Image Reproduction using a Genetic Algorithm and Particle Swarm Optimization

Bo Bleckel[1], Jasper Houston[1], Dylan Parsons[1]
[1]*Bowdoin College, Brunswick, Maine, USA*
{*lbleckel, jhouston, dparsons*}*@bowdoin.edu*

Keywords: PSO, GA, genetic algorithm, particle swarm optimization, image reproduction, evolutionary algorithm

Abstract: Genetic Algorithms (GAs) are a form of evolutionary algorithm that can be used to solve a variety of optimization problems. In this paper, we applied a GA to perform image reproduction with a limited number of colored semi-transparent polygons, optimizing the output image to be as close to the input image as possible. We used Particle Swarm Optimization (PSO), another common optimization technique, to optimize the parameters on our GA. This allowed us to perform parameter optimization algorithmically rather than manually through testing a limited set of parameter values. We found that, while not successful at creating a perfect copy of the source image, this PSO/GA technique was successful in producing images that were visually identifiable as the source image to the human eye.

1 INTRODUCTION

In this paper, our goal is twofold. First, we will explore Image Reproduction (IR) using a Genetic Algorithm (GA). Second, we will use Particle Swarm Optimization (PSO) to optimize our GA to perform this task.

Our source of inspiration for this task was also twofold. After creating a number of, quite frankly, visually boring programs, we were interested in working on something that would catch the eye and show visual improvement over time. With this in mind, we looked around for inspiration until we stumbled upon Roger Alsing's blog and his use of a rudimentary genetic algorithm to re-create the Mona Lisa with a limited number of polygons. We decided that we would work with his idea as a baseline and improve on it by using a more complex genetic algorithm and hopefully improving performance to take less than three hours and almost a million generations.

One element that has consistently been on our mind while looking at other nature inspired techniques has been parameters. In our GA, there are a lot of parameters present including population size, probability of crossover, and probability of mutation, among others. Previously, we have looked at a limited number of possible values for each parameter and created a test function to loop through all possible combinations of parameters in order to optimize the function's performance. We decided that for this endeavor, we wanted a better way to optimize our parameters. This is where we had the idea to use PSO for parameter optimization.

When looking at PSO previously, we were optimizing challenging functions in n dimensions. In this case, we can think of n as being the number of parameters for the GA that we are trying to optimize. Each particle will run the GA to reproduce an image and the value of the particle will correlate to the fitness of the image created with the given parameters. Parameters will then be adjusted based on the particles around them and each particle will be re-evaluated given the new parameters for its GA.

In Section 2 we will discuss the somewhat complex topic of image reproduction. In Section 3 we will go into more depth about Genetic Algorithms and specifically the GA that we are using followed by a similar section about PSO in Section 4. These will be followed by a report of our experimental methods in Section 5 as well as results, potential further work, and conclusions in Sections 6, 7, and 8 respectively.

2 IMAGE REPRODUCTION

Image Reproduction (IR) refers to the process of taking an existing image and recreating it using some process. A very common application of this process is the reproduction of artwork for personal pur-

poses or for sale. 1st Art Gallery, found at https://www.1st-art-gallery.com/, is a company that reproduces artwork and is the world's largest supplier of Made-to-Order Oil Paintings. This art is also done on a smaller, non-industrial scale in the form of art projects in many schools, including the school of paper author Jasper Houston. For a project in high school, he reproduced a still life originally painted in 1925 by Pablo Picasso entitled Mandolin, Fruit Bowl, and Plaster Arm. The original piece in oil paints can be seen in Figure 1. The Houston reproduction in oil pastels can be seen in Figure 2.

Figure 1: Picasso Original
Credit: http://www.metmuseum.org/art/collection/search/486750

Figure 2: Picasso Reproduction by Jasper Houston

2.1 Roger Alsing and the Mona Lisa

In 2008, Roger Alsing was playing around with genetic programming and had an idea. In his words, "Could you paint a replica of the Mona Lisa using only 50 semi transparent polygons?" He built a successful Genetic Algorithm to perform this task which will be discussed further in Section 3. His results of recreating the Mona Lisa can be seen in Figure 3. It took almost a million iterations to perform this task (the .jpg name under each image is the iteration number), a process which Alsing reported taking around three hours.

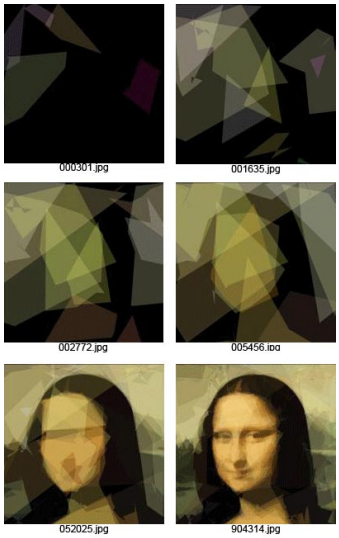

Figure 3: IR of the Mona Lisa
Credit: Roger Alsing, https://rogerjohansson.blog/2008/12/07/genetic-programming-evolution-of-mona-lisa/

2.2 Alpha Blending & Translucency

The translucency of each individual's triangles is an essential aspect of this process. When all triangles are translucent, their colors blend, allowing for more complex combinations and images. Similarly, a triangle can be fully opaque, eclipsing all triangles behind it. Changes in alpha values are very influential on the fitness of an individual.

3 GENETIC ALGORITHMS

A Genetic Algorithm (GA) is a type of evolutionary algorithm that implements a form of natural selection in the populations of animals over many generations in order to solve a given problem.

3.1 Alsing's GA

As previously mentioned, the reproduction of the Mona Lisa using a genetic algorithm was Roger Alsing's brainchild. Alsing's algorithm used the structure shown in Figure 4 using a string of DNA for polygon rendering. Alsing used a population size of two,

which meant that each generation had a parent and a child. The parent was the result of the previous generation and the child was the mutated version of the parent. In this way, Alsing's method is a form of asexual reproduction because it relies entirely on mutation to improve. There is no element of crossover because it is a single parent creating a single child for comparison.

```
BEGIN
create a random DNA string
for each generation:
  copy the current DNA sequence
  mutate DNA sequence slightly
  use new DNA to render polygons
  compare rendering to the source image
  if it is more accurate than previous:
    overwrite current DNA with new DNA
END
```

Figure 4: Pseudocode for Alsing's GA.

3.2 Evaluating Fitness

Accurate fitness evaluation is critical to the success of a GA. In order for the population to improve over time, each individual must have some measure of its fitness. Individuals with higher fitness are those that are more likely to continue on to the breeding pool to create the next generation. Our fitness evaluation is a measure of how far off the created image is from the original image. The fitness is given as a value between 0 and 1, and represents the fidelity of the recreation with respect to the original.

3.3 Selection

Selection is the process of choosing which individuals from a population to put into the breeding pool. There are many methods of selection, each with different rules for which individuals are chosen. Here, we've used Boltzmann selection, a method that biases individuals with higher fitness, but gives lower-fitness individuals the opportunity to be included. Methods that rely solely on high fitness levels become susceptible to convergence to local minima, and ignore potentially useful genes in less-fit individuals.

3.4 Recombination

The process of recombination takes individuals from the breeding pool, and creates a new generation by DNA combination. For simplicity, we implemented uniform crossover, a recombination technique that creates children by taking DNA from two parents with equal probability. Recombination, along with these other processes, mimics the process of evolution by combining the DNA of one generation to create another, hopefully more fit, generation.

3.5 Mutation

Where selection tends to increase the exploitative nature of GA, mutation provides the exploration. During each generation, individuals are given the chance to mutate, changing certain parts of their DNA with a small probability. In this case, individuals can mutate their triangles, which consist of vertices, a color, and an alpha value. Note that these four processes – fitness evaluation, selection, recombination, and mutation – are described in more detail in Section 5.

4 PARTICLE SWARM OPTIMIZATION

Particle Swarm Optimization (PSO) is a nature inspired technique that draws inspiration from the motion of a flock of birds to find an optimal solution in a solution space. Each particle in the swarm is likely to a bird in the swarm and represents a candidate solution to the problem, located within the solution space. The search works by updating the location of each particle based on their personal best—the cognitive component—and based on the best of the swarm or a subset of the swarm—the social component. The social component is often referred to as the global best, but in certain iterations of the algorithm it is set to the neighborhood best, where the neighborhood is defined for each particle as a certain subset of the swarm. Section 4.2 details the implementation of a neighborhood topology. In our previous work, we tested the effectiveness of the algorithm given different neighborhood topologies. In this iteration of the project, we fix the neighborhood topology so to be able to better evaluate our results. The goal is to use a Particle Swarm Optimization algorithm to optimize the parameters of a Genetic Algorithm.

4.1 PSO Algorithm

The general PSO algorithm works by generating a set of virtual particles each with an initial random velocity and position in the solution space. In the case of optimizing a parameter space, the initial velocities and positions are set to be in a predetermined range

for each parameter. Once the positions and velocities are initialized, for a specified number of iterations, update the velocity of each particle based on its personal best and the global/neighborhood best, update the position, evaluate the function f—in this case run the GA and let the best solution's fitness be the evaluation—at the new position, and finally update the personal best and global/neighborhood bests based on each particles' update positions. It is important to note that the PSO algorithm generally is optimizing a function to its minimum. Thus we made the GA return a value where a lower value means a better fit image to the PSO, even while the GA itself is trying to maximize the fitness value. More details on how we implemented this can be found in Section 5.2 For each particle i we call x_i the particle's position, and $y_i = f(x_i)$ the particle's value at position x_i. The velocity of each particle is denoted v_i, and the personal best of each particle is denoted p_i. We will call $pbest_i = f(p_i)$ the value of the personal best and, similarly, g_i is the global best for each particle i and $gbest_i = f(g_i)$ is the function value at the global best g_i. For each iteration, the position gets updated based on the velocity. Thus we must understand how the velocity gets updated. Equation 1 shows the velocity update equation used for all particles $1 \leq i \leq n$ where n is the total number of particles.

$$v_i = \chi\Big(v_i + U(0,\phi_1) \times (p_i - x_i) + U(0,\phi_2) \times (g_i - x_i)\Big) \tag{1}$$

In equation 1, χ is what is called the constriction factor. The constriction factor is used to keep the velocities of each particle from getting our of control on each velocity update. The constriction factor is generally described as

$$\chi = \frac{2k}{\phi - 2 + (\phi^2 - 4\phi)^{\frac{1}{2}}} \text{ where } k = 1,\ \phi = \phi_1 + \phi_2.$$

For our purposes we will use a constriction factor $\chi = 0.7298$ based on the assumption that $k = 1$ and $\phi_1 = \phi_2 = 2.05$. If k is higher, that allows for more exploration, and if k is lower that allows for more exploitation. v_i is, as described above, the velocity of the particle, which in the case of the v_i on the right side of the equation is the previously velocity and in the case of the v_i on the left side of the equation is the velocity after the current update. U is a function that takes two parameters k_1, k_2 and returns a vector with n elements, populated with random numbers $x_i \in \mathbb{R} \,|\, k_1 \leq x \leq k_2, 1 \leq i \leq n$.

4.2 Neighborhood Topologies

Neighborhood topologies are used to create the subsets within the swarm that will share a common "global" best. This best will be used as the value for g_i when we evaluate the equation detailed in equation 1. In Section 4 we called the neighborhood best the social component. There are many possible neighborhood topology variations used in PSO algorithms. The most straightforward would be the Global topology, which connects every particle in a swarm to every other particle in the swarm, thus following the global best as it is discovered. Global neighborhoods can lead to the swarm getting stuck in local minima. Other options such as the Von Neumann neighborhood topology and the Ring topology are common, and we explored them in our previous work. For this project, based on results from our previous work, we focused on the Random topology. The Random topology performed best on functions with many local minima. We believed that exploring the GA parameter space might lead to many local minima, thus we decided that the random neighborhood topology would be our best bet to optimize the problem.

4.2.1 Random Neighborhood Topology

In the Random topology, a neighborhood of size k is initialized for each particle by choosing $k-1$ particles at random without repetition from the swarm. This method is dynamic as it creates a new neighborhood for each particle with probability 0.2 on each iteration. This gives way to one issue that we must point out with the Random topology: it is more computationally expensive, since we must recreate a topology for each particle with probability 0.2 on each iteration. Note that in a Random neighborhood topology, like many other topological groupings, each particle is included in its own neighborhood.

4.3 Parameter Optimization

PSO algorithms are generally used to optimize complex multidimensional functions. Our use of PSO will be for parameter optimization for the GA. Each dimension of this problem can be considered a parameter for the particle to explore. Each particle will be an instance of the GA performing IR on a given image. The final output of the PSO algorithm will be a set of the optimal parameters for which to run the GA.

5 EXPERIMENTAL METHODOLOGY

5.1 Genetic Algorithm

5.1.1 Selection

We chose to use Boltzmann selection in order to give each individual in the population a chance to be selected. This process is outlined in Figure 5.

```
BEGIN
do n times:
   for each individual in the population:
      select individual i with probability:
      e^f[i] / Σ_{j in population} e^f[j]
      exit for
END
```

Figure 5: Pseudocode for our Selection Method.

Where n is the number of individuals in the population, and f[i] is the fitness of the i^{th} individual. Here, the individual selected in a given iteration of the 'do' loop is the first whose ratio, expressed above, is greater than a random value.

5.1.2 Fitness Evaluation

A perfect individual would look exactly like the original image; hence, all of its pixels' color and alpha values would be exactly the same. Hence, we calculate fitness as the sum of the differences in pixel values.

```
BEGIN
   for pixel in candidate solution:
      difference = 255 − ((abs(r − r0) + abs(g − g0) + abs(b − b0)) / 3)
      fitness += difference
END
```

Figure 6: Pseudocode for our Selection Method.

Where r, b, and g are the RGB values of the individual, and r_0, g_0, and b_0 are the RGB values of the original image at the same pixel. In English, the fitness of an individual *pixel* is the average of the difference of its RGB color values, subtracted from 255 (the maximum difference). In this way, a smaller difference results in a higher overall value.

5.1.3 Recombination

Recombination is an essential step in a genetic algorithm; it allows the combination of different strands of DNA that might otherwise never occur. We used uniform crossover to achieve this step; in uniform crossover, DNA is chosen from each parent with equal probability. Here, a child has the opportunity to inherit triangles, colors, and alpha values from its parents. Another option would be to inherit individual vertices, but we chose to use entire triangles instead. We introduced a slight bias towards the fitter parent: the child is more likely to inherit a piece of DNA from the fitter parent (a 70/30 split).

5.1.4 Mutation

Providing an important boost to the exploration ability of the algorithm, mutation, randomly changes certain strands of DNA by a given amount. We applied the chance of mutation to the coordinates of the vertices of each triangle, each triangle's color, and each triangle's alpha value. The default amounts for these mutations were 10% (of the image size), 10 (on a 0-255 RGB scale), and 0.1 (for a 0-1 float), respectively.

5.2 Particle Swarm Optimization

The Particle Swarm Optimization algorithm needed some tweaking to work with the solution space of a genetic algorithm. Each dimension in this case represented a parameter in the GA. The range for the initial values for the position in each dimension did not fall within a uniform range for each dimension like they might when optimizing a mathematical function. Instead, we initialized the value for each dimension of the position vector in each particle to a random value within a range that we thought would be a generally good range for that parameter. We assigned each parameter their dimension arbitrarily but not randomly. The order was set as follows with the following range for initial values:

1. Number of individuals: $\{2, 10\}$
2. Number of triangles: $\{4, 100\}$
3. Probability of crossover: $\{0.0, 1.0\}$
4. Probability of mutation: $\{0.0, 1.0\}$
5. Alpha mutation amount: $\{0.0, 1.0\}$
6. Color mutation amount: $\{1, 20\}$
7. Point (x, y) mutation percentage (set to be a percentage of the height of the image): $\{0.1, 0.2\}$

The number of individuals in the number of individuals in a population within the GA. The number of

triangles was the total number of triangles that each individual in the population had to make up the image. These were essentially a set of three points, and a color. The probability of crossover and probability of mutation are parameters described in Sections 3.3 and 3.4 respectively. The alpha, color, and point mutation amount are specific to the problem of image reproduction using a limited number of polygons. The alpha mutation amount was the amount that alpha would mutate if mutation occurred. Similarly, color mutation amount was the amount that the color values would mutate if mutation occurred. The point mutation amount was the amount that each of the three points of the triangle would mutate if mutation occurred. The direction of mutation was always random.

Given that the GA will only take a certain value for some parameters, we had to limit the range of the parameter space that the PSO algorithm explored. For example, there cannot be a negative number of individuals or triangles. Similarly the probability of crossover and mutation cannot be negative or greater than 1. In order to do this, we would return 100 anytime that any parameter was out of the bounds of its given range. As the PSO was set up to minimize the function, returning a value of 100 will quickly push the particles away from any such parameters. In practice it was interesting to find that after just one iteration of the PSO algorithm it quickly strayed from any parameters that would be out of bounds.

As mentioned earlier, the PSO algorithm is one that minimizes a function. Given that our GA was set up to evaluate the fitness of an image, with a value closer to 1 being the best, and a value closer to 0 being the worst, we had to invert the return value from the GA when returning to the PSO. Because all values returned from the GA are between 1 and 0 (unless a parameter was out of bounds and it was 100), we simply returned 1 minus the fitness of each image, and thus the PSO algorithm could optimize the function.

We also had to deal with parameters that were required to be integers. Specifically, the number of individuals, the number of triangles, and the color mutation amount (dimensions 1, 2, and 6) were required to be integers, however, the nature of a PSO algorithm explores the continuous space over all real numbers. Thus when sending a parameter to the genetic algorithm, we took the value in the associated dimension, and floored that value. For example, a value of 5.78 in dimension 1 (the number of individuals) would be floored $\lfloor 5.78 \rfloor = 5$. Similarly, we found that the probabilities for crossover and mutation quickly became too large, and thus we divided them by 100 before sending them to the GA.

To maintain consistency, we set the number of particles to a fixed value of 10 particles, and the number of iterations to 1000 iterations. It is important to note the low number of particles and low number of iterations. PSO generally does not perform terribly well over just 1000 iterations, however we were limited on time. For each iteration, and each particle, we had to call the GA, which itself required many generations to be effective. To help with this, we did limit the GA to run only for a minute and then return. Luckily we found in our previous work that the PSO algorithm with the random neighborhood topology was effective with fewer iterations and fewer particles.

We first tested the genetic algorithm using arbitrary values for the parameters on many images. The results from that are described in the following section. Once we were confident that the GA was running somewhat properly, we ran the PSO algorithm on it. To test the effectiveness of the PSO algorithm, we only tested it on three images: two simple ones with no more than four total colors, and one complex drawing of a house. The results for this are also detailed in the section below.

6 RESULTS

6.1 Genetic Algorithm

We wanted to ensure that our GA was working before started using PSO to optimize it because we knew it was going to run for hours and we didn't want to end up with nothing after an overnight run of the algorithm. After extensive debugging, we were able to manipulate parameters manually such that we were able to generate the Firefox logo as seen in Figure 7. The source image is seen in Figure 8 for reference. Once we were able to generate this successfully, we continued with PSO.

6.2 Particle Swarm Optimization Optimizing GA Parameters

We ran the PSO on three pictures, with 1000 iterations. Given that there is no known best parameter setting other than the one that results in 100% fitness (a practically impossible feat using this image recreation approach), the results from the PSO are much more visual than quantitative. While the parameters that the PSO algorithm finds are noteworthy, the images that those parameters produce are arguably more interesting. Note, however, that we did not have enough to time to run the program for the full 1000 iterations

Figure 7: Firefox Replica with fitness 85.9%.

Figure 8: Firefox Logo.
Credit: https://en.wikipedia.org/wiki/File:Firefox-logo.svg

on each image. The most we got was 98 on one of the images. The images that we used are shown in Figures 9, 10, and 11.

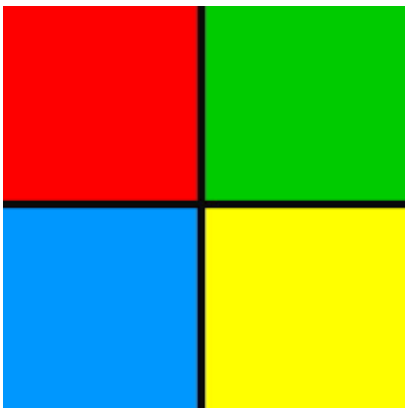

Figure 9: Colorful squares that vaguely resemble the Windows logo
Credit: https://www.pinterest.com/pin/461056080578814227/

Our most successful run–visually–of the PSO on the GA was with the squares show in Figure 9. The

Figure 10: Image of a heart
Credit: https://www.indiegogo.com/projects/we-love-you--3

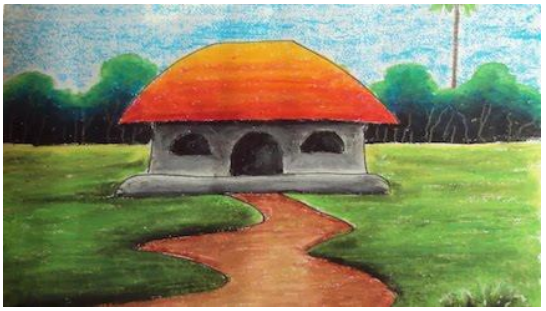

Figure 11: Painting of a house
Credit: https://i.ytimg.com/vi/8a8a4hM83pQ/maxresdefault.jpg

PSO ran for 98 iterations before we had to cut if off because of time constraints, but was able to find parameters that produced the image shown in Figure 12. The parameters that it found reflect common sense parameters for the most part. Those parameters are shown in Figure 13. The progress of the PSO algorithm is show in Figure 19. As shown in the graph, 98 iterations was not enough to clearly show the progress of the PSO algorithm in optimizing the parameters, but again we were limited by time.

The results for the PSO on the other two images are shown in Figures 14 through 17.

The results show that the PSO algorithm was able to learn reasonably good values for the GA when running it on simple images. The Squares image had a best fitness of 91.9% and the Heart image had a best fitness of 96.2%. However, looking at the progress shown in Figure 19 it is clear that the algorithm's global best stagnates for a while before getting better. This leaves us to weary to make any assumptions about how conclusive the data is. We would need to allow more iterations of the PSO algorithm to determine its effectiveness in optimizing the parameters for the GA. We can say, however, with some certainty, that the PSO was not able to find parame-

Figure 12: PSO best for Figure 9. Parameters for this image are shown in Figure 13. The fitness of this image was 91.9%.

Individuals	2
Triangles	40
Probability of crossover	0.18
Probability of mutation	0.09
α mutate amount	0.04
Color mutate amount	27
Point (x,y) mutate amount	5.46
Total Fitness (%)	91.9

Figure 13: Results from PSO on GA with image shown in Figure 12.

Figure 14: PSO best for Figure 10. Parameters for this image are shown in Figure 17. The fitness of this image was 96.2%.

ters that yielded better results than we were able to using brute force methods. Preliminary results of the GA with arbitrary parameters that we thought were in a reasonable range (these ranges were within the ranges enumerated in Section 5.2), qualitatively gave us results that were as good if not better than what the PSO was able to do. This is most likely due to the fact that we let the generations of the GA run their due course, rather than stopping them after 1 minute.

Individuals	9
Triangles	8
Probability of crossover	0.317
Probability of mutation	0.084
α mutate amount	0.095
Color mutate amount	15
Point (x,y) mutate amount	30.4
Total Fitness (%)	96.2

Figure 15: Results from PSO on GA with image shown in Figure 16.

Figure 16: PSO best for Figure 11. Parameters for this image are shown in Figure 17. The fitness of this image was 96.2%.

Individuals	18
Triangles	25
Probability of crossover	0.134
Probability of mutation	0.013
α mutate amount	0.115
Color mutate amount	23
Point (x,y) mutate amount	16.5
Total Fitness (%)	88.2

Figure 17: Results from PSO on GA with image shown in Figure 16.

Given the 1 minute time constraint, the PSO was relatively successful in finding good parameters (based on the fitness of the images that it produced) and qualitatively effective at reproducing images.

6.2.1 Fitness Function in the GA

One thing that that we pondered deeply was the effectiveness of our fitness function in the genetic algorithm. Figure 16 is a perfect example of when the GA's fitness value does not correspond well to the visual accuracy of the IR. Therefore our fitness function could have been improved to better reflect how accurate the IR was to the source image. Additionally, one of the biggest limiting factors in the project was the time that it took the algorithm to run and we believe that this is due mostly to the speed at which we currently compute the fitness of an image. Because

each particle had to call the GA on each iteration, we could really do nothing but limit the time that the GA would run for. As a consequence, we didn't get to let the GA reach its full potential. In addition, the PSO was still only able to run around 90 iterations over 12 hours. In looking at the work of our colleagues who have explored this problem before, it seems that there is a better and quicker way to evaluate the fitness of an image.

7 FURTHER WORK

7.1 Disappearing Triangles

Upon examining the evolution of a population over many generations, it became clear that triangles sometimes mutate to the point of being practically invisible. This happens, for example, when a colored triangle is in an area corresponding to a white chunk in the original image, so the individual sees increased fitness from reducing the amount of space the triangle occupies. However, it also precludes the possibility of that triangle contributing somewhere else in the image, potentially helping shape and color the individual. In order to avoid this, we might set a lower bound on the area of each triangle, and prevent vertex mutations that bring the area of the triangle below that area. In this way, evolution will shift the triangles elsewhere or change their color, rather than making them vanish.

7.2 More Vertices

We chose to only use triangles with our algorithm when re-creating images, but it would certainly be possible to use other shapes as well. We thought of this when looking at one particular image we were trying to reproduce, a Mondrian painting seen in Figure 18. Our algorithm had a hard time reproducing the image because it had so many rectangles in it, and this made us think of using quadrilaterals rather than triangles. The quadrilateral idea then turned into the idea of being able to change the polygon used in re-creation to be any polygon. Using polygons other than triangles would potentially allow for better reproduction of some images.

7.3 Algorithm Speed

One thing we noticed about our algorithm was that it was very slow compared to others we had come across while researching, such as Chris Cummins' Grow

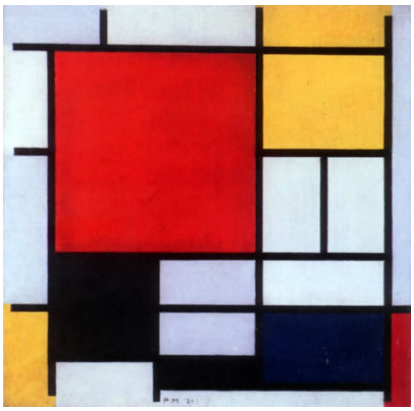

Figure 18: Mondrian Painting
Credit: `http://www.theartstory.org/artist-mondrian-piet.htm`

Your Own Mona Lisa at `http://chriscummins.cc/s/genetics/`. Not only did it take more generations to find an accurate image, but it was also significantly slower to move from one generation to the next when given similar parameters. We did not have the opportunity to look into why this was, but we do know that there is a faster way of performing image re-creation with a GA than the technique we used.

7.3.1 Advanced Initialization Tactics

A possible way to speed up our algorithm would be cheating a little bit and not using a randomized starting population. One method of cheating with the starting population could be to sample colors from the source image in the center of each random triangle and assigning those color values to the triangles for the initial population. This would result in a fairly accurate initial population that would then be able to perform crossover and mutation in order to perfect itself.

7.3.2 Alternative Mutation

A second possible way of speeding up our algorithm would be an alternative mutation algorithm. In our algorithm, we choose whether or not we are going to mutate for each element of each triangle of each individual. Rather than doing this, we could simply decide whether or not we were going to mutate an individual then mutate every aspect of that individual. This could result in more extreme mutation, which could potentially help us reach a more accurate solution more quickly.

8 CONCLUSIONS

By implementing PSO to optimize our GA, we were able to generate good results without having to manually optimize the parameters. Even if we had optimized the parameters manually, it is important to note that these optimal parameters would likely change from image to image. By having parameter optimization as a part of our algorithm, we know that we will be generating an optimal solution regardless of the image and the changing optimal parameters.

Our algorithm had its strengths and drawbacks. It was relatively accurate in reproducing high-contrast images and less complex images. This makes sense based on our limitations we set in terms of time for the PSO and generations when just running the GA. Given more time and more triangles to work with, we believe it would have a shot at reproducing more complex images. The speed of our algorithm is certainly an issue, with PSO performing an instance of the GA for every particle in every iteration. Given more time, we would have been able to give each instance of the GA more time to evolve and produce a better image and PSO would have had more accurate fitness values to work with. Additionally, in hindsight, using triangles may not have been the best polygon for most image reproduction.

We can see this technique of using PSO for parameter optimization being used for other applications outside of hybrid nature-inspired algorithms in order to guarantee the use of optimal parameters.

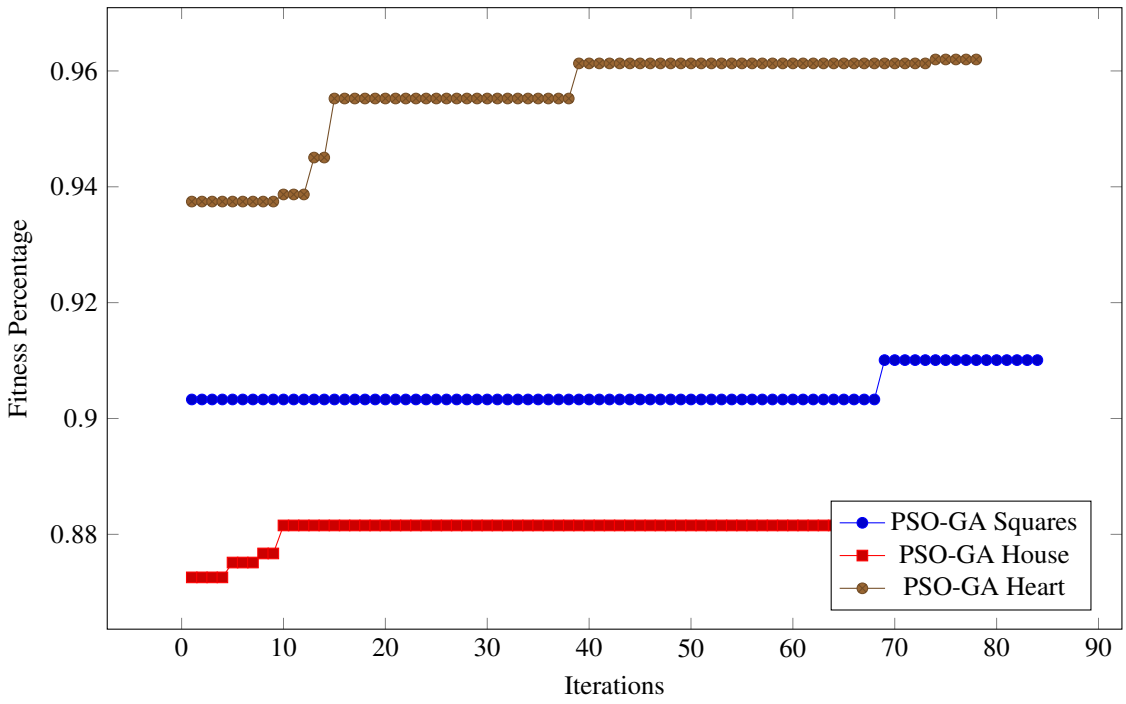

Figure 19: Particle Swarm Optimization algorithm optimizing parameters for a GA on multiple images.