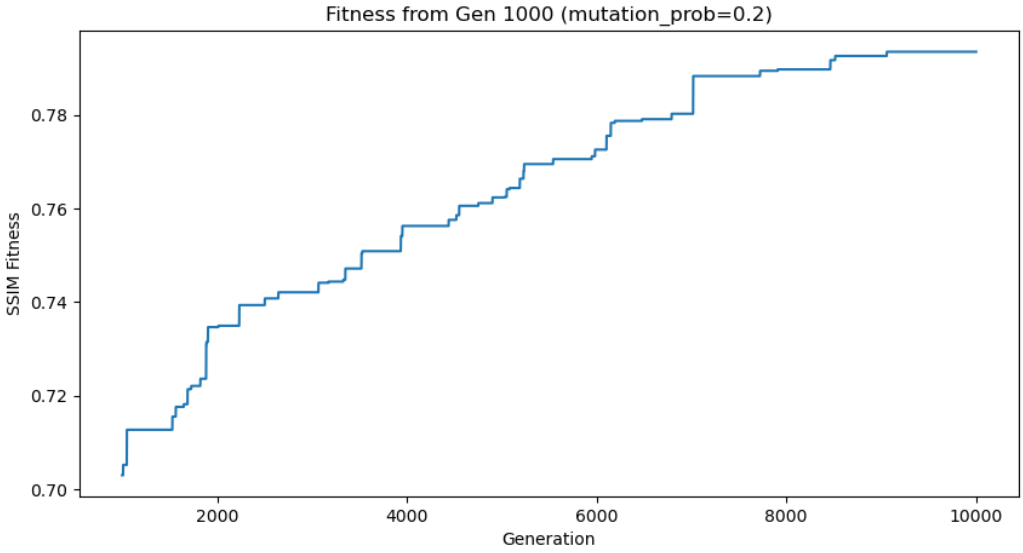

Fitness from Gen 1000 (frac_parents=0.4)
SSIM Fitness
0.78
0.76
0.74
0.72
0.70
2000
4000
6000
8000
10000
Generation

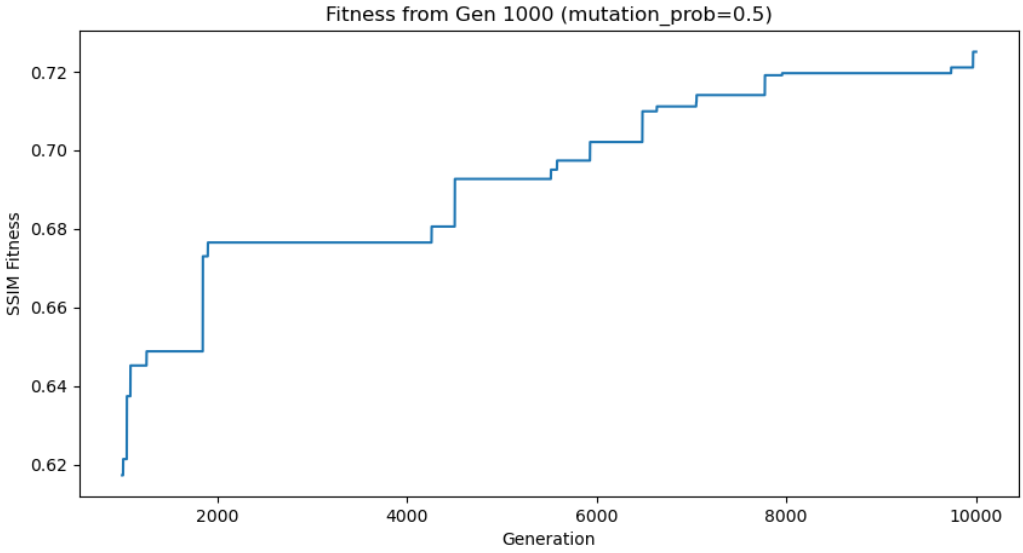

Fitness from Gen 1000 (frac_parents=0.6)
SSIM Fitness
0.75
0.74
0.73
0.72
0.71
0.70
2000
4000
6000
8000
10000
Generation

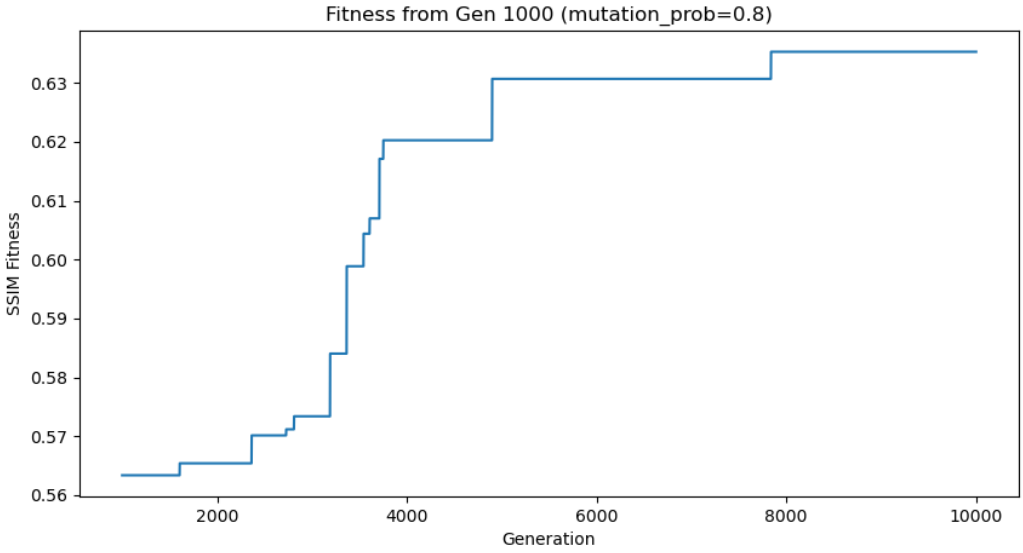

Fitness from Gen 1000 (frac_parents=0.8)
SSIM Fitness
0.80
0.78
0.76
0.74
0.72
2000
4000
6000
8000
10000
Generation

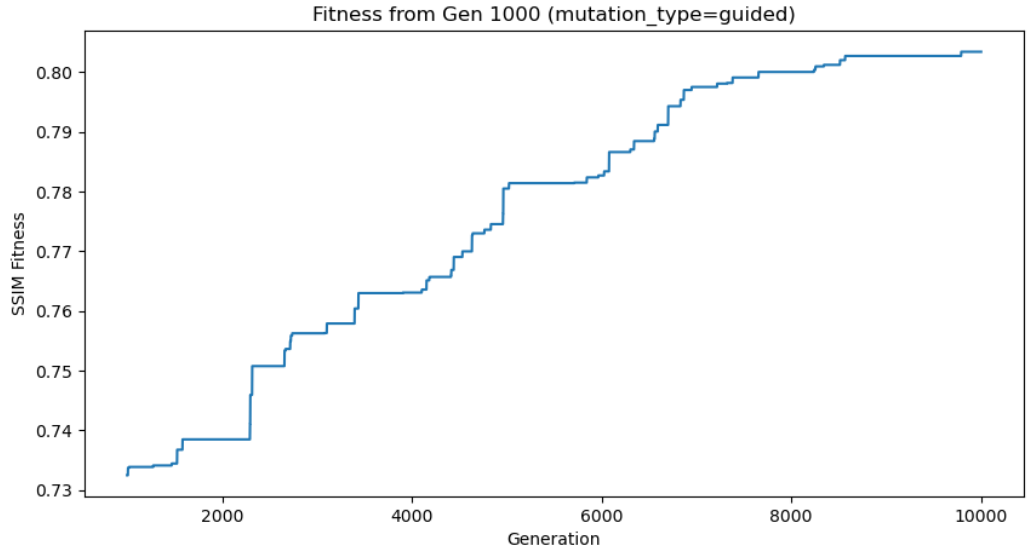

Fitness from Gen 1000 with Improvement: Hybrid
SSIM Fitness
0.75
0.74
0.73
0.72
0.71
0.70
0.69
2000
4000
6000
8000
10000
Generation

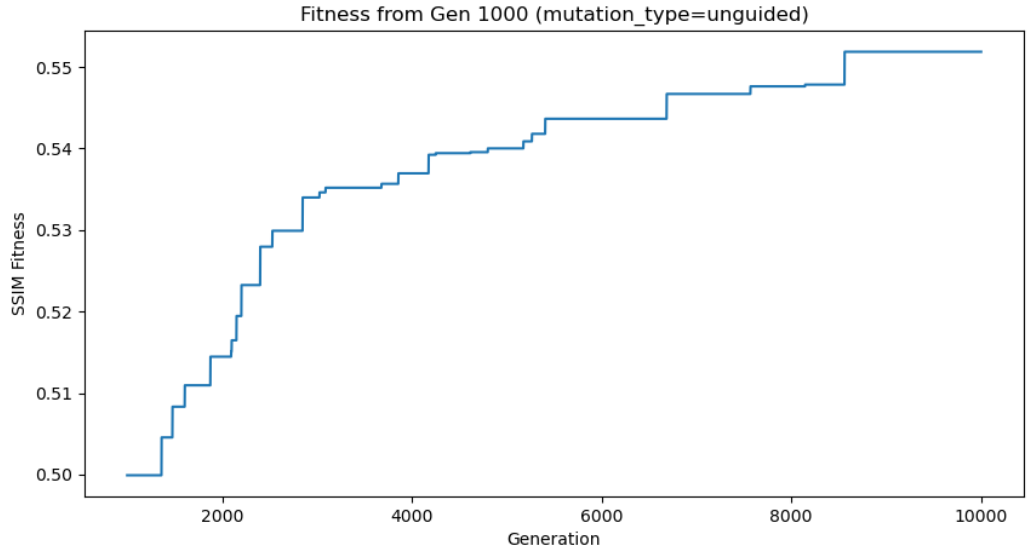

Fitness from Gen 1000 (mutation_prob=0.1)
SSIM Fitness
0.84
0.82
0.80
0.78
0.76
2000
4000
6000
8000
10000
Generation

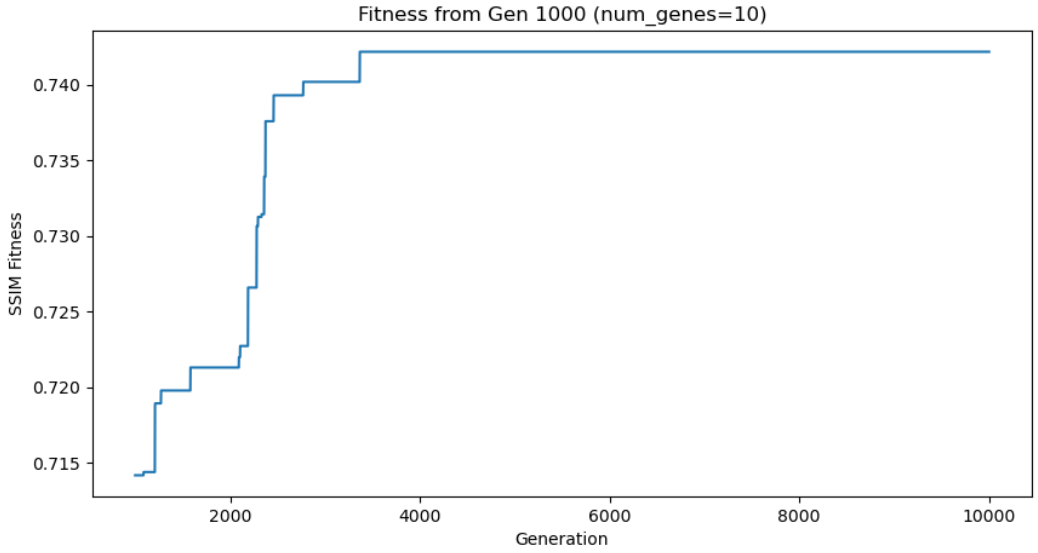

Fitness from Gen 1000 (mutation_prob=0.2)
SSIM Fitness
0.78
0.76
0.74
0.72
0.70
2000
4000
6000
8000
10000
Generation

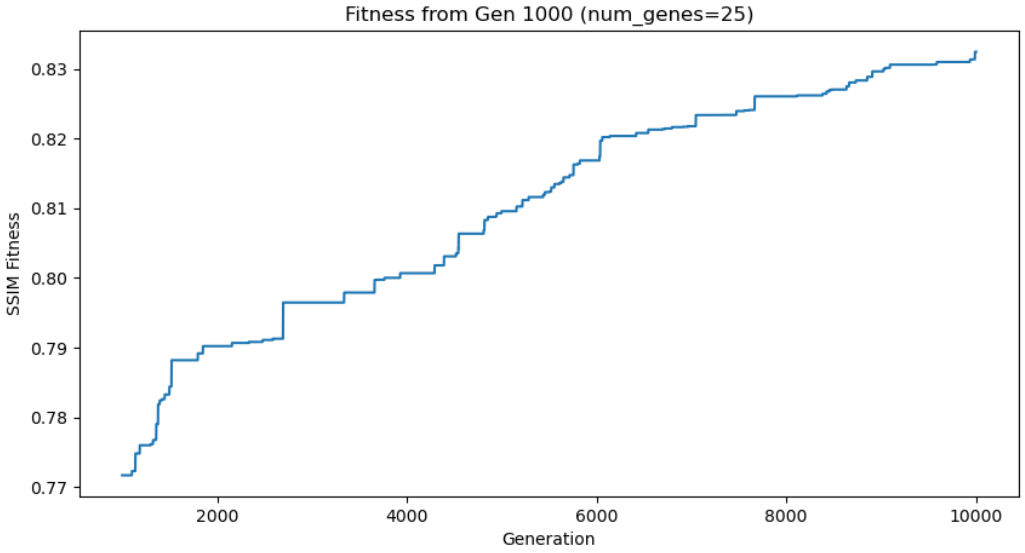

Fitness from Gen 1000 (mutation_prob=0.5)
SSIM Fitness
0.72
0.70
0.68
0.66
0.64
0.62
2000
4000
6000
8000
10000
Generation

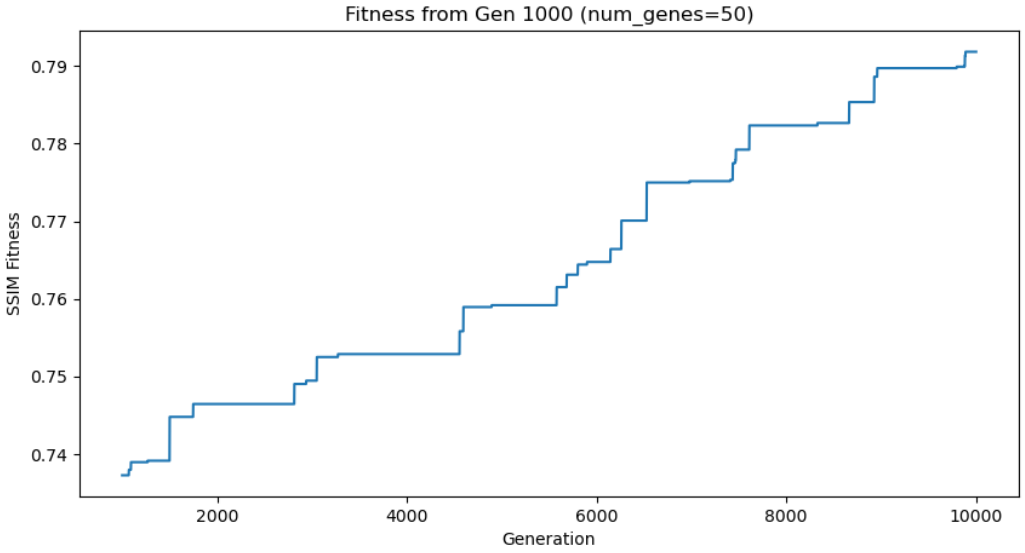

Fitness from Gen 1000 (mutation_prob=0.8)
SSIM Fitness
0.63
0.62
0.61
0.60
0.59
0.58
0.57
0.56
2000
4000
6000
8000
10000
Generation

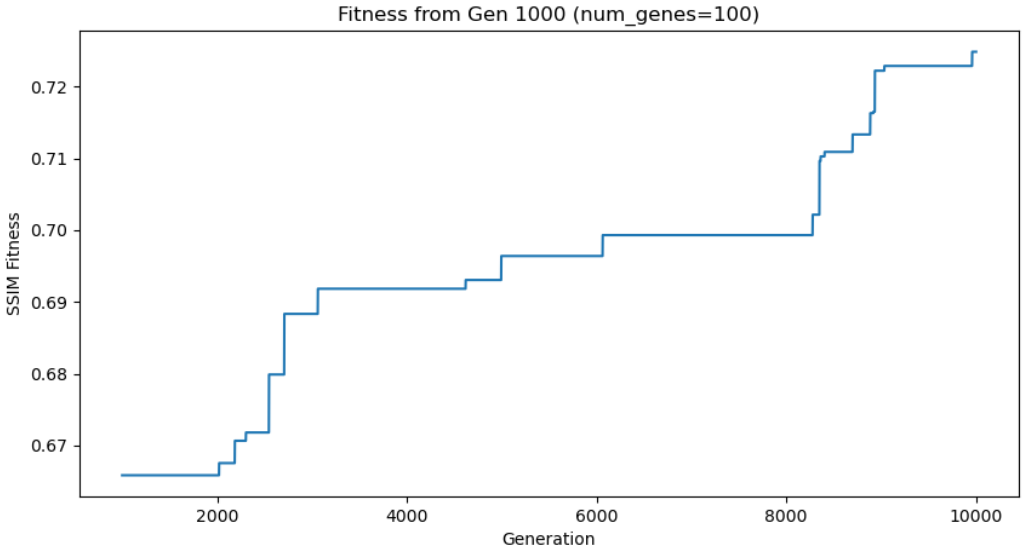

Fitness from Gen 1000 (mutation_type=guided)
SSIM Fitness
0.80
0.79
0.78
0.77
0.76
0.75
0.74
0.73
2000
4000
6000
8000
10000
Generation

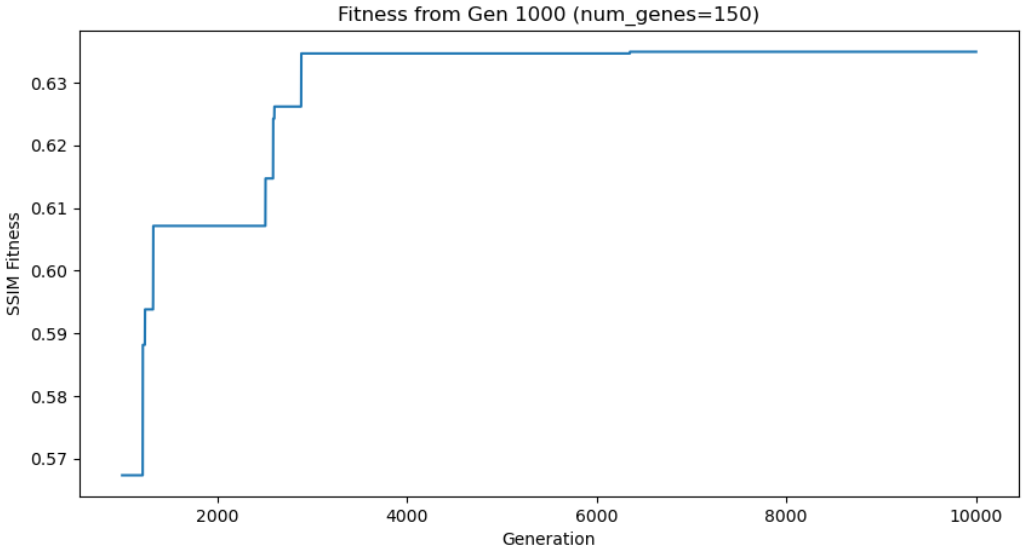

Fitness from Gen 1000 (mutation_type=unguided)
SSIM Fitness
0.55
0.54
0.53
0.52
0.51
0.50
2000
4000
6000
8000
10000
Generation

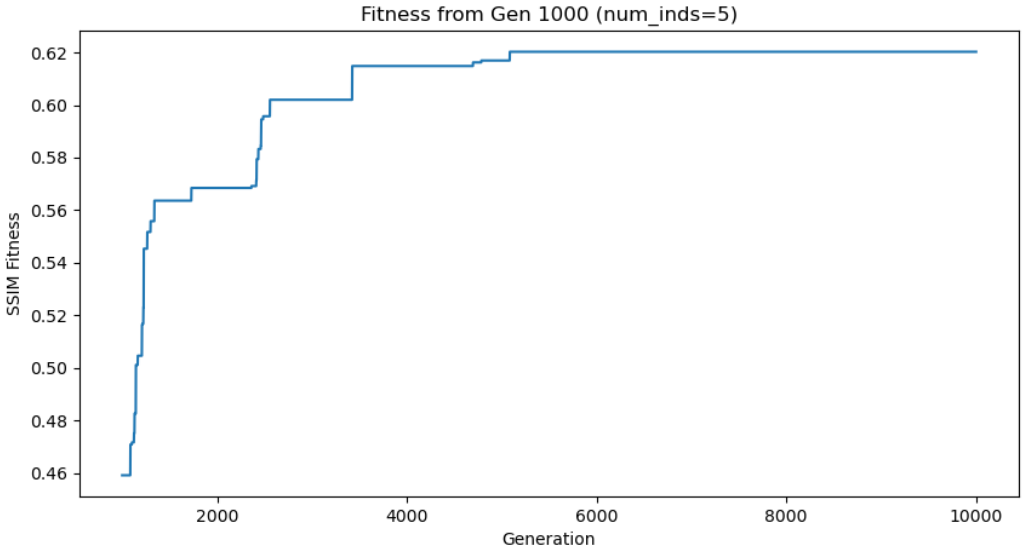

Fitness from Gen 1000 (num_genes=10)
SSIM Fitness
0.740
0.735
0.730
0.725
0.720
0.715
2000
4000
6000
8000
10000
Generation

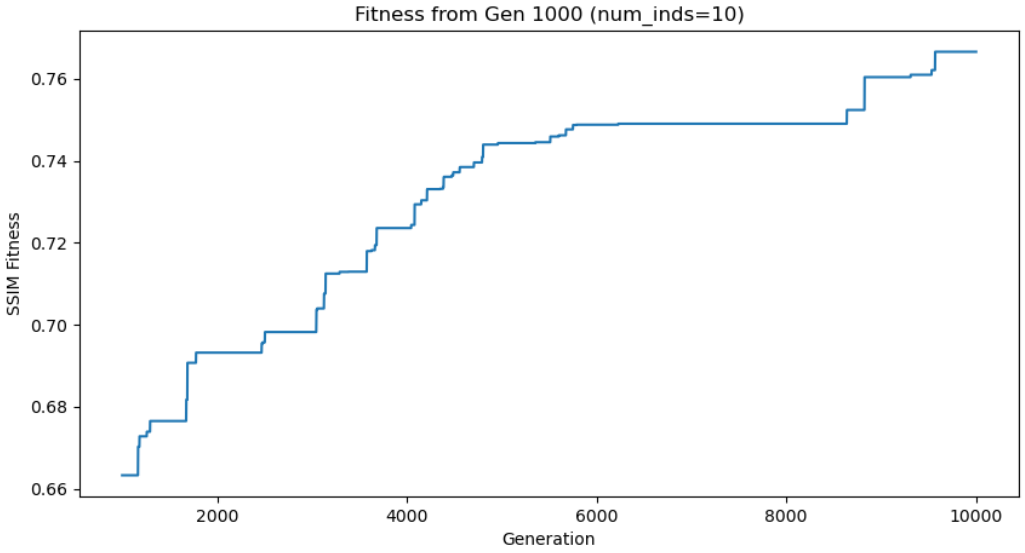

Fitness from Gen 1000 (num_genes=25)
SSIM Fitness
0.83
0.82
0.81
0.80
0.79
0.78
0.77
2000
4000
6000
8000
10000
Generation

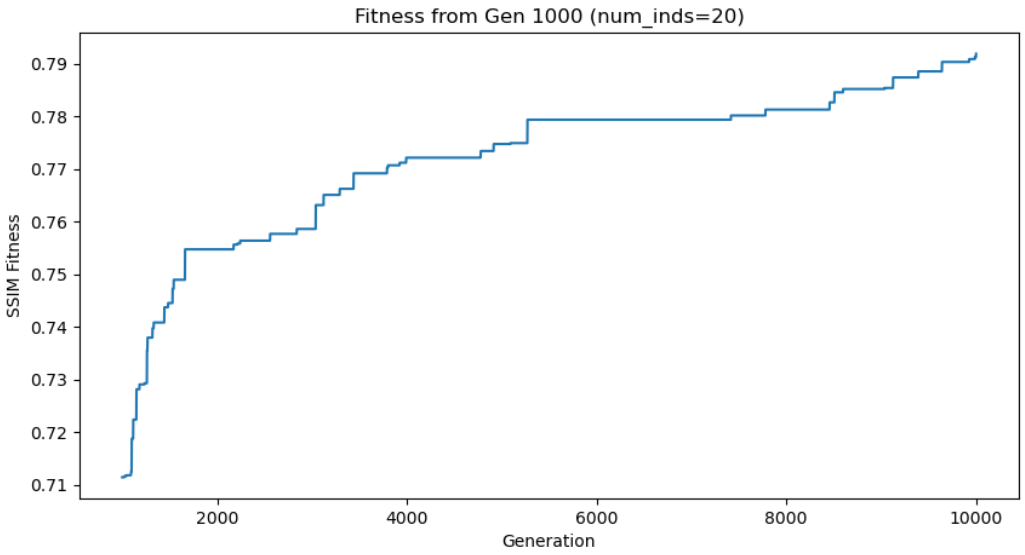

Fitness from Gen 1000 (num_genes=50)
SSIM Fitness
0.79
0.78
0.77
0.76
0.75
0.74
2000
4000
6000
8000
10000
Generation

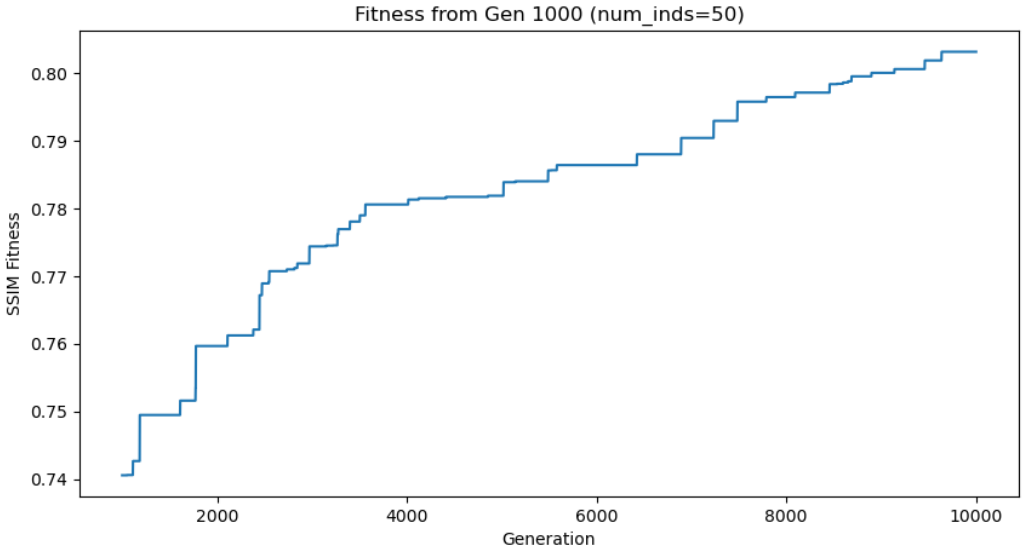

Fitness from Gen 1000 (num_genes=100)
SSIM Fitness
0.72
0.71
0.70
0.69
0.68
0.67
2000
4000
6000
8000
10000
Generation

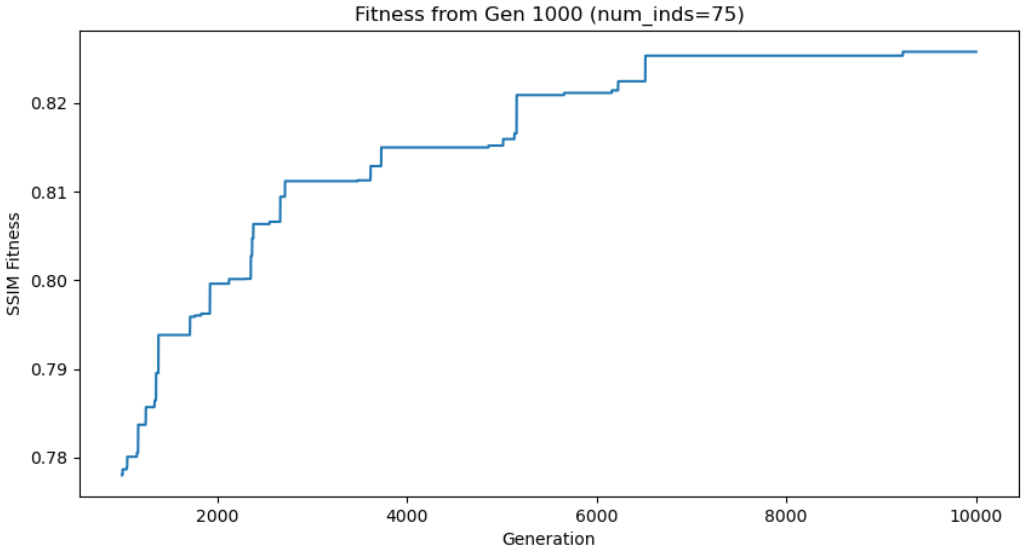

Fitness from Gen 1000 (num_genes=150)
SSIM Fitness
0.63
0.62
0.61
0.60
0.59
0.58
0.57
2000
4000
6000
8000
10000
Generation

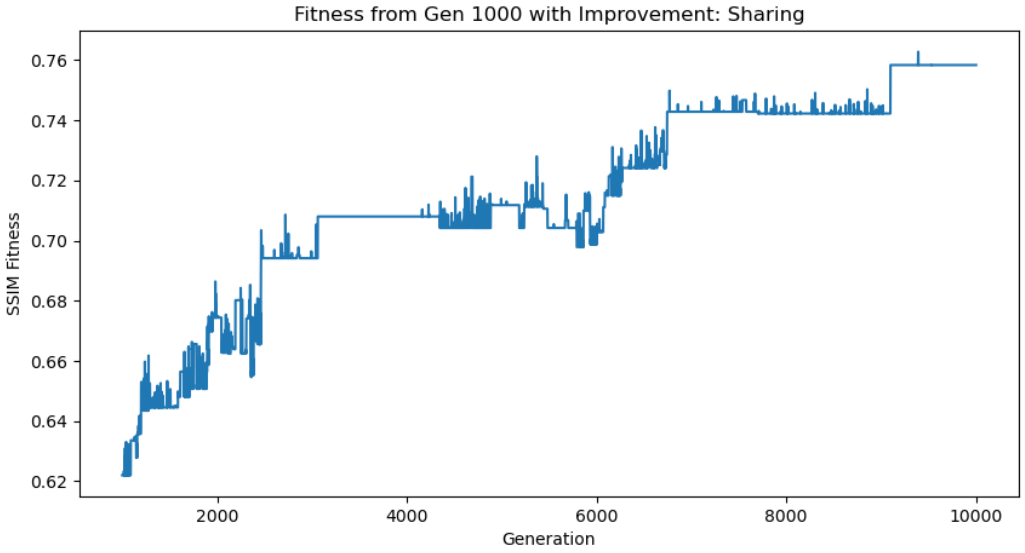

Fitness from Gen 1000 (num_inds=5)
SSIM Fitness
0.62
0.60
0.58
0.56
0.54
0.52
0.50
0.48
0.46
2000
4000
6000
8000
10000
Generation

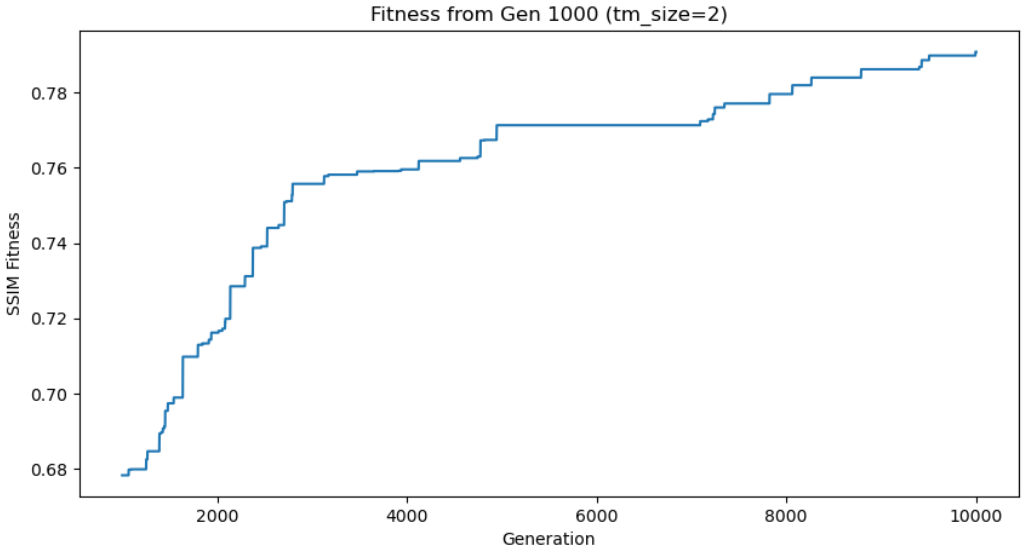

Fitness from Gen 1000 (num_inds=10)
SSIM Fitness
0.76
0.74
0.72
0.70
0.68
0.66
2000
4000
6000
8000
10000
Generation

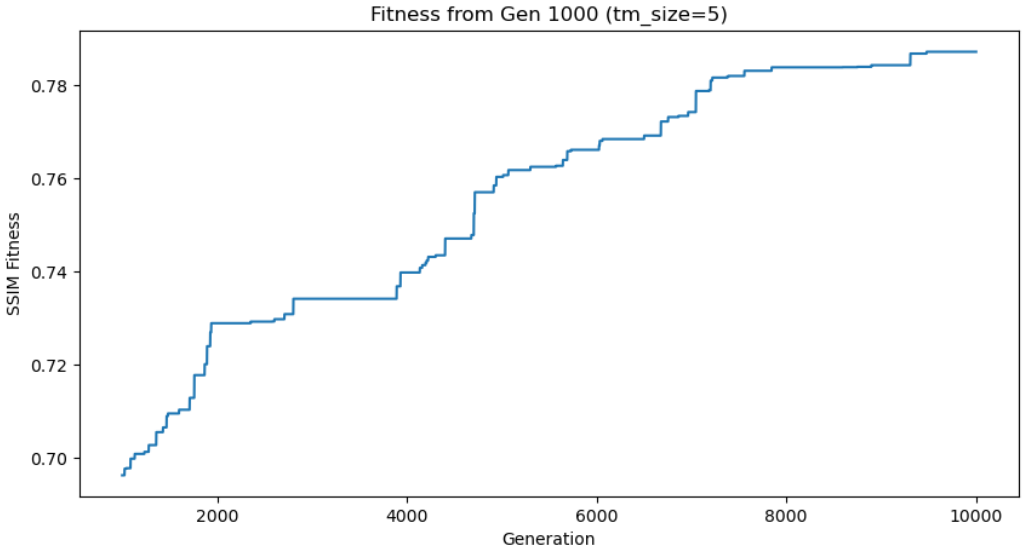

Fitness from Gen 1000 (num_inds=20)
SSIM Fitness
0.79
0.78
0.77
0.76
0.75
0.74
0.73
0.72
0.71
2000
4000
6000
8000
10000
Generation

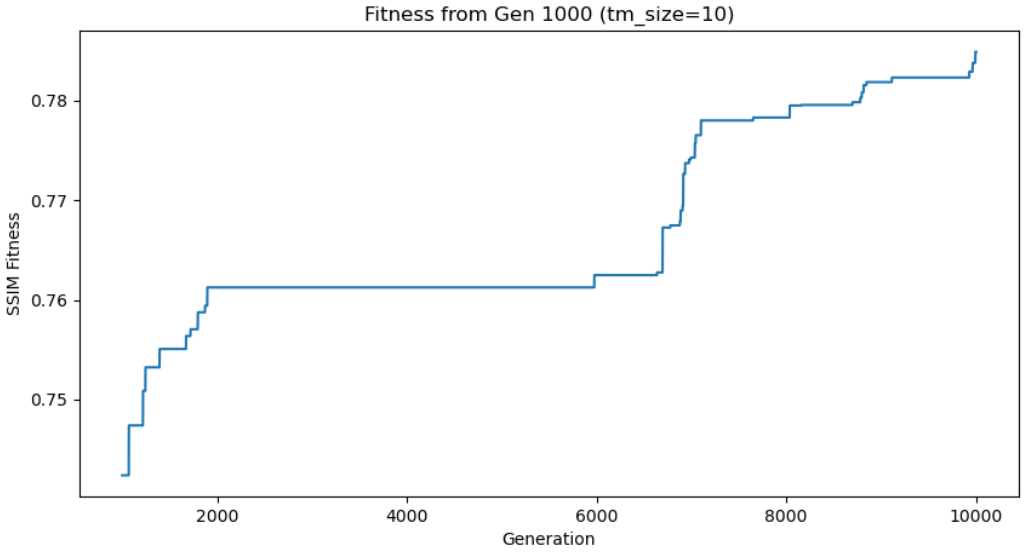

Fitness from Gen 1000 (num_inds=50)
SSIM Fitness
0.80
0.79
0.78
0.77
0.76
0.75
0.74
2000
4000
6000
8000
10000
Generation


Fitness from Gen 1000 (num_inds=75)
SSIM Fitness
0.82
0.81
0.80
0.79
0.78
2000
4000
6000
8000
10000
Generation


Fitness from Gen 1000 with Improvement: Sharing
SSIM Fitness
0.76
0.74
0.72
0.70
0.68
0.66
0.64
0.62
2000
4000
6000
8000
10000
Generation

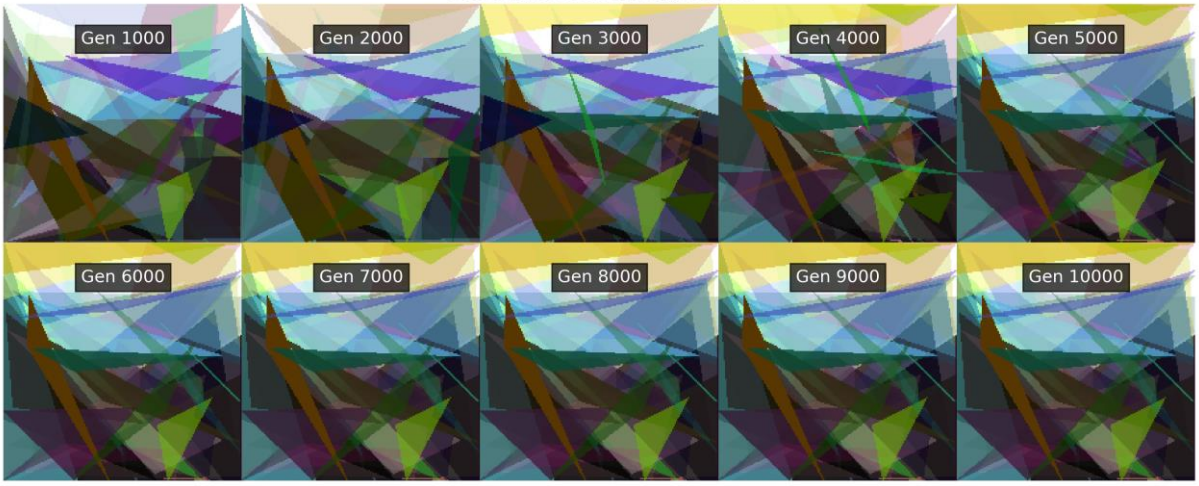

Fitness from Gen 1000 (tm_size=2)
SSIM Fitness
0.78
0.76
0.74
0.72
0.70
0.68
2000
4000
6000
8000
10000
Generation



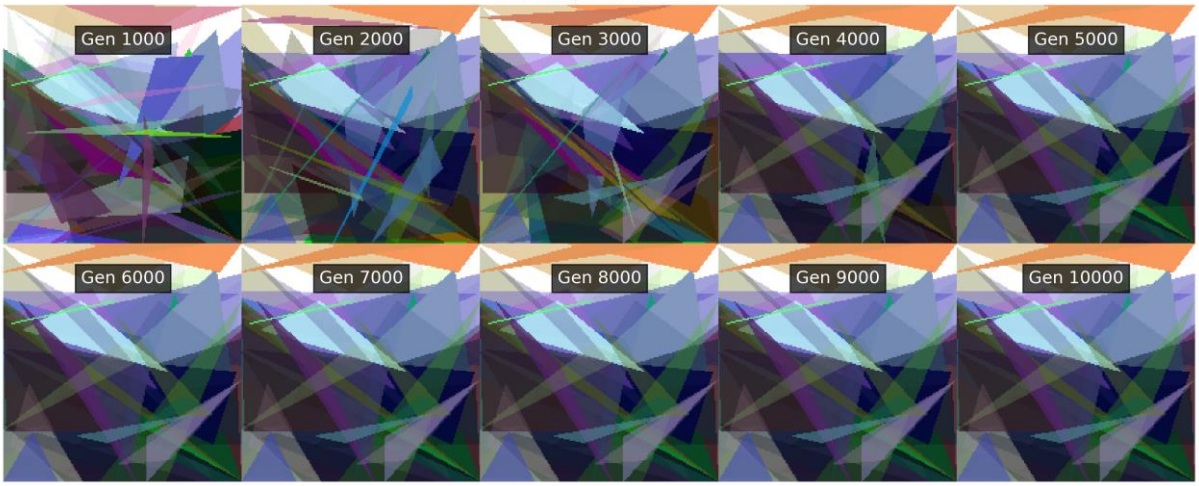


C. OUTPUT IMAGES

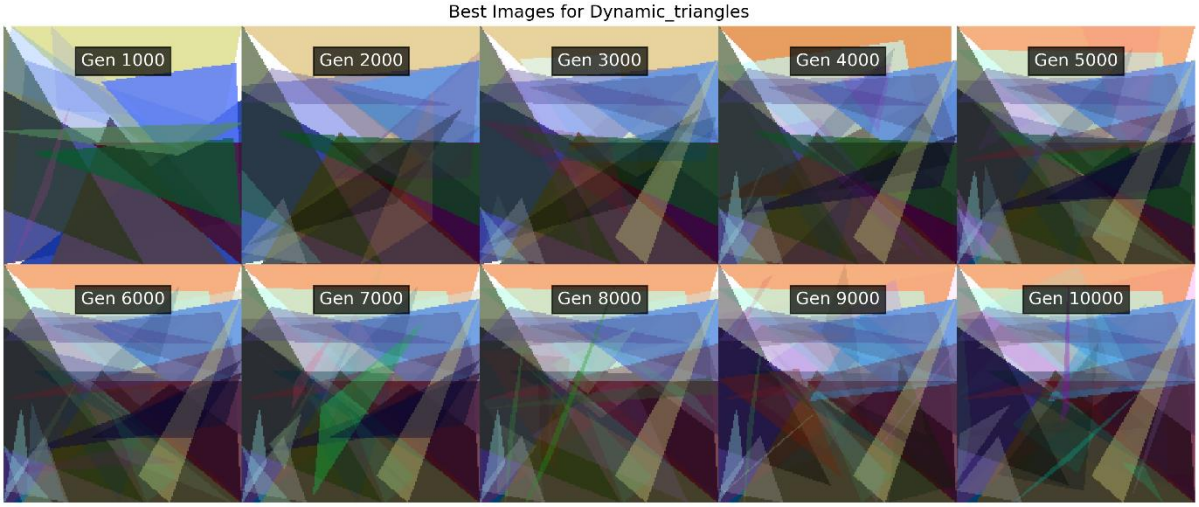
Best Images for Adaptive_mutation



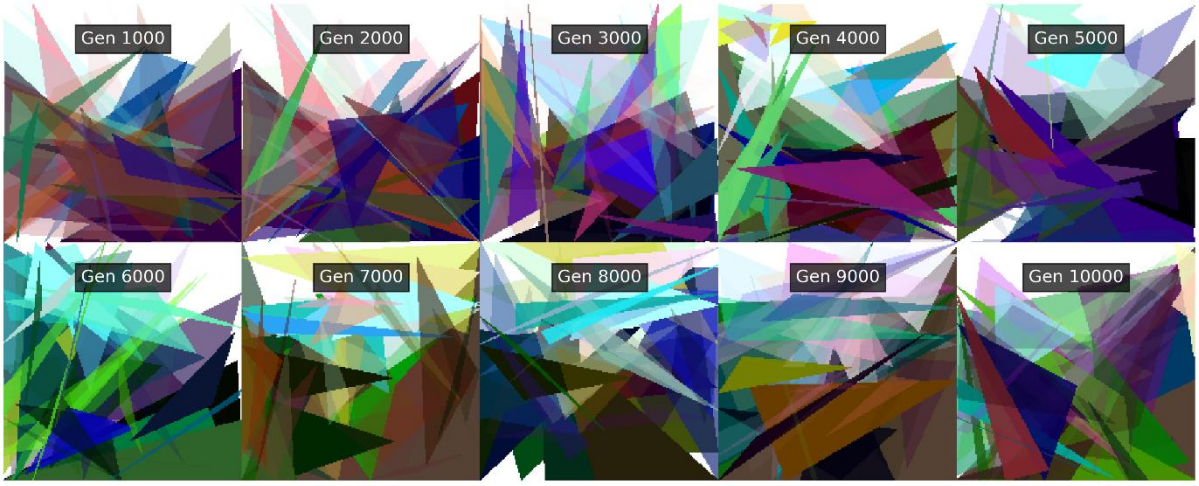
Best Images for Baseline



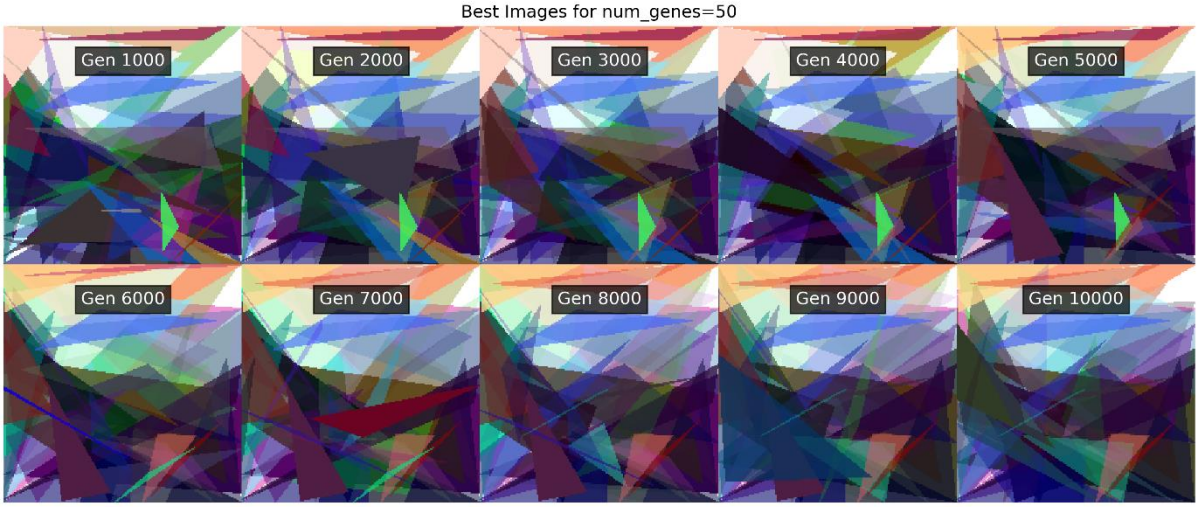
Best Images for Dynamic_triangles



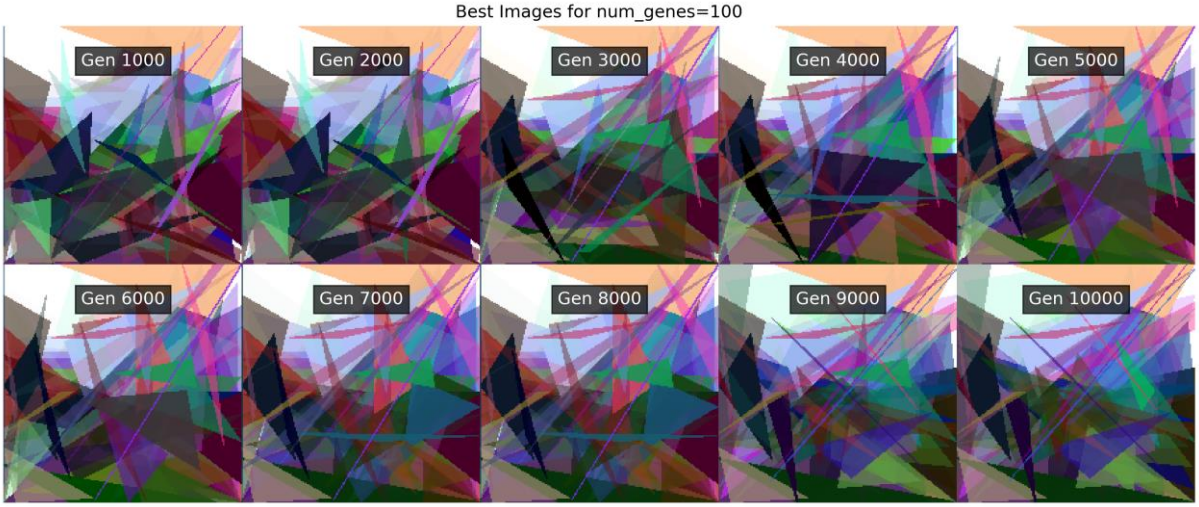
Best Images for Fitness_sharing



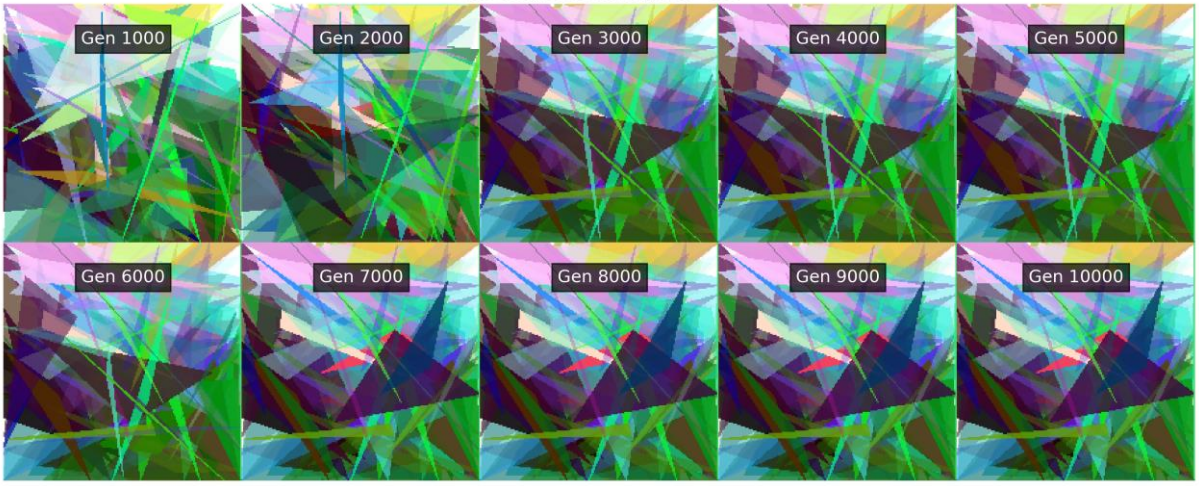
Best Images for num_genes=50



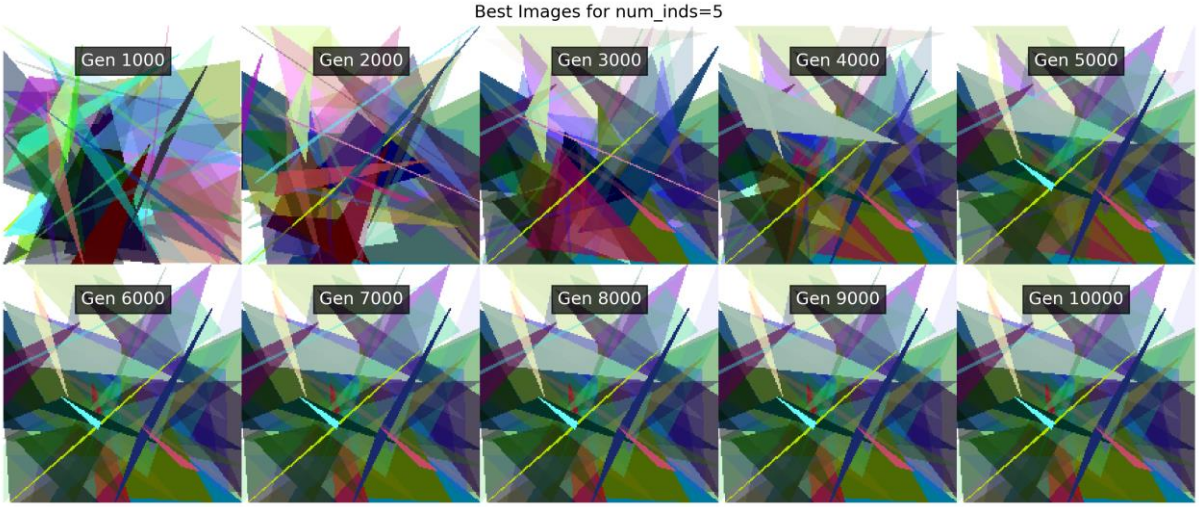
Best Images for num_genes=100



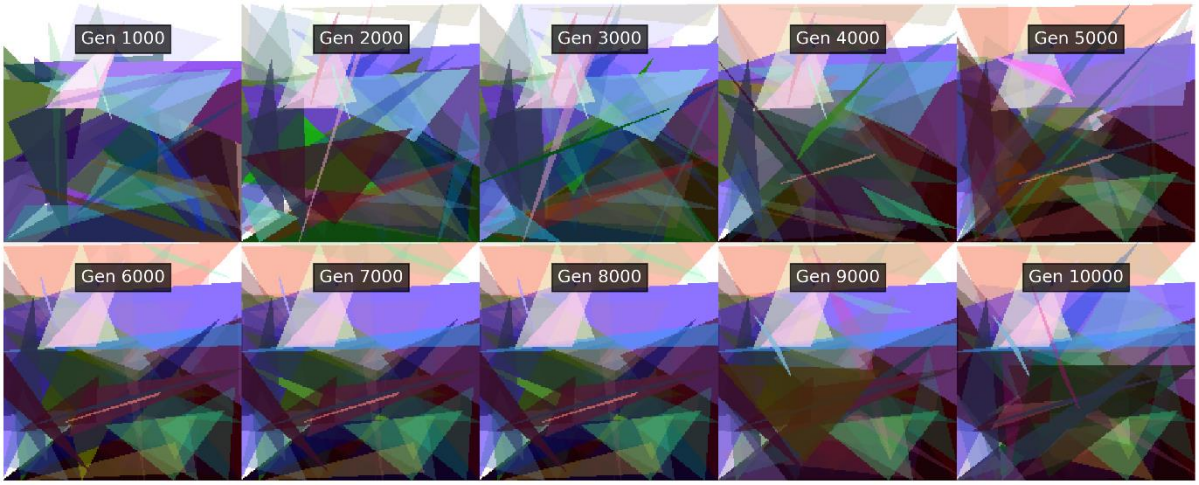
Best Images for num_genes=150



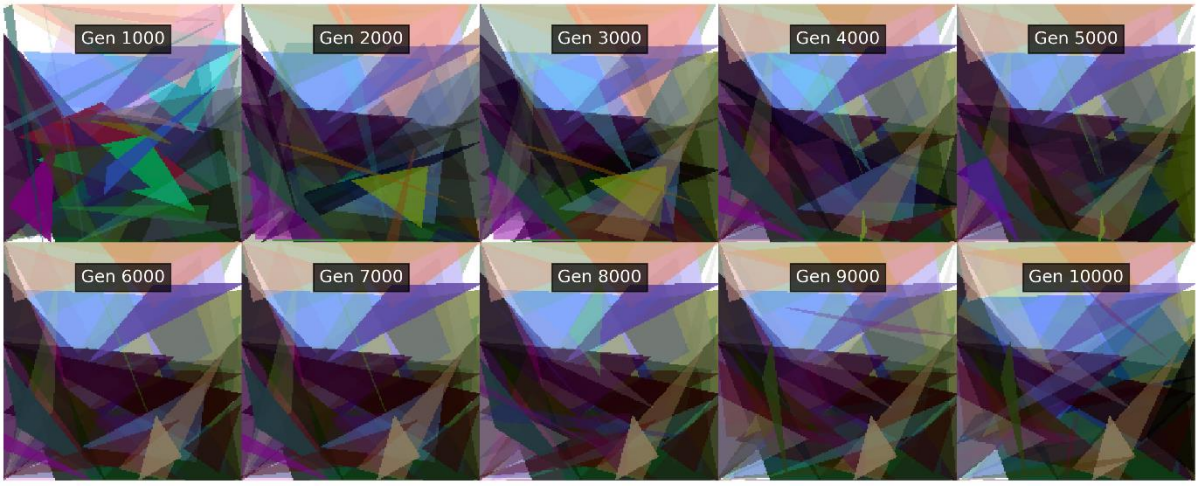
Best Images for num_inds=5



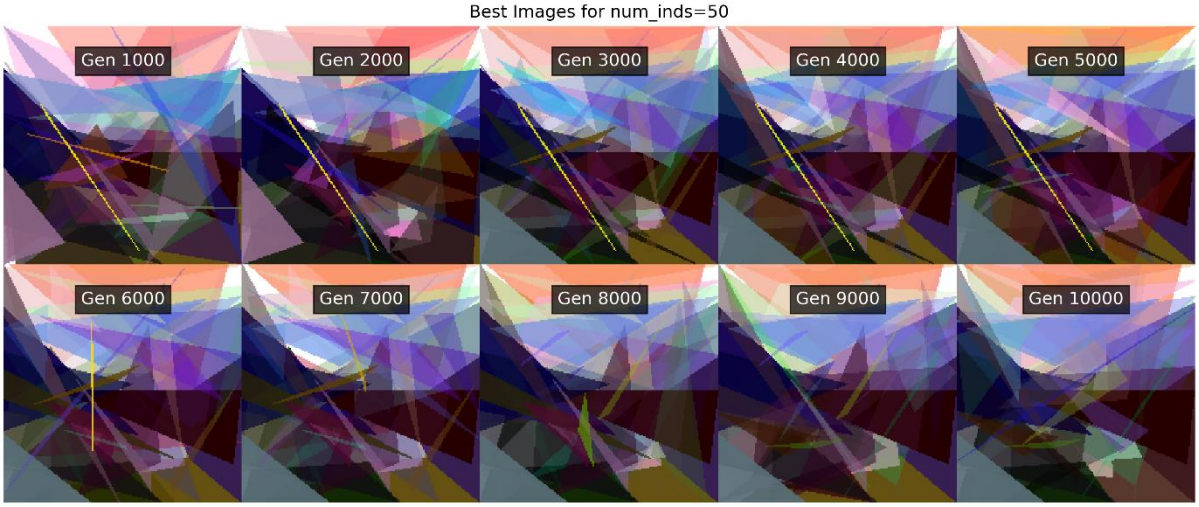
Best Images for num_inds=10



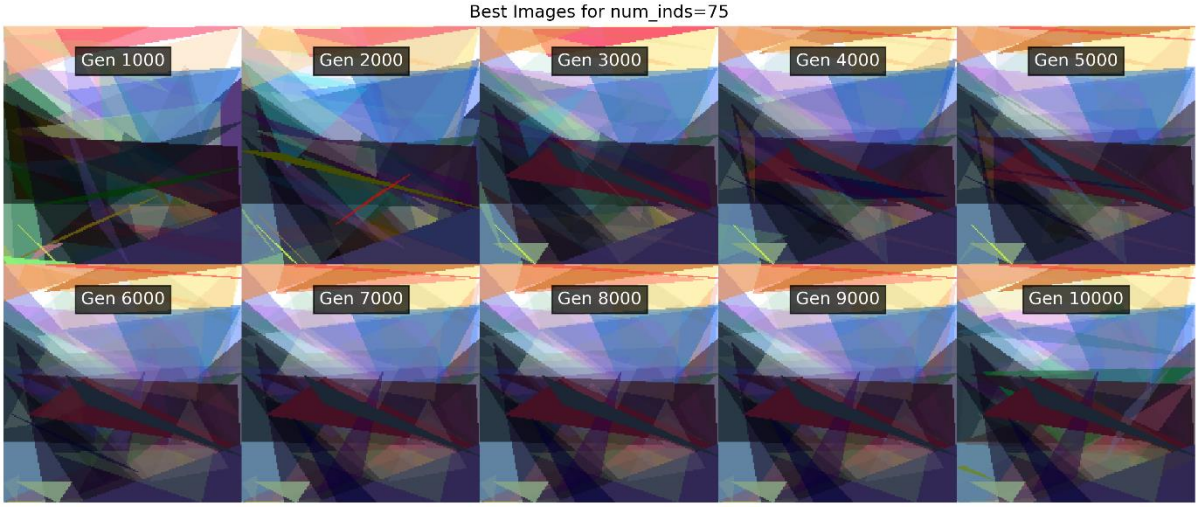
Best Images for num_inds=20



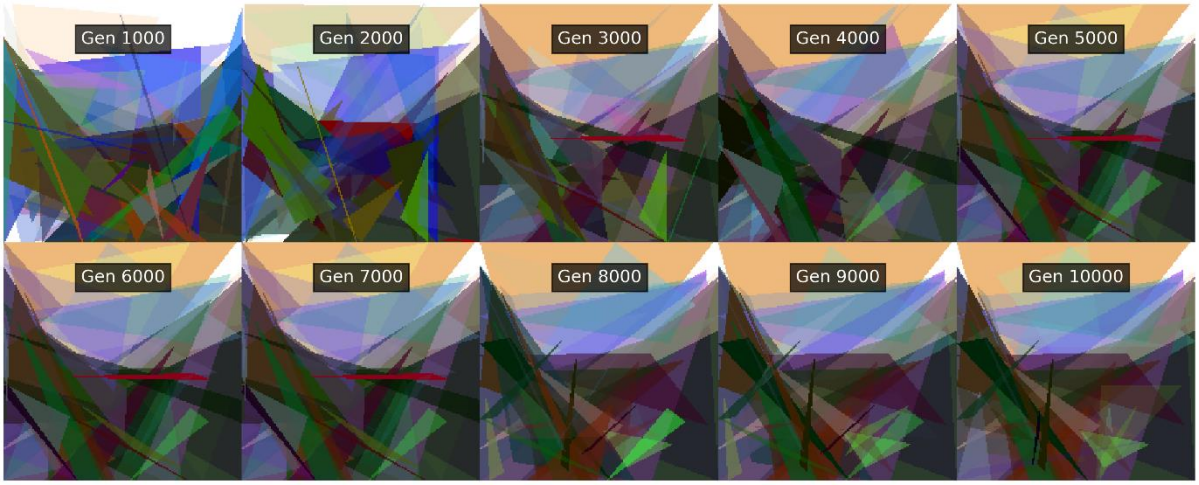
Best Images for num_inds=50



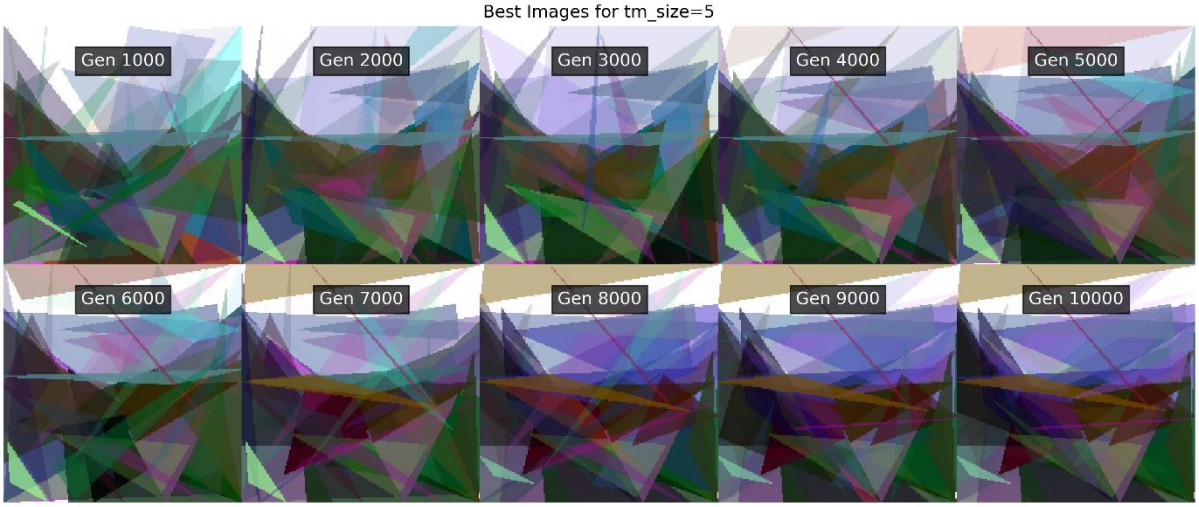
Best Images for num_inds=75



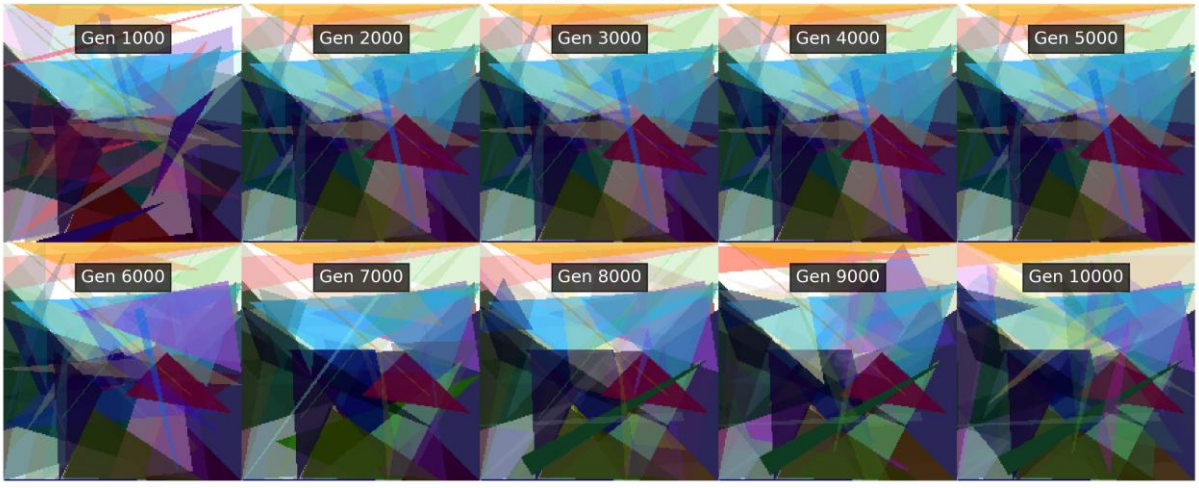
Best Images for tm_size=2



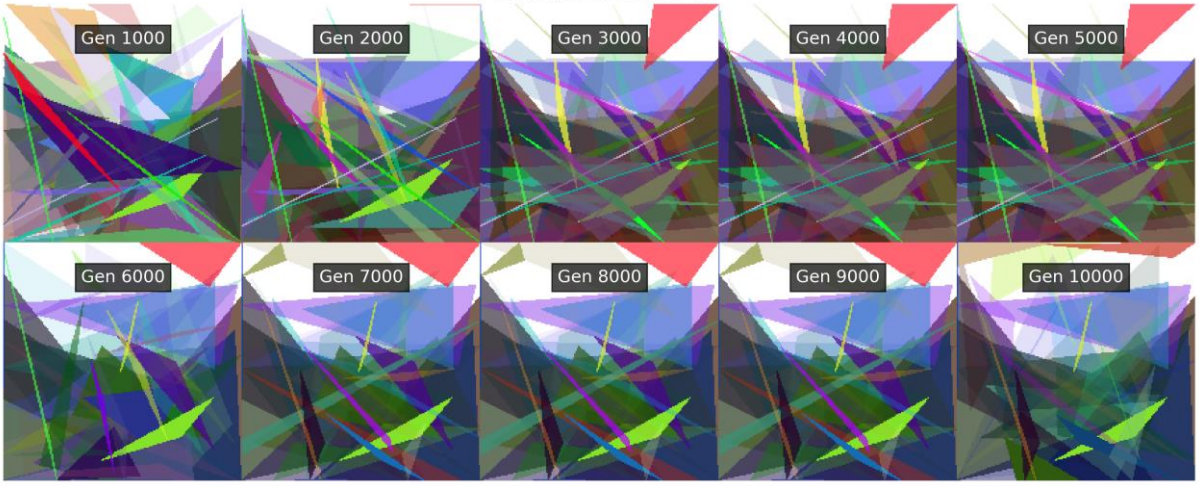
Best Images for tm_size=5



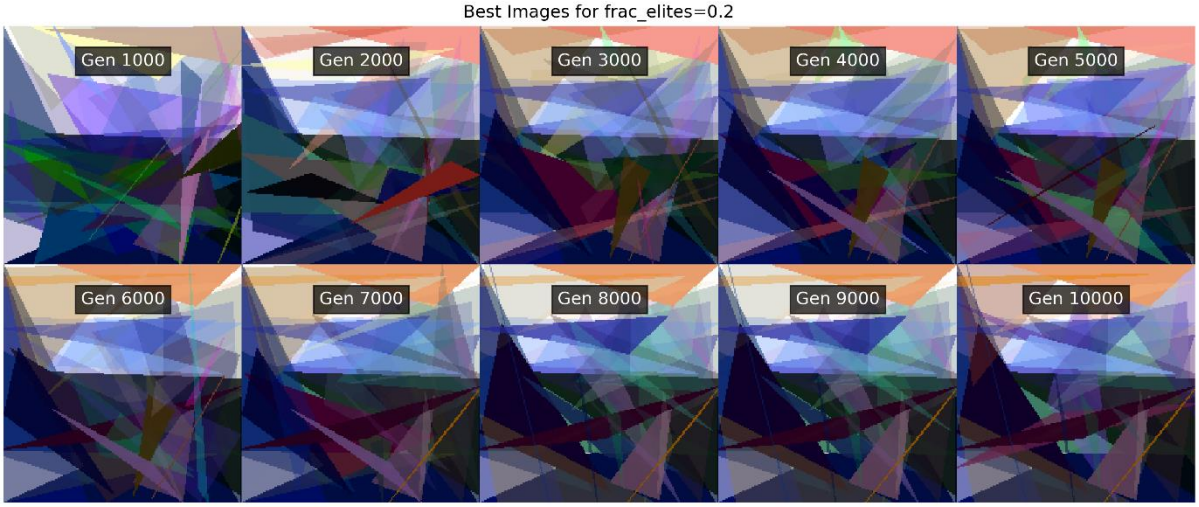
Best Images for tm_size=10



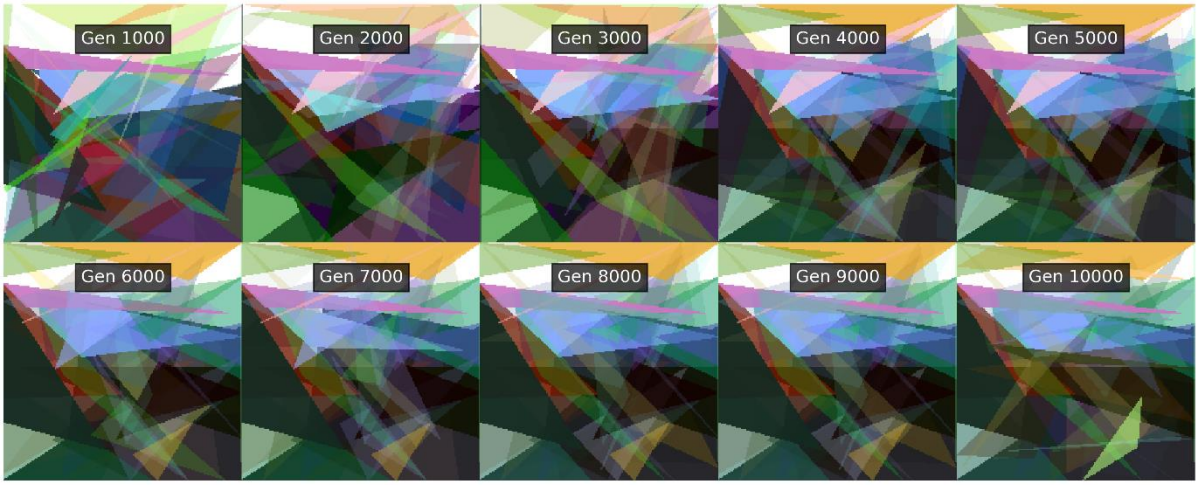
Best Images for tm_size=20



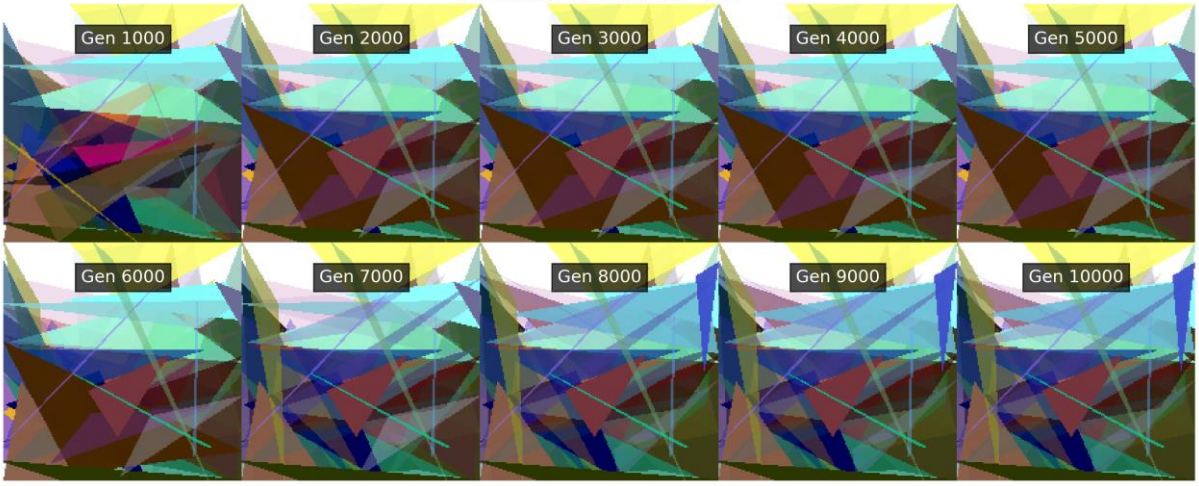
Best Images for frac_elites=0.2



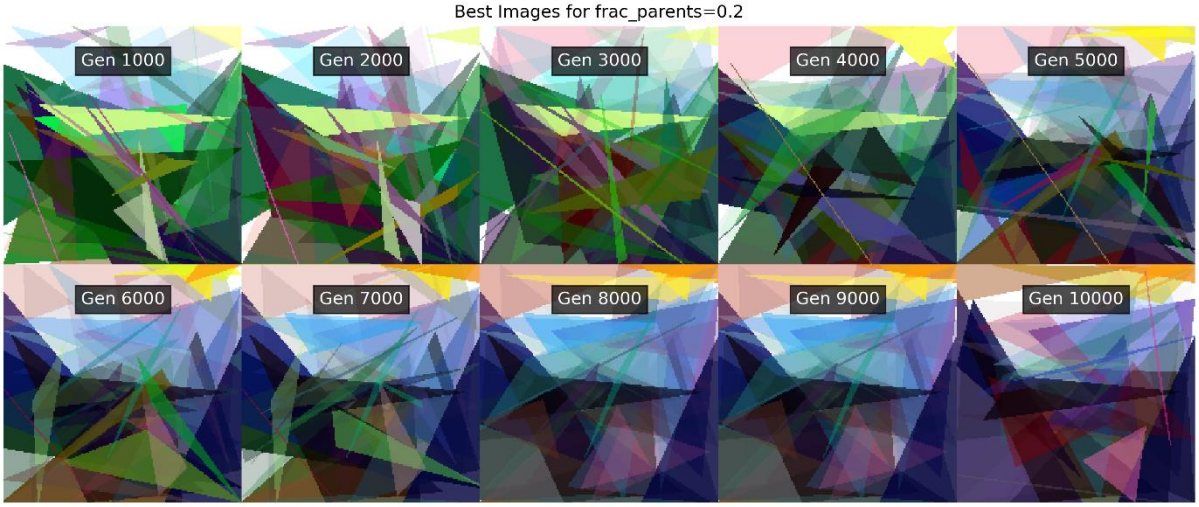
Best Images for frac_elites=0.4



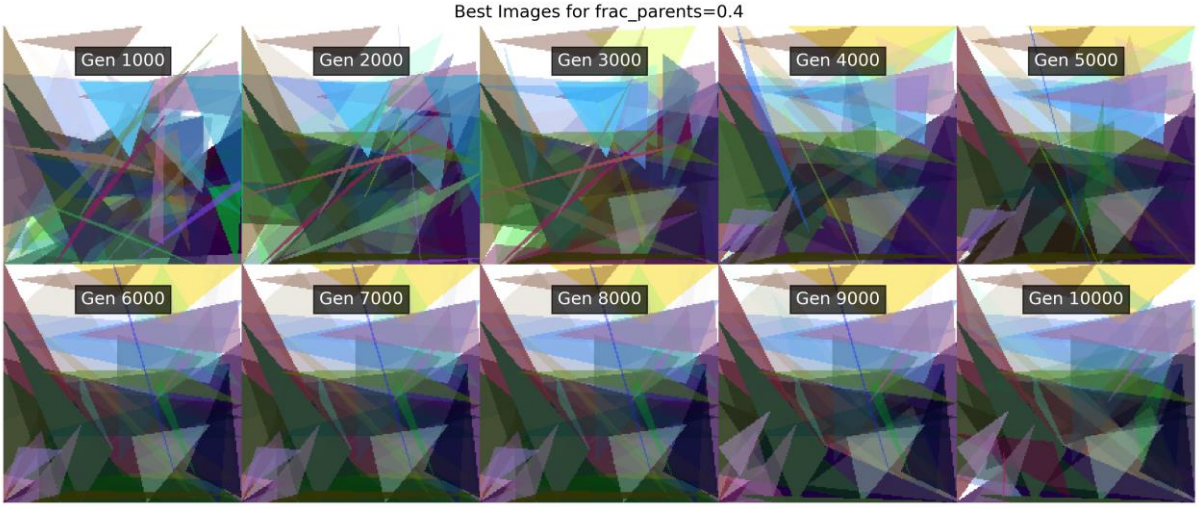
Best Images for frac_elites=0.05



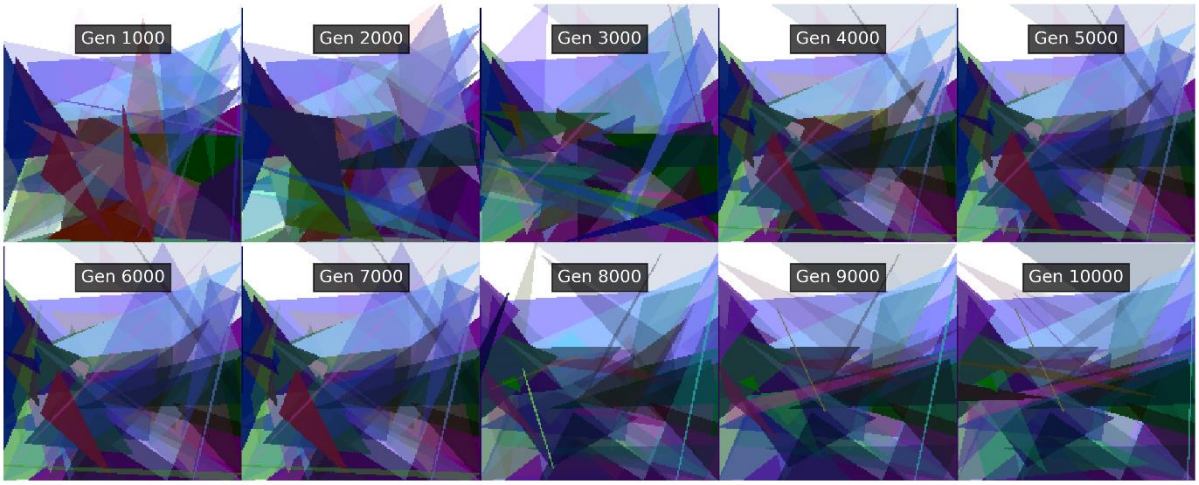
Best Images for frac_parents=0.2



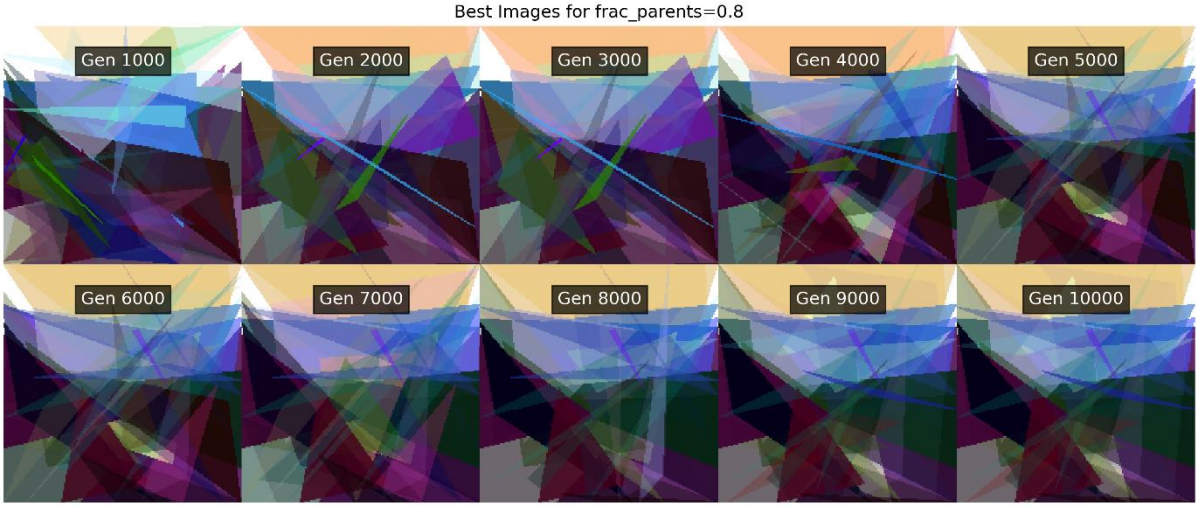
Best Images for frac_parents=0.4



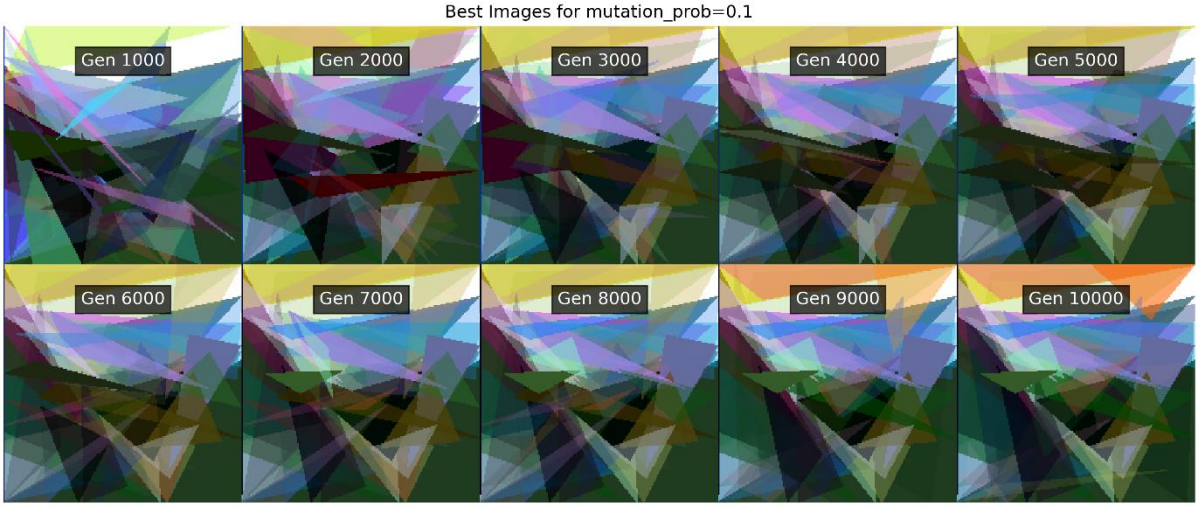
Best Images for frac_parents=0.6



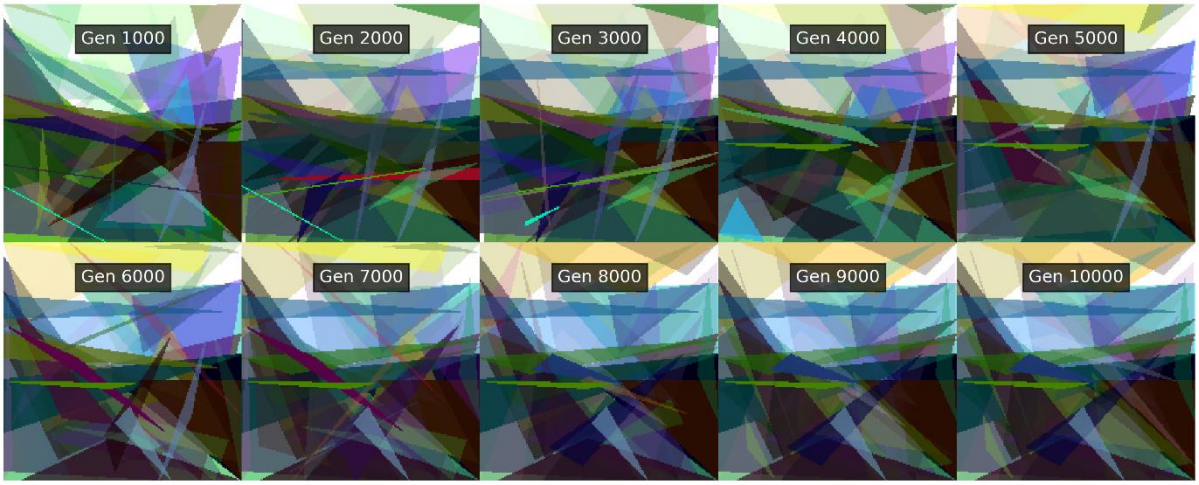
Best Images for frac_parents=0.8



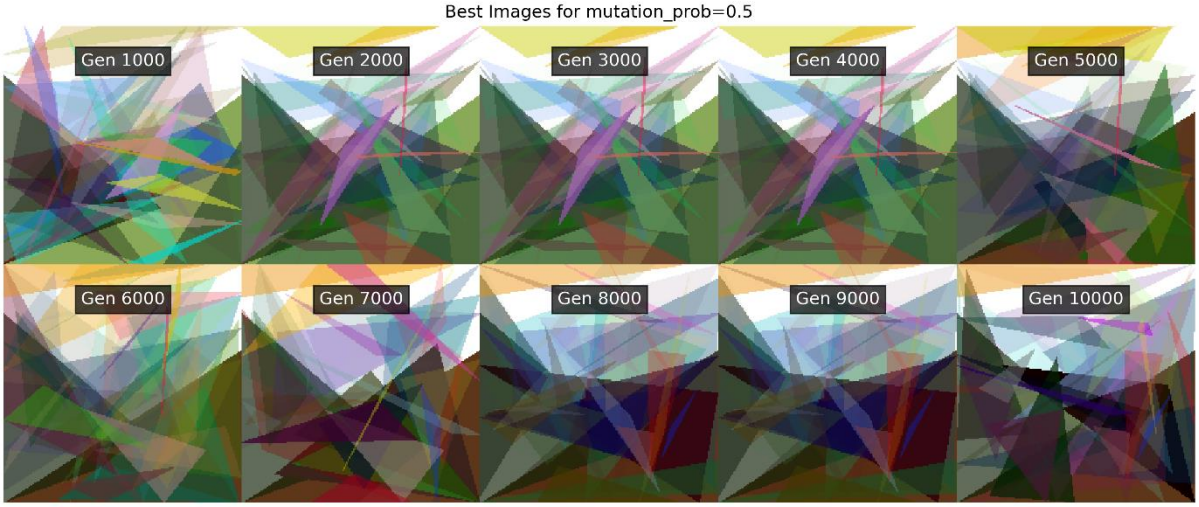
Best Images for mutation_prob=0.1



Best Images for mutation_prob=0.2

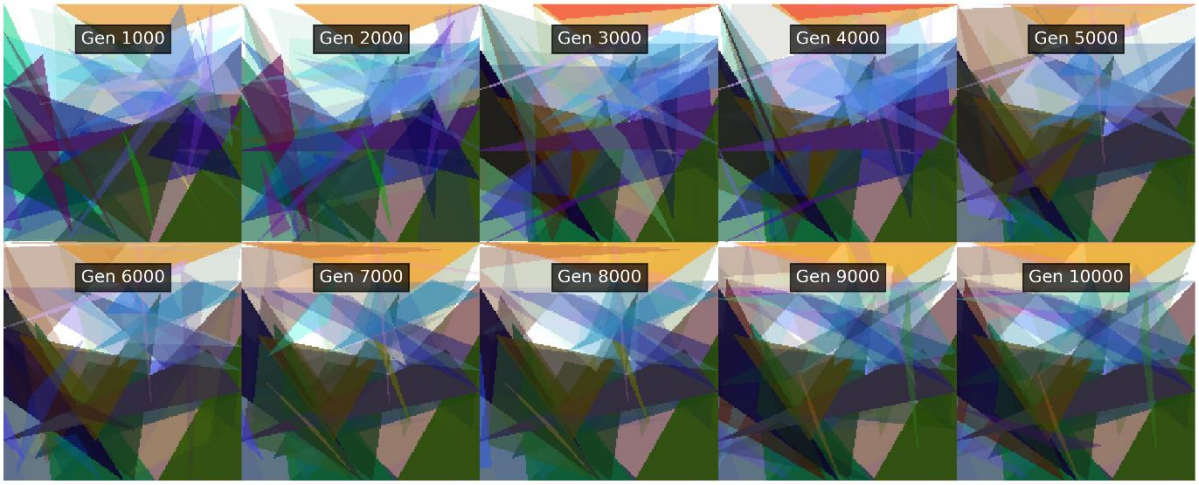


Best Images for mutation_prob=0.5

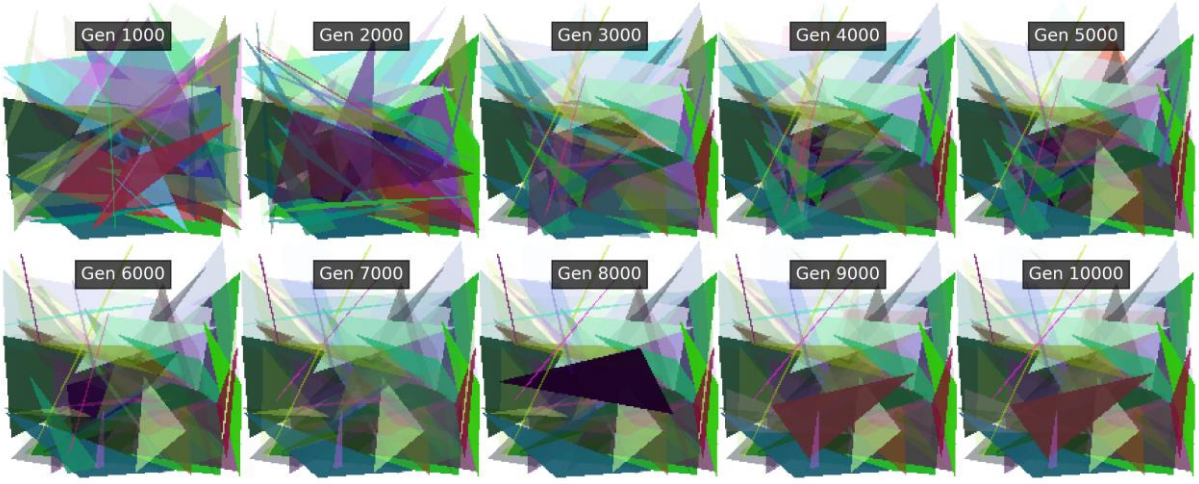


Best Images for mutation_prob=0.8

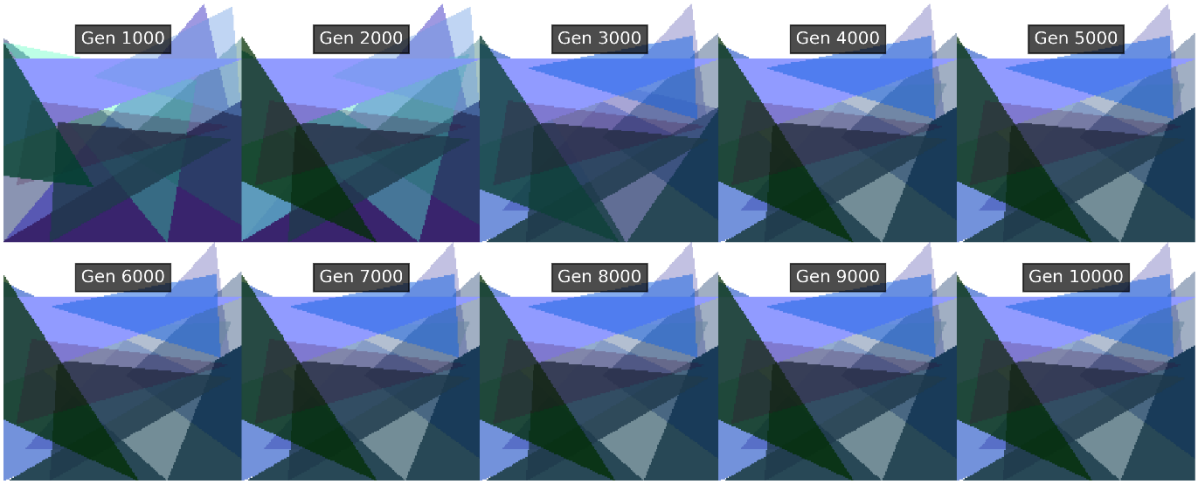
Best Images for mutation_type=guided



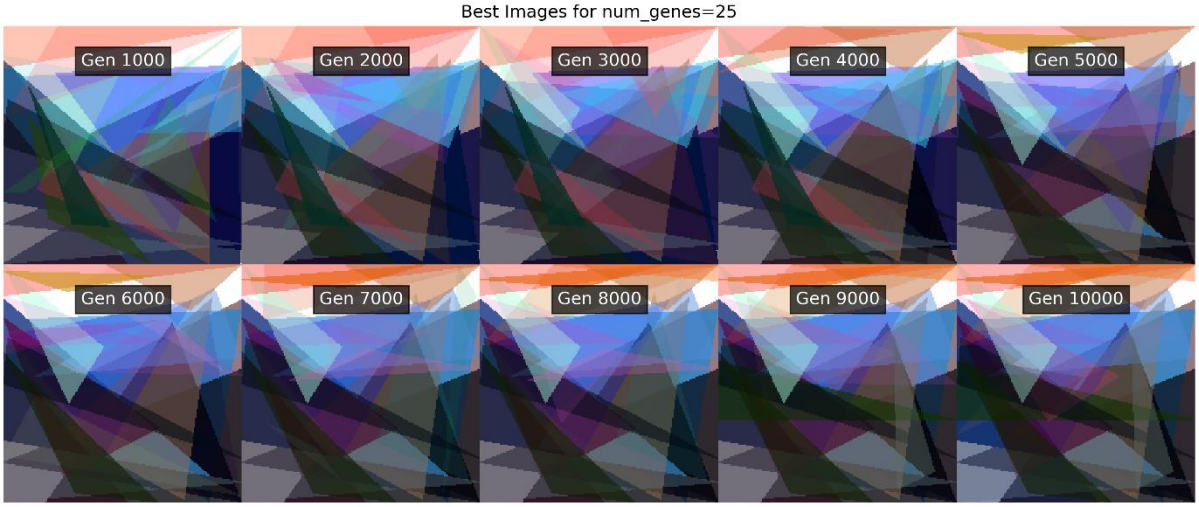
Best Images for mutation_type=unguided



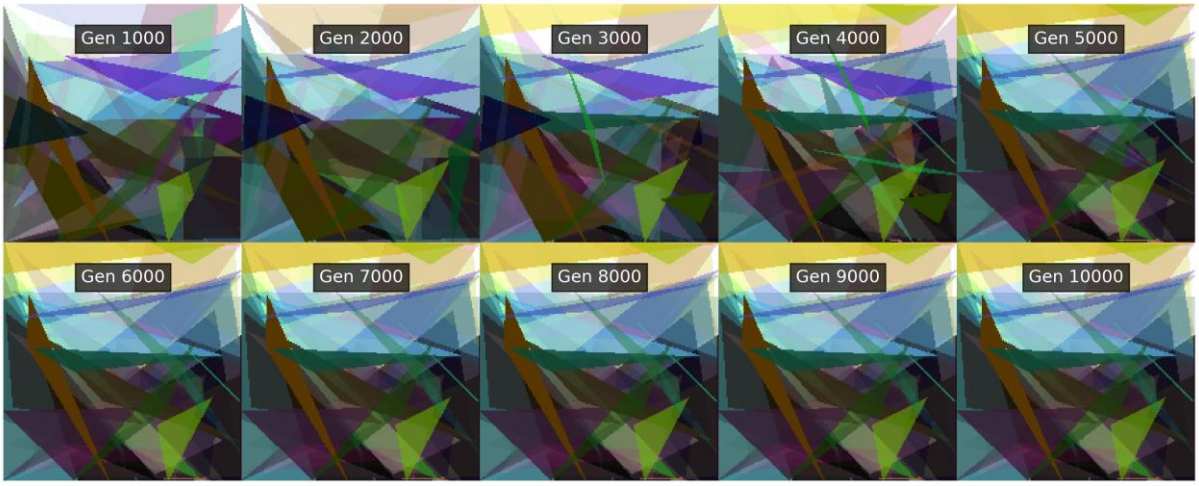
Best Images for num_genes=10



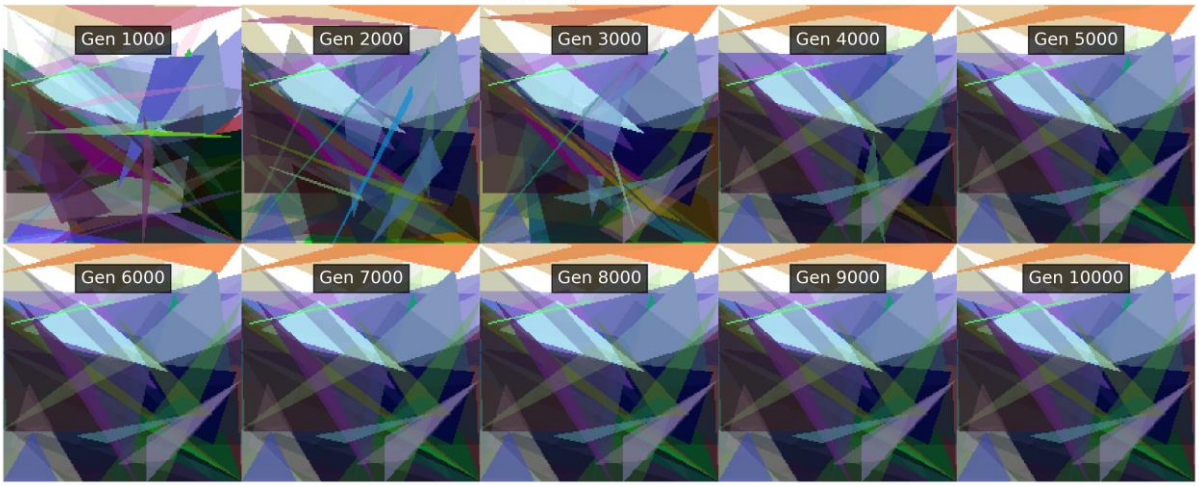
Best Images for num_genes=25



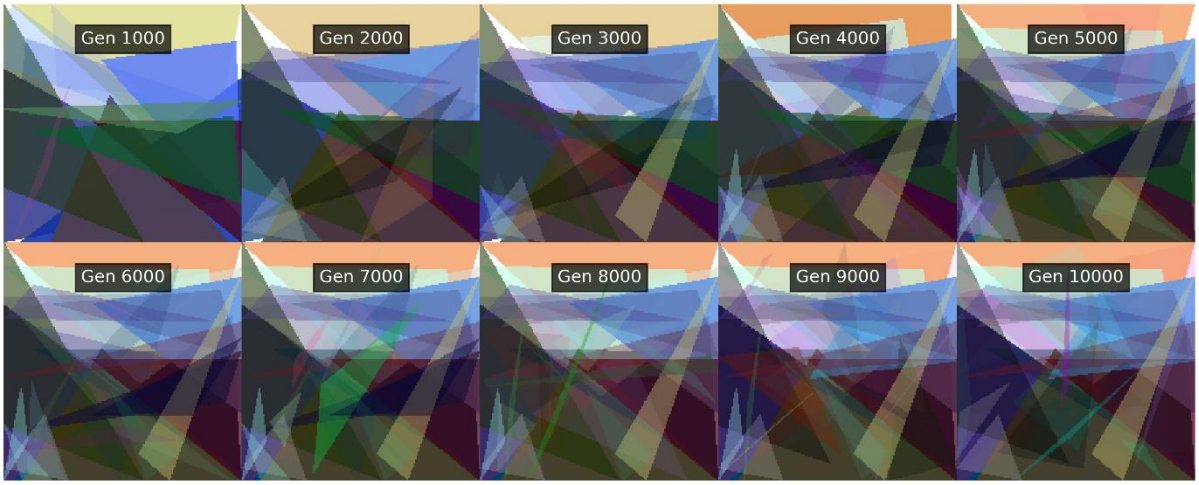
Best Images for Adaptive_mutation



Best Images for Baseline



Best Images for Dynamic_triangles



Best Images for Fitness_sharing