Computational Intelligence Assignment I

Syed Mustafa sm06554
Zain Ahmed Usmani Zu06777

February 15, 2024

1 Travelling Salesman Problem (TSP)

1.1 Problem Formulation

TSP was implemented in almost the same manner as explored in the lectures.

1.1.1 Representation

The data is sourced from National Traveling Salesman Problems (uwaterloo.ca) and, in our implementation, is a .tsp file which is read by the dataloader of the TSP class. Chromosome was a permutation of the list of all cities (Nodes) in Qatar. Initialized as a random numpy list of these cities' IDs. A list of distances between each city is kept and used to make calculations for a fitness list which corresponds 1-1 with the city ID.

1.1.2 Fitness

Fitness is calculated as the negative of Euclidean distance between two points. The lower the distance, the higher the fitness.

1.1.3 Crossover

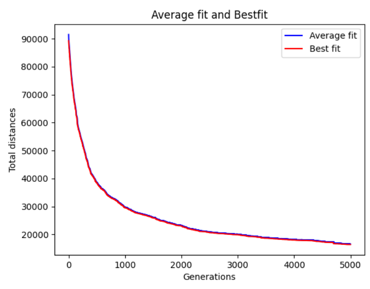
Order 1 Crossover was implemented based on the lecture material, where an arbitrary part of the first parent is added to the child, and the rest of the chromosome from the second parent is adjusted into the list in order.

1.1.4 Mutation

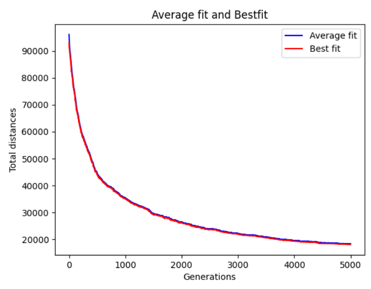
Insert Mutation Picks a city in the chromosome and inserts it at a random point in the list. **Swap Mutation** Picks two cities and swaps their position.

1.1.5 Result

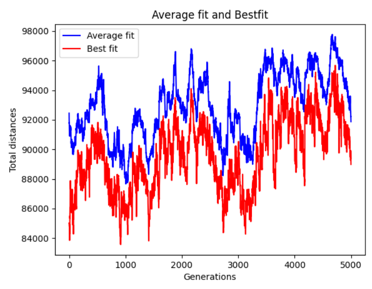
Best score achieved: 12,718 on rank-based survival and selection, with a population of 100 and 0.5 mutation rate, over 5000 generations. The tests below were also conducted with the same parameters. Figures 1a and 1b demonstrate convergence of the algorithm normally. 1f also manages to converge, albeit in an unusually proportional manner.
For 1c and 1d however we observe what seems to be random behavior. It is suspected that this might be because of some inexplicable issues with fitness proportional sampling in our implementation. It is noteworthy that the range in variation in 1d is small, so the average fitness has not changed too much.
Additionally randomness in 1e comes from the random selection schemes. In hindsight. it would have



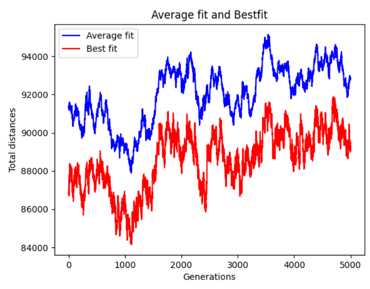
(a) Binary tournament (parent) and Truncation (survival). Fittest: 16410



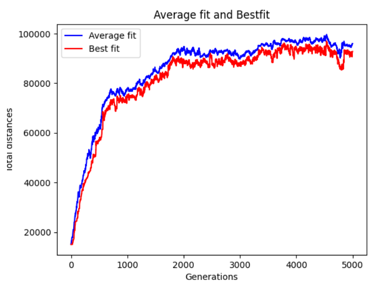
(b) Fitness proportional selection (parent) and rank-based selection (survival). Fittest: 18164



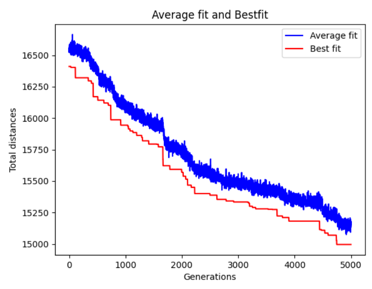
(c) Fitness proportional (parent) and random (survival). Fittest: 83576



(d) Fitness proportional (parent) and random (survival) (2). Fittest: NA



(e) Random (parent) and random (survival). Fittest: 14996



(f) Truncation selection (parent) and truncation selection (survival). Fitness: 14996.

Figure 1: Graphs generated with different schemes for parent and survival selection. Configured for 5000 generations, with a population size of 100 and mutation rate of 0.5.
Note: Instead of implementing the best fitness so far, we have implemented the best fitness per generation and then reported the overall best fitness in all iterations.

been better to implement these schemes such that a child is only added to the population if its fitness is greater.

2 Job Shop Scheduling Problem

2.1 Problem Formulation

Much of JSSP was implemented in a manner similar to TSP with a few implementation differences in representation and correspondingly crossover, mutation and fitness.

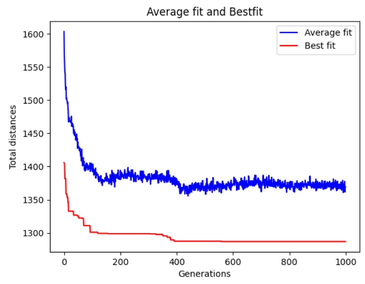
2.1.1 Representation

Chromosome is a sequence (list) containing integers. The integers correspond to a job J_i. The first instance of the integer corresponds to the first operation for that job, the second instance correspond to the second operation/process in that job, and so on.

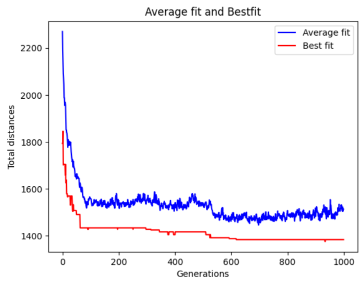
2.1.2 Crossover, Mutation, Fitness

The implementation for EA for this is essentially the same as in TSP, only with a few adjustments to ensure compatibility. Fitness is measured by the time taken by the machines, and is also taken to be the negative of this time, so the overall process tries to minimize this value.

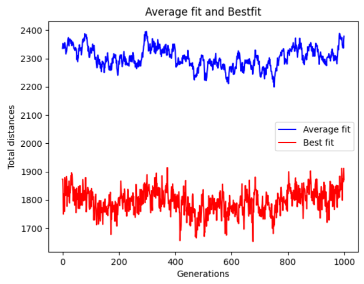
2.2 Results



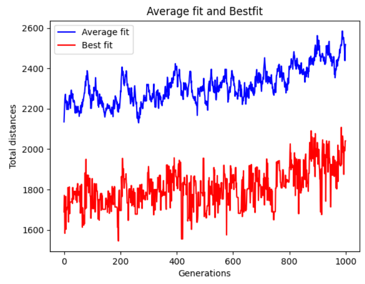
(a) Binary tournament (parent) and Truncation (survival). Fittest: 1287



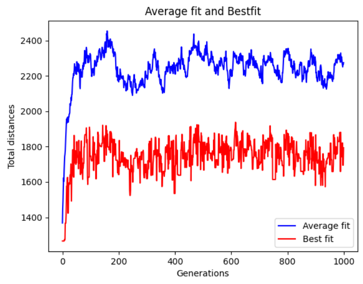
(b) Fitness proportional selection (parent) and rank-based selection (survival). Fittest: 1376



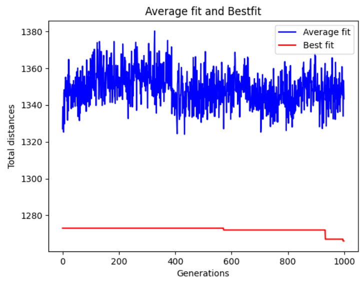
(c) Fitness proportional (parent) and random (survival). Fittest: 1444



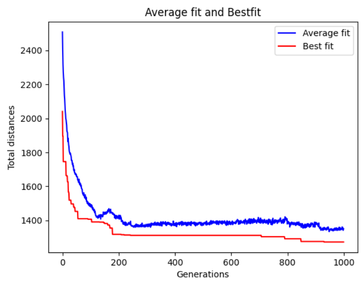
(d) Fitness proportional (parent) and random (survival) (2). Fittest: 1545



(e) Random (parent) and random (survival). Fittest: 1266



(f) Truncation selection (parent) and truncation selection (survival). Fitness: 1266.



(g) Binary tournament selection (parent) and truncation selection (survival). Fitness: 1266.

Figure 2: Graphs generated with different selection schemes for JSSP. **Note:** Instead of implementing the best fitness so far, we have implemented the best fitness per generation and then reported the overall best fitness in all iterations.

1a, 1b and 1g show convergence. Any graphs with random selection schemes fail to converge. In 1f truncation selection schemes should ideally lead to convergence, but somehow fail to do so, maybe because the average population was initialized poorly, and one initial solution was close to being perfect.

3 Evolutionary Art

3.1 Problem Formulation

The problem can be summarized as generating an image through a source image using polygons of varying colors using evolutionary algorithms such that the resulting image is as close to the source image in resemblance.

3.1.1 Representation

The source image was read and handled as RGB images, and hence the chromosomes that would form the solution were also RGB images. To initialize a chromosome, a central point was defined at a randomly chosen point in the image range. Around this point at least 3 more points were added within a random region range. These points were then filled with a randomly chosen flat color. A variety of such shapes were drawn over the same canvas and the final RGB values at each pixel were stored in an array. This array forms the chromosome. Generally, throughout our experimentation we have kept about 100 different chromosomes in our population.

3.1.2 Fitness function

We experimented with two different fitness functions. One of these was just the mean of differences in RGB values for all pixels in the image. The other approach used Delta E color difference to differentiate between the two. We used CIE1976 implementation of the delta E function. This resulted in significantly better results (as shown through sections 3.3.0.1 and 3.3.0.1 at the very end). **Other parameters:** generations, population size.

3.1.3 Polygon Sides

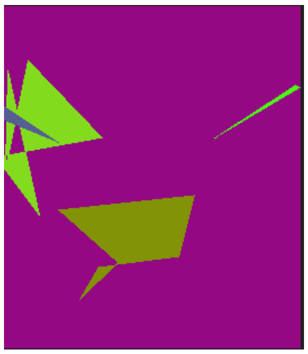


Figure 3: Initialized image with polygons having up to 6 sides.

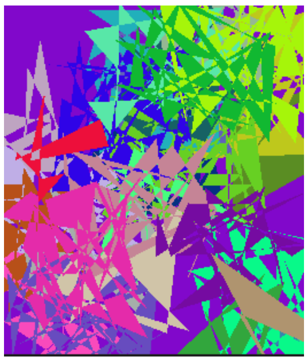


Figure 4: Initialized image with polygons having up to 50 sides.

We hypothesized that custom polygons would result in better solutions, this change it appears does not make a stark difference in fitness achieved after 500 generations, however, visually the one with 50 polygons ended up looking a lot more like the final image than the other (see figure 8).

3.2 Image size

Using a smaller image size allowed the simulation to run faster and converge earlier.

3.2.1 Crossover Functions

Random Crossover: The most simple crossover I implemented was a random crossover where sets of random pixels are selected from the two parents at random and put them together to create a child (essentially the same function used in Q1). The crossover used for testing however went a step further and multiplied together the randomly selected pixels. This sort of overlays/merges the values from the two parents together into an offspring.

Random Crossover with Blend In [2], Charmot does the same as above however, also ends up blending the two images together using a blend function. This worked much better in terms of producing an image that incrementally converged well.

3.2.2 Mutation Functions

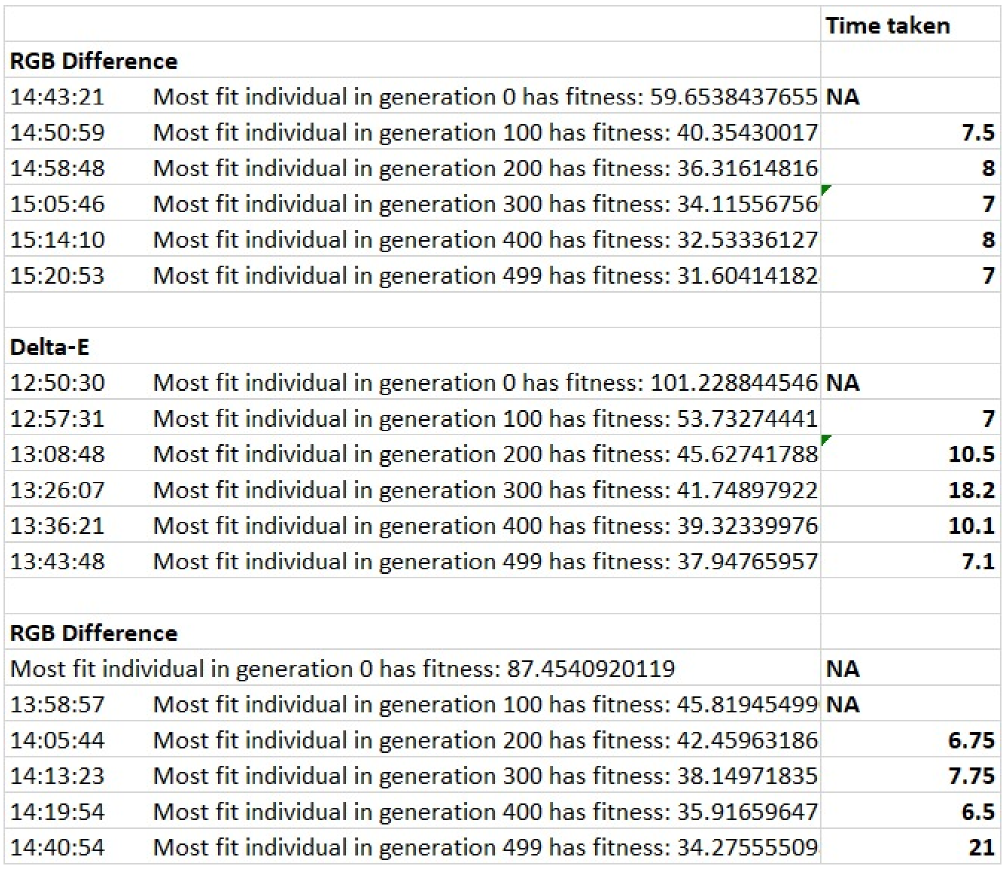
Pixel-wise update A mutation that adds random new pixels into the image and hopes to improve the fitness.

Update shapes A mutation that introduces and overlays a new shape onto the image.

3.3 Results

While the testing could not be done across various generations due to limitations in time. I managed to test the above parameters for up to 500 generations. A few results are appended on the final pages this document (see sections 3.3.0.1 and 3.3.0.1).

- Population size = 100
- Generations = 500
- Mutation Rate = 0.3
- Crossover Rate = 0.7
- Offsprings = 100



		Time taken
RGB Difference		
14:43:21	Most fit individual in generation 0 has fitness: 59.6538437655	**NA**
14:50:59	Most fit individual in generation 100 has fitness: 40.35430017	**7.5**
14:58:48	Most fit individual in generation 200 has fitness: 36.31614816	**8**
15:05:46	Most fit individual in generation 300 has fitness: 34.11556756	**7**
15:14:10	Most fit individual in generation 400 has fitness: 32.53336127	**8**
15:20:53	Most fit individual in generation 499 has fitness: 31.60414182	**7**
Delta-E		
12:50:30	Most fit individual in generation 0 has fitness: 101.228844546	**NA**
12:57:31	Most fit individual in generation 100 has fitness: 53.73274441	**7**
13:08:48	Most fit individual in generation 200 has fitness: 45.62741788	**10.5**
13:26:07	Most fit individual in generation 300 has fitness: 41.74897922	**18.2**
13:36:21	Most fit individual in generation 400 has fitness: 39.32339976	**10.1**
13:43:48	Most fit individual in generation 499 has fitness: 37.94765957	**7.1**
RGB Difference		
Most fit individual in generation 0 has fitness: 87.4540920119		**NA**
13:58:57	Most fit individual in generation 100 has fitness: 45.81945499	**NA**
14:05:44	Most fit individual in generation 200 has fitness: 42.45963186	**6.75**
14:13:23	Most fit individual in generation 300 has fitness: 38.14971835	**7.75**
14:19:54	Most fit individual in generation 400 has fitness: 35.91659647	**6.5**
14:40:54	Most fit individual in generation 499 has fitness: 34.27555509	**21**

Figure 5: This image shows the best fitness across various generations. The configuration is as stated above. Only fitness functions were varied.

3.3.0.1 Comparing polygon resolution/number of sides

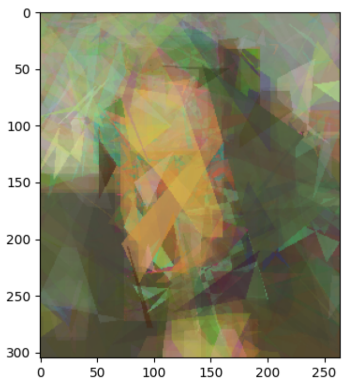


Figure 6: Mona Lisa with maximum polygon resolution at 50, after 500 iterations with a delta E difference (fitness) of 37.95.

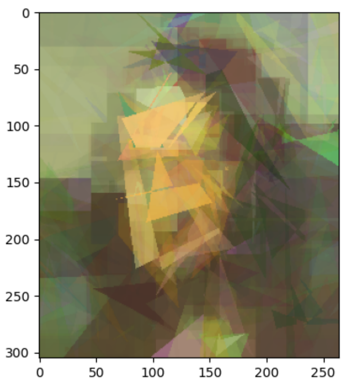
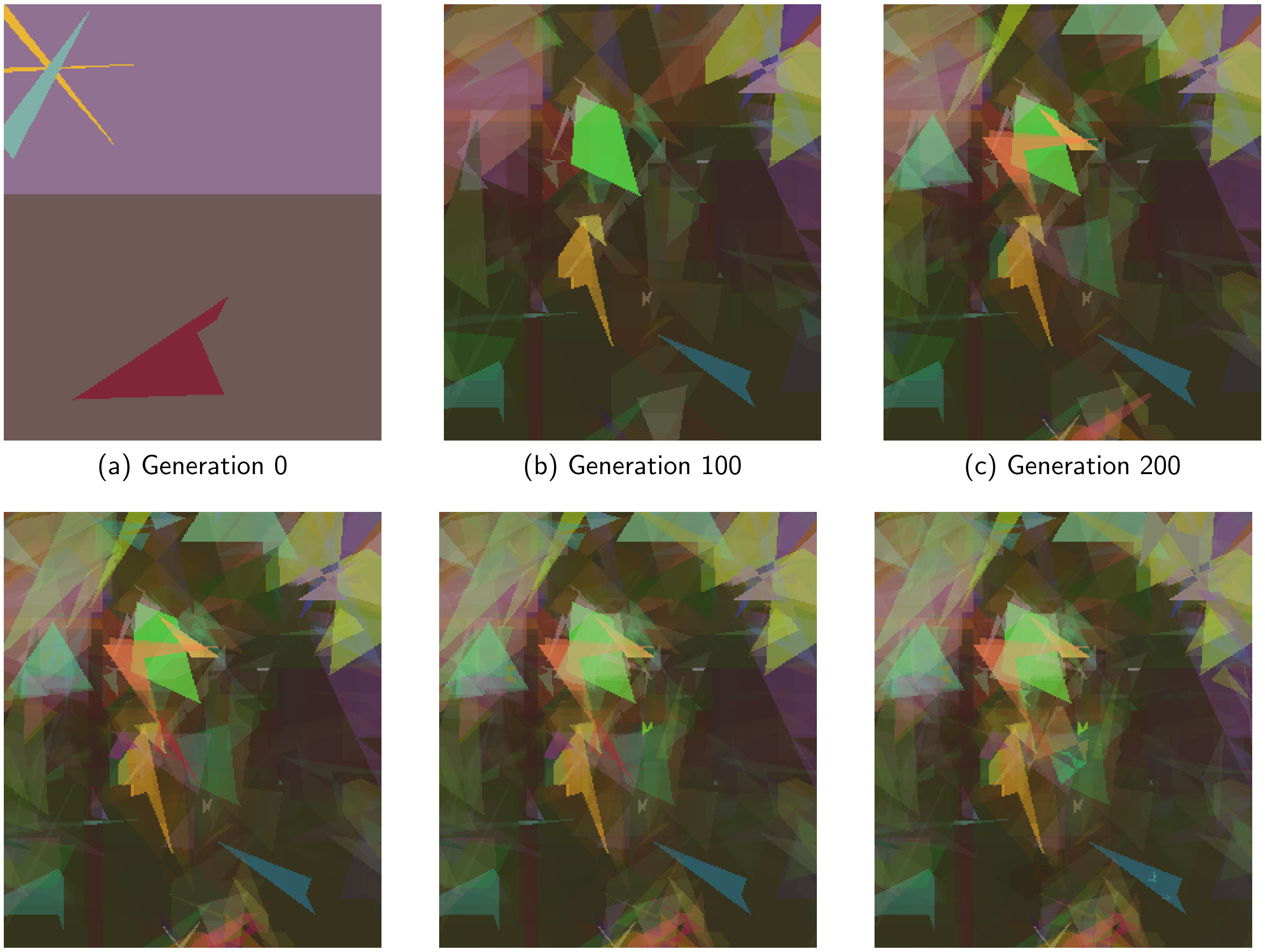


Figure 7: Mona Lisa with maximum polygon resolution at 6, after 500 iterations with a delta E difference (fitness) of 34.28.

Figure 8: While slight differences in image accuracy can't be compared without a vigilant eye or by pixel-peeping, more often than not the image with the higher polygon resolution resulted in an image that looked more like the source image.


(a) Generation 0 (b) Generation 100 (c) Generation 200

(d) Generation 300 (e) Generation 400 (f) Generation 499

Figure 9: Mona Lisa over 500 generations using difference between RGB values for fitness. Notice how the image has very little overlay and a lot of fixed shapes.

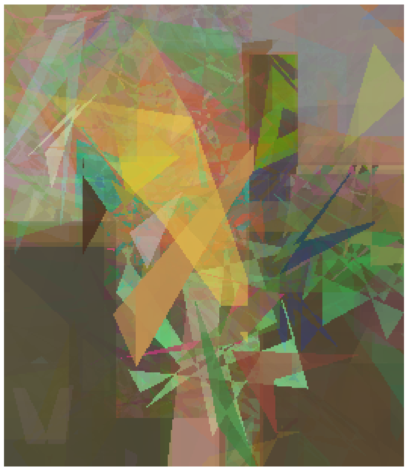
3.4 Future Work

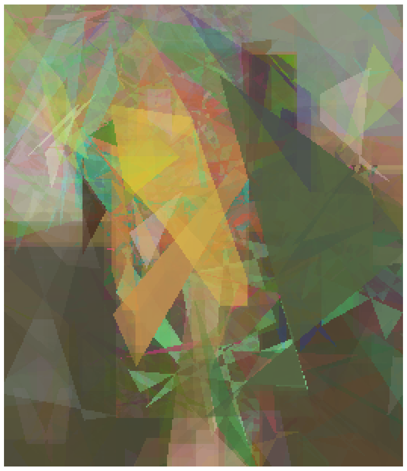
- Allow for the evolutionary process to be resumed after an incomplete run. Writing the current representation (population) onto some file and restoring the evolutionary process would be helpful. It would also allow us to experiment with how changing values on the go affects the results.

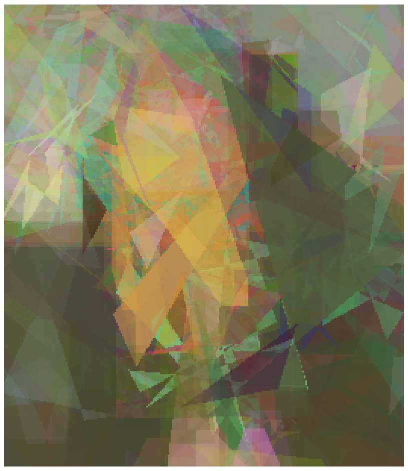
References

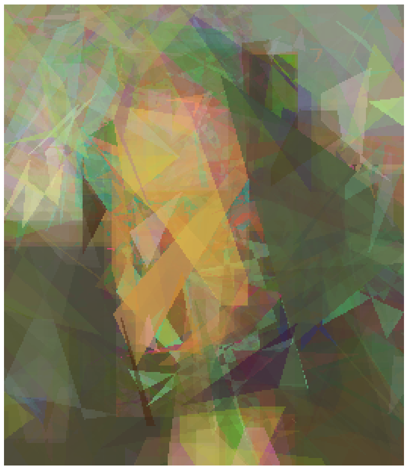
[1] Omar, M., Baharum, A., & Hasan, Y. A. (2006, June). A job-shop scheduling problem (JSSP) using genetic algorithm (GA). In Proceedings of the 2nd im TG T Regional Conference on Mathematics, Statistics and Applications Universiti Sains Malaysia.

[2] Charmot, S. (2022, September 4). A True Genetic Algorithm for Image Recreation — Painting the Mona Lisa.


(a) Generation 100


(b) Generation 200


(c) Generation 300


(d) Generation 400

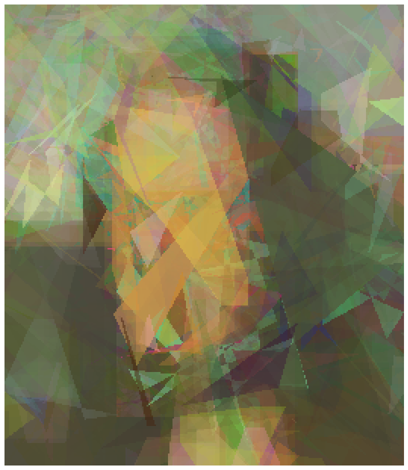

(e) Generation 499

Figure 10: Mona Lisa over 500 generations with maximum polygon size of 50, using delta E difference for fitness.