CS268 Project: From Polygons to Image

Chen Li, Disi Ji

December 11, 2015

1 Abstract

In this project, we aim at using a fixed number of polygons with a fixed number of vertices to approximate a given image. For each polygon, the position of each vertex, RGBA of the shape need to be decided. Previous work to solve this problem used the random search method, i.e., to randomly modify one polygon each time, and then decide whether to accept the modification by calculating the change of fitness introduced by the it, where fitness is defined to be the distance between the result and the original image. We used three different methods, including gradient descent, simulated annealing, and genetic algorithm, to improve the performance. Experiments are conducted to compare the performance of different algorithms. Results show that gradient descent works best among all the algorithms.

2 Problem Statement

Given an image, each of which includes information of RGBA. Now use 50 hexagons to approximate it. For pixels in the shape of a hexagon, they are colored with the same RGBA values. For those out of any hexagons, they are set by default to be white. The set of 50 hexagons is a solution to the problem. Fitness of a solution is defined to measure the distance between the solution and the original image. Fitness is higher when the solution approximates the image better. Now we need to decide the position and the color of these hexagons.

3 Introduction

Using polygons to approximate images has been an active field of study.[1] In this project, given an image, we try to use a fixed number of polygons with a fixed number of vertices(50 hexagons to be more specific) to achieve this goal. Obviously, it is not a particularly sophisticated method to image compression, not at least when the number of polygons are limited to 50. But experimental results have displayed that lossy image compression can be performed when using appropriate methods to select polygons, and approximated images are acceptable when number of iterations is big enough.

For a image with $m * n$ pixels, each pixel is a point in a 4-dimensional space comprising the intensities of the red, green, blue and alpha channels, each of which is stored with 8 bits of precision. Then to transmit the whole image directly would cost $8 * 4mn$ bits.

Now suppose we use N polygons each of which contains v vertices to approximate the image. Position of vertices are also stored with 32 bits of precision. Then the total cost will be $N(2v * 32 + 8 * 4)$, which is a dramatic reduction compared to storing directly.

Our work is based on Roger Alsing's excellent idea(`http://rogeralsing.com/2008/12/07/genetic-programming-evolution-of-mona-lisa/`), and the implementation `http://alteredqualia.com/visualization/evolve/` by alteredq. The algorithm they used to select polygons is basically random search. Different methods, including gradient descent, simulated annealing and genetic algorithm are used to improve the performance, in terms of both running time and fitness.

1. Marc Levoy, "Polygon-assisted JPEG and MPEG compression of synthetic images," in *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques* (ACM, 1995), 21–28.

4 Contribution

Based on the implementation of the framework before, we implemented 3 different methods to improve the performance and compared the results among them and the original method. Both of us worked together and contributed equally to this project.

5 Algorithms

Random search is a naive idea to solve the problem. It works but with the price with large scale of sampling and calculation. To be more specific, randomly modify one polygon each time, and decide whether to accept it by calculating fitness. If the modification leads to the increase of fitness, then accept, else refuse it. Gradient descent and simulated annealing are two direct methods to improve

Algorithm 1 Random Search

```
Shapes[#ofpolygons] ← Randomly initialize each polygon
while (StoppingCriteria) do
    i = rand(#ofpolygons)
    NewPolygon ← Randomly generate a new polygon
    ShapesNew = Shapes
    ShapesNew[i] = NewPolygon
    if (fitness(ShapesNew) > fitness(Shapes)) then
        Shapes = ShapesNew
    end if
end while
```

the idea, while genetic algorithm is a rather different way to solve this problem.

5.1 Gradient Descent

Instead of randomly generate mutation, if last modification of shapes results in the increment of fitness, it is very likely that to continue modifying the shape in this direction will still increase fitness. Compared to generating changes blindly, experiments proved that gradient descent improved the performance *****

Algorithm 2 Generate NewPolygon with Gradient Descent

```
if (fitness(ShapesNew) > fitness(Shapes)) then
    i = index of the changed the shape
    Step = ShapesNew[i]-Shpes[i]
    NewPolygon += Step
else
    NextPolygon ← Randomly generate a new polygon
end if
```

5.2 Simulated Annealing

In the random search method, modification is only accepted when it improved the fitness of the result. However, this hill climbing method may lead to local minimal. To solve this problem, we introduce simulated annealing method.[2] "Metropolis algorithm" is employed, i.e., bad modifications are also accepted with a certain probability.

2. Ashok D Belegundu and Tirupathi R Chandrupatla, *Optimization concepts and applications in engineering* (Cambridge University Press, 2011).

Algorithm 3 Accept Modifications using Simulated Annealing

$delta = fitness(ShapesNew) - fitness(Shapes)$
if ($delta \geq 0$) **then**
    Shapes = ShapesNew
else
    $q = Bolzmann(delta)$
    $p = q/(1+q)$
    $t = rand(1.0)$
    if (t<p) **then**
        Shapes = ShapesNew
    else
        break
    end if
end if

5.3 Genetic Algorithms

Genetic algorithm (GA) is a search heuristic that mimics the process of natural selection. It is routinely used to generate useful solutions to optimization and search problems using techniques inspired by natural evolution, such as inheritance, mutation, selection, and crossover.[3]

In our problem, each polygon is considered to be a DNA. Each individual has 50 DNA and there are a group of individuals to be considered as the possible solution. In each generation, individuals are selected according to their fitness to produce the next generation, with crossover happening during the process. Individuals with higher fitness are more likely to chosen. Genes also mutate randomly at every position. For each generation, the size of the population stays the same, and the individual with the highest fitness is considered to be the solution to the problem.

Algorithm 4 Genetic Algorithm

Individuals[population size]← Randomly Initialize DNA of the population
while (!*stop*) **do**
    Calculate fitness of the group $fitness[populationsize]$
    Calculate probability of being chosen to generate children $probability[i] = \frac{fitness[i]}{sum\_of\_fitness}$
    for (i in 0:Population size) **do**
        Generate one child through crossover of two chosen parents
    end for
    Each position in DNA of each individual mutates randomly with p_{mutate}
end while

6 Experiments

6.1 Experiment Setup

We build a simple web app, using JavaScript to implement the optimization algorithm and HTML5 canvas to render the results. Also, we choose the classical Mona Lisa as the rendering target to test our algorithms.

6.1.1 Implementation

1. Random Search: It randomly change an attribute(color or position) of a random polygon, then compute the fitness after change. If the result it better than previous, it accepts the change,

3. Darrell Whitley, "A genetic algorithm tutorial," *Statistics and computing* 4, no. 2 (1994): 65–85.

else it goes back to the previous status and make a random search again

2. Gradient: Gradient is an improved version of random search algorithm. If a random search step provides better performance, then the next step will follow this direction as long as the result is getting better. If the result get worse, then it goes back to random search to find an new direction.

3. Simulated Annealing: First take a random step, if the step gives better performs, accept it, otherwise accept it with a probability based on the current temperature.

4. Genetic: First encode our polygons into DNA code, then preform selection, recombination and mutation on the set of DNA.

6.1.2 Data

Pixel of the picture Mona Lisa. Each pixel consists of RGB and transparency value.

6.1.3 Fitness function

Fitness function is defined to be

$$Fitness(s) = 100 * (1 - \frac{cost(s)}{width * height * 3 * 255})$$

, where cost function is the sum of difference by pixels.

6.1.4 Parameters

1. Initial temperature in Simulated Annealing.

2. The population, mutate rate and crossover length in Genetic Algorithm.

6.2 Results

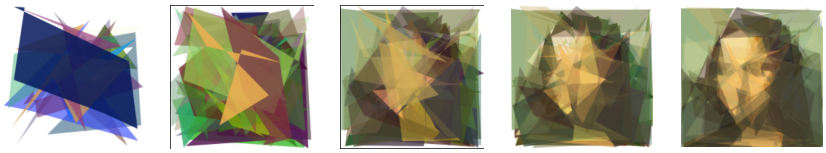
6.2.1 improvements and running time during 50000 mutations

We run random search, gradient descent and simulated annealing method each 50000 mutations(iterations) to compare their difference. The genetic algorithm is treat differently, because it can not get a reasonable results in the limited iterations and also the computation is very slow because of the underlying HTML Canvas implementation. Thus we focus on tuning its parameters to see the difference. Also, the time is depends on the browser and the rendering process, so it's not only the time of running the algorithm, but also other part of the app, so we choose mutations to compare.

Random Search Method

Random search performs as the benchmark of our different algorithms. It's fitness grows steadily with the increment of mutations and results in an acceptable fitness. Also, it takes reasonable time.

$Mutation$	$Fitness(\%)$	$Time(s)$
100	67.356%	0.807
1000	81.099%	8.330
5000	88.118%	41.742
10000	90.561%	83.507
50000	93.969%	570.019



(a) N = 100 (b) N = 1000 (c) N = 5000 (d) N = 10000 (e) N = 50000

Figure 1: **Original Method.**

Gradient Method

Gradient method has a significance increment on the fitness and take the approximately same time as the Random Search Method.

$Mutation$	$Fitness(\%)$	$Time(s)$
100	61.608%	0.803
1000	83.926%	8.411
5000	91.201%	41.845
10000	92.945%	87.251
50000	95.003%	419.030

Simulated Annealing Method

Simulated Annealing also has improvement on random search, and the results get better when using higher temperature.

$Tempareture$	$Mutation$	$Fitness(\%)$	$Time(s)$
25	100	54.919%	0.818
	1000	79.366%	8.353
	5000	89.327%	41.782
	10000	91.793%	83.481
	50000	94.280%	425.765
50	100	55.996%	0.812
	1000	80.638%	8.380
	5000	89.548%	41.695
	10000	91.972%	83.422
	50000	94.325%	416.526

Genetic

Genetic Algorithm will stabilize on a very low fitness after amount of mutations. Probably because this rendering problem is not like the other optimization problem which has the tendency that two

(a) N = 100 (b) N = 1000 (c) N = 5000 (d) N = 10000 (e) N = 50000

Figure 2: **Simulated Annealing, T = 25.**

DNA with high fitness will provide children with higher fitness(at least not lower) children. In our problem, recombination between DNA is changing color and position in corresponding polygons, which may has an unpredictable influence on the value of pixel and the fitness. We tuning the population, mutate rate and crossover length, and the results shows in the follow graph. We can see that increase the population is not guarantee to increase the fitness, so does the mutate rate, which seems to perform best near 0.01. Also, we can see that the shorter the crossover length, the better the results.

Compare

We compare the algorithms on 50000 mutations, and the results are as follows. The random search, gradient descent and simulated annealing gets the similar results, where gradient descent gets the best result. Genetic algorithm stop improving after 10000 iterations.

7 Conclusion

Using polygons to approximate images is a practical problem to solve. Random search, gradient descent and simulated annealing all generate reasonable results, but genetic algorithm stop improving after several iterations. Among all these methods, gradient descent achieves best performance in terms of both running time and fitness.

References

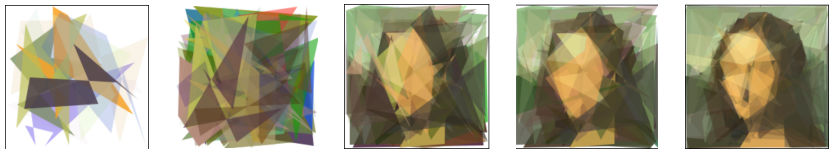
Belegundu, Ashok D, and Tirupathi R Chandrupatla. *Optimization concepts and applications in engineering*. Cambridge University Press, 2011.

Levoy, Marc. "Polygon-assisted JPEG and MPEG compression of synthetic images." In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, 21–28. ACM, 1995.

Whitley, Darrell. "A genetic algorithm tutorial." *Statistics and computing* 4, no. 2 (1994): 65–85.

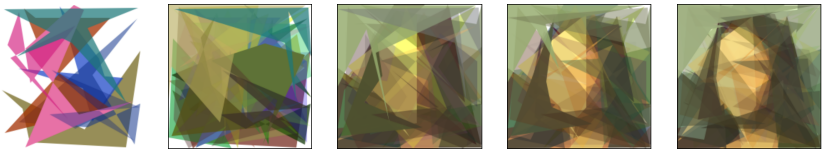
(a) N = 100 (b) N = 1000 (c) N = 5000 (d) N = 10000 (e) N = 50000

Figure 3: **Simulated Annealing, T = 50.**



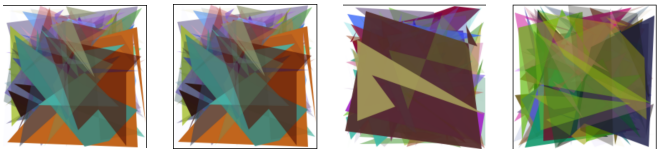
(a) N = 100 (b) N = 1000 (c) N = 5000 (d) N = 10000 (e) N = 50000

Figure 4: **Simulated Annealing, T = 100.**



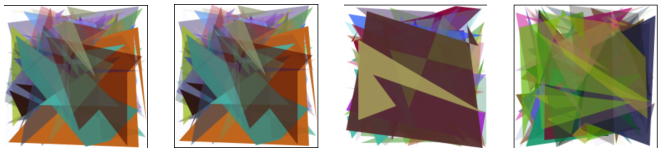
(a) N = 100 (b) N = 1000 (c) N = 5000 (d) N = 10000 (e) N = 50000

Figure 5: **Gradient Descent.**



(a) N = 100 (b) N = 500 (c) N = 1000 (d) N = 5000

Figure 6: **Genetic Algorithm, Population Size = 50.**



(a) N = 100 (b) N = 500 (c) N = 1000 (d) N = 5000

Figure 7: **Genetic Algorithm, Population Size = 100.**

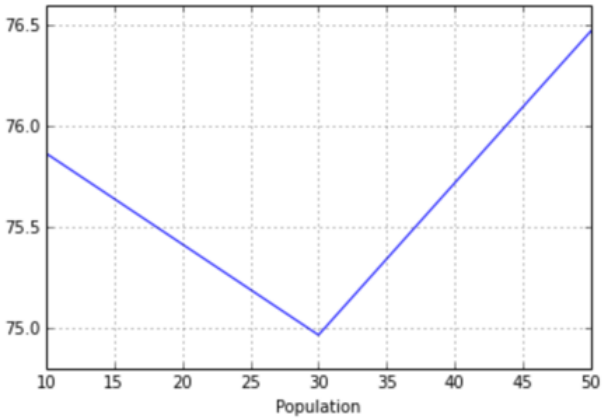


Figure 8: **Different population size in genetic algorithm.**

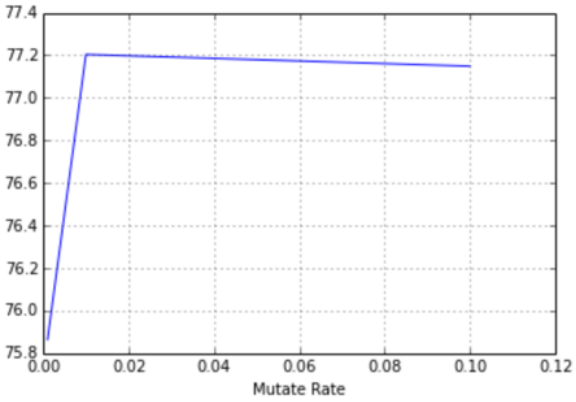


Figure 9: **Different mutation rate in genetic algorithm.**

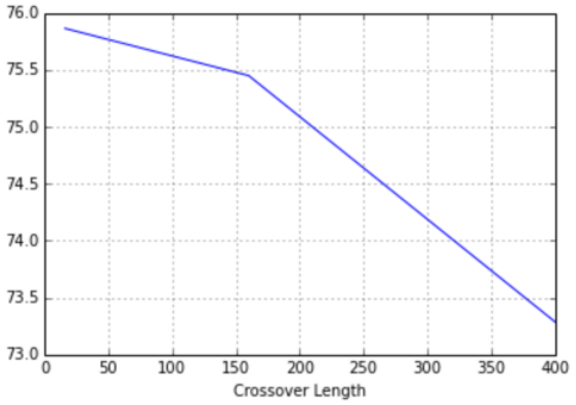


Figure 10: **Different crossover length in genetic algorithm.**

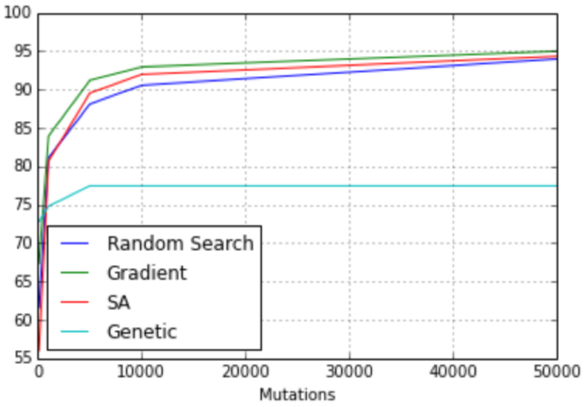


Figure 11: **Compare different algorithms.**