A Genetic Algorithm Approach to Regenerate Image from a Reduce Scaled Image Using Bit Data Count

Kishor Datta Gupta
Ph.D. Student, Computer Science
University of Memphis, United States of America
3720 Alumni Ave, Memphis, TN 38152, USA
kdgupt1@memphis.edu

Sajib Sen
Ph.D. Student, Computer Science
University of Memphis, United States of America
3720 Alumni Ave, Memphis, TN 38152, USA
ssen4@memphis.edu

Abstract

Small scaled image lost some important bits of information which cannot be recovered when scaled back. Using multi-objective genetic algorithm, we can recover these lost bits. In this paper, we described a genetic algorithm approach to recover lost bits while image resized to the smaller version using the original image data bit counts which are stored while the image is scaled. This method is very scalable to apply in a distributed system. Also, the same method can be applied to recover error bits in any types of data blocks. In this paper, we showed proof of concept by providing the implementation and results.

1. Introduction

Revolution of a portable camera with computer started to produce an exponential rate of media files, and users are sharing these files with everyone. So, using the cloud to store images is becoming a favorite choice for users. But cloud does not only store huge files which are approximately 1.2 trillion in 2017 (Perret, 2017), it also has to transfer these files to a different network to serve users. To reduce load, the cloud system started to use different compression algorithm. These algorithms have a tradeoff between time and space. Most of these have better time complexity than space. But as the cloud has powerful and distributed computing power, it may be better to focus on saving space. As data transfer takes more time than processing same data in the cloud. A perfect use case is a mobile sending the large image to the cloud takes more data transfer time than the compression and decompression process in the cloud. So, in this age of the distributed computer, it is better to reduce size as computation time is less important than network data transfer time.

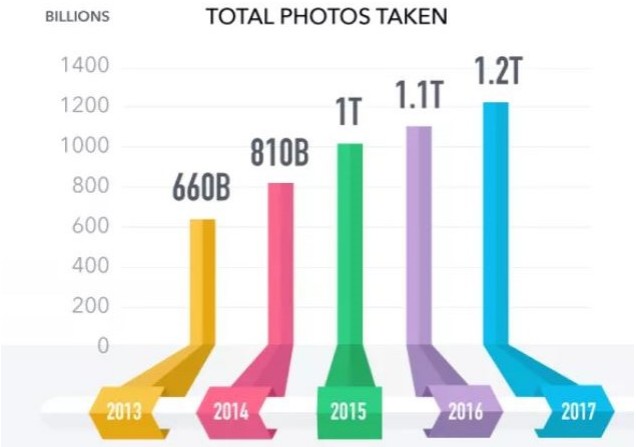


Figure 1. Photo amount by year (Perret, 2017)

Currently, 4.7 trillion of photos are saved in the cloud (Perret, 2017). And only a few percentage are called to use again. So less used files can be stores in a compression technique which can save more space than time and make the cloud system faster as memory redundancy time will be reduced.

Figure 2. Image stored in cloud each year (Perret, 2017)

Therefore, it's expected that we need a more space preserving algorithm which can work well in distributed operating system.

2. Background

An image file can be modeled using a continuous function of three variables; they are X, Y and T. X and Y are coordinates of x, y in a plane, and T is time, if image changes in respect to time. For normal image time T is always static 1. So, we can ignore it for our work. (55:148 Digital Image Processing, 2017). To work with image, we need to digitize the image, and it means that function f (x, y) have to sample in a matrix with H rows and W column.

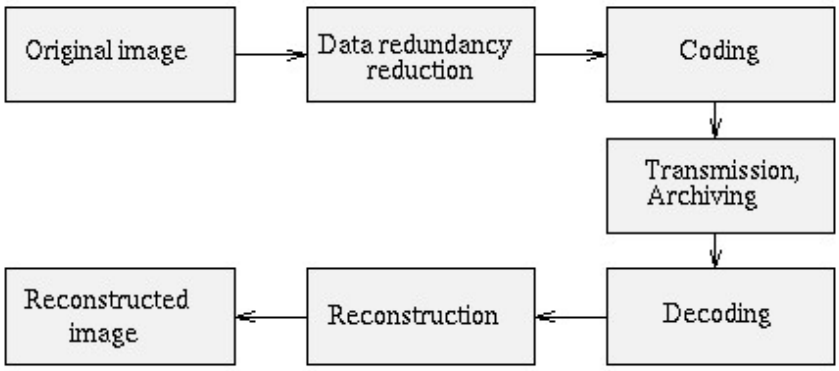


Figure 3. Data compression and image reconstruction (55:148 Digital Image Processing, 2017)

There are several techniques which are normally divided into two categories lossy and lossless image compressions. In lossy compression, after recovery there are negligible difference present where lossless gives accurate image. Huffman encoding is very well known, which can provide optimal compression and decompression without error (55:148 Digital Image Processing, 2017). The basic idea of Huffman coding is to represent data by number of variable size, where more frequent info being represented by shorter number (55:148 Digital Image Processing, 2017). Currently the Lempel-Ziv (or Lempel-Ziv-Welch, LZW) algorithm for dictionary-based coding has got attention as a better compression algorithm (55:148 Digital Image Processing, 2017).

As Genetic algorithms are types of computational models which get inspired from genetic evolution (Whitley, 1994). The way these algorithms work is that they encode all the probable solutions on a simple data and use these data to mix between them and some random change and calculate probability of solution from these data (Whitley, 1994).

3. Related works

In 2008, Roger Johansson was able to regenerate a Mona Lisa image from random sampling (Roger Johansson, 2017). It uses a genetic algorithm to model a population of individuals, each containing a string of DNA which can be visualized in the form of an image (Grow Your Own Picture Genetic Algorithms & Generative Art, 2017).

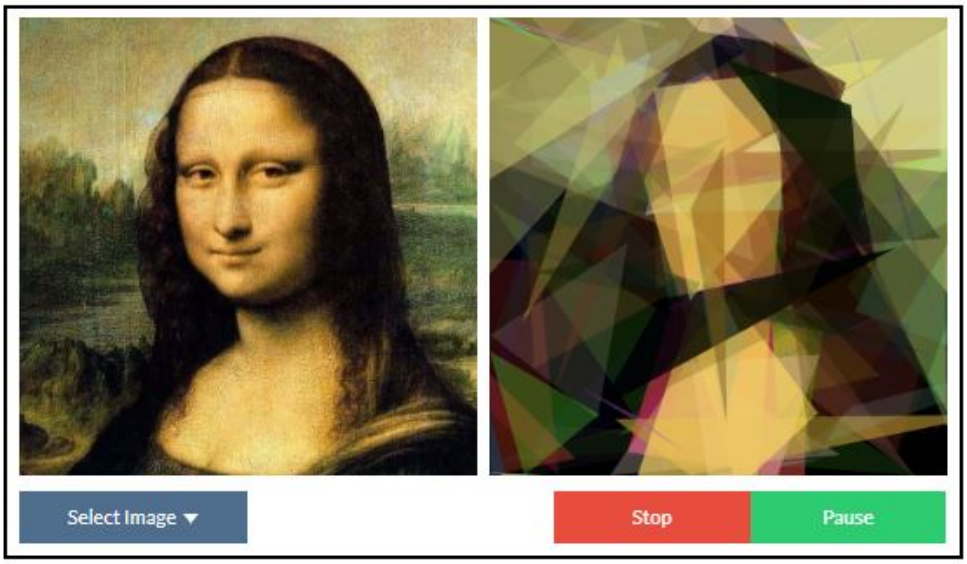


Figure 4. Image regeneration using GA

By starting with a population consisting of a randomly generated gene pool, each individual is compared to the reference image (the one on the left), and the individuals can then be ranked by their likeness to it, known as their "fitness", with the best fit being displayed on the output image (the one on the right) (Grow Your Own Picture Genetic Algorithms & Generative Art, 2017). By breeding the fittest individuals from the population, the DNA which produces the most accurate representation of the reference image is selected over successive generations, effectively demonstrating the power of a natural selection process to produce the best candidate for any given environment (Grow Your Own Picture Genetic Algorithms & Generative Art, 2017).

In 1992, a structured genetic algorithm was applied for automatic image registration of digital images (Dipankar Dasgupta, 1992). In 2001, Gradient based genetic algorithms in image registration were applied (Igor & Maslov, 2001). In 2001, genetic clustering was applied for image classification (Bandyopadhyay, 2002). Also, Genetic programming is successfully implemented for pattern recognition, control, planning and the generation of neural networks (Koza, 1992). It was also proved that genetic algorithm can work for content-based image retrieval (Ricardo da S. Torres, 2009) (Luca Piras, 2017). Genetic programming it is now also widely used for image clustering techniques (Ujjwal Maulik, 2000).

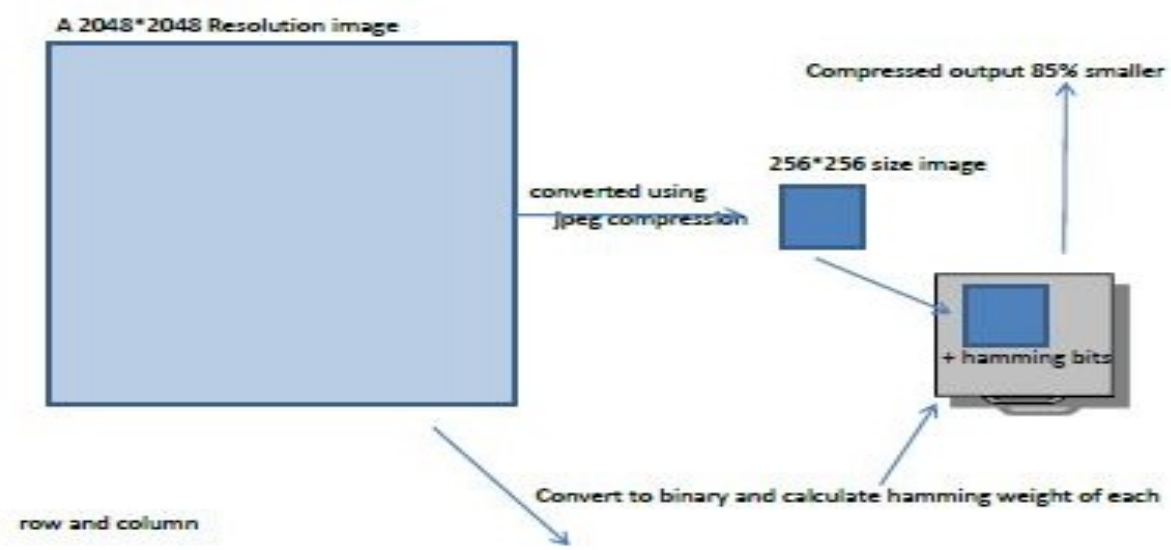
4. Our Proposal

Our proposal is to store each column and row bits count in a separate file and used that to reproduce the image using genetic algorithm.

Figure 5. Image down sample

If we take 10% of an image size and the row and column image hamming bit count our total size will be approximately below 15% of the actual image size. We proposed a method to reproduce original image from using this 15% information.

4.1. Methodology Formulation



	4	6	0	4	2	1	5	7	2	1
6	1	1	0	1	0	1	0	1	1	0
5	1	1	0	0	0	0	1	1	0	1
3	1	0	0	1	0	0	0	1	0	0
5	0	1	0	0	1	0	1	1	1	0
2	0	0	0	1	0	0	1	1	0	0
4	1	1	0	0	1	0	0	1	0	0
3	0	1	0	0	0	0	1	1	0	0
3	0	1	0	1	0	0	1	0	0	0

Figure 6. Methodology

Our method will first resize the image using normal image resizing option provided by operating system or standard library and attach the extra 2 array of data which contains no of 1 in original image in each row and column. Also, the total no of 1 in that image will be present too.

Now we will use genetic algorithm which will use this information to reproduce the image in original size.

4.2. Methodology Steps

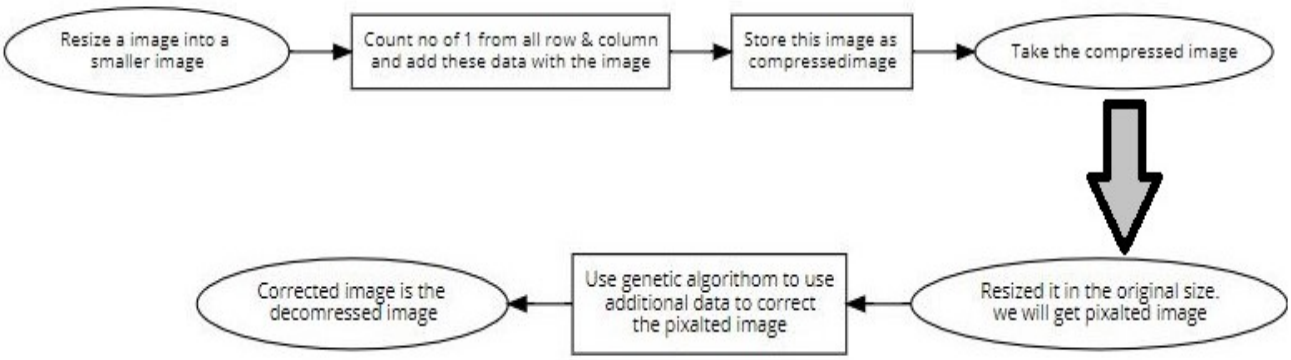


Figure 7. Basic Methodology

As in figure 7 we are storing the extra data which is look like figure 8. Where a 20*20 size image of alphabet ‘A’ data has been stored. When we regenerate image, we are using these data.

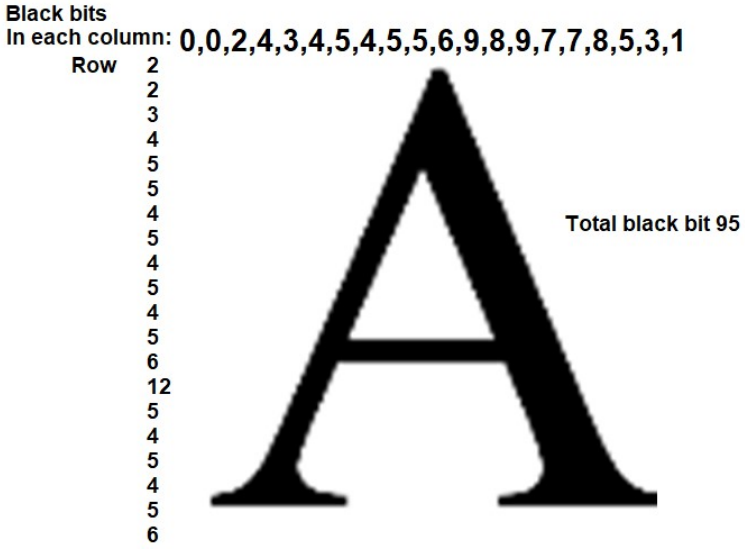


*Figure 8. Sample Data Extraction for a 20*20 size image*

5. Data extraction technique

For the extraction, our goal is to divide an image into smaller blocks and keep the row and column data for these blocks. But for our experiment we used a single block, which means taking the full image as a single block. For bigger image we should always divide the image in separate blocks and work on them par rally. As in figure 8, after extracting the data we can add the row and column bits information in the resized image or saved in a separate file. For proof of concept we saved it in a text file. And later that file is used to feed GA to make the fitness function, in figure 9 an extraction has been shown. In upper and side textbox containing the information which later is saved in a text file.

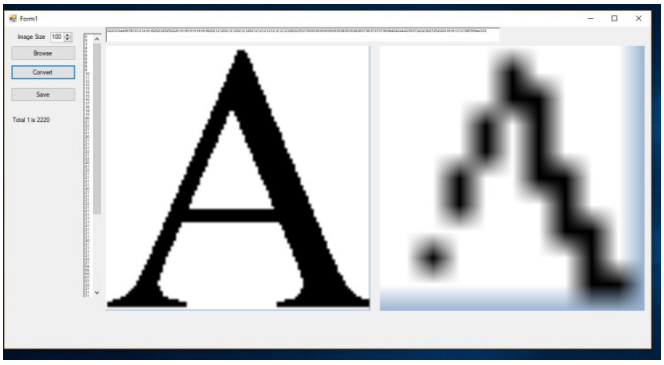


*Figure 9. Resized image and extracted data for a 100*100 size image*

6. Image Regeneration

First, we have to convert the pixelated small image to the original size image, and then divide these into the blocks as it done in extraction time. Then we have to add or subtract random bits from each of these blocks to equal each block hamming bit to original hamming bit number. And then GA is applied to match these randomness to original image hamming bits in per row and column. In figure 10 there is a 4-block example which regenerates randomly using total block bit count. Now we will try to match their row and column bits of information with the extracted data which is described in later section.

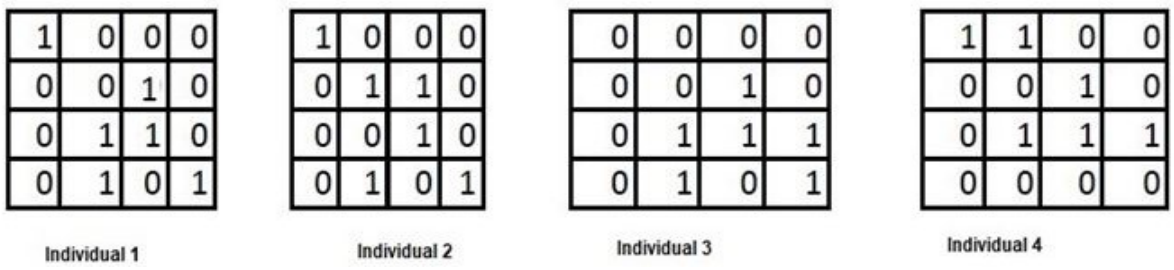


Figure 10. Block example

7. GA implementation

We use genetic algorithm with tournament selection method. We used a 3-point crossover and dynamic mutation rate between 15% to 30%. We had 4 fitness functions, all of them had different weight to calculate total fitness of each individual.

7.1. Algorithm

1. *Image Regeneration (image data)*
2. *Get from data,*
3. *Number of black bits in a block is m*
4. *Number of black bits in p number of row, row[p]*
5. *Number of black bits in q number of column, column[q]*
6. *Randomly generate N number of images name population[N], which contain m number of black bits placed randomly.*
7. *For each image I_i in population*
8. *Get fitness F_i from I_i*
9. *If any fitness F_i is near 0*
10. *return I_i*
11. *Select N/2 number of I with best fitness using tournament method*
12. *Do Crossover to generate N/2 offspring using selected N/2 images*
13. *Do mutation on N images*
14. *Go to step 7*

7.2. Encoding

For each block We will create a population with random n*n size blocks where no of 1 is equal to provide no of 1s for that block. Genotypes are 1 and 0 from pixel bit conversion. Phenotypes are each block matrix. So, an image will be encoded to a single string of its height*width size.

7.3. Population

We generated M number of images which act as a single individual in population. Each individual has the same number of bits present in the original image block. We set this bit completely randomly. For each block we will have to create a different population set which can run in different process.

7.4. Fitness

It is a multi-objective genetic algorithm.

We will have four objectives for each individual.

So, we will define four fitness function as:

- $F_r(r)$ = Calculate difference of rows (r_d) from provided blocks rows data (O_{dr}).
- $F_c(c)$ = Calculate difference of columns(c_d) from provided blocks column data (O_{dc}).
- $F_b(b)$=Calculate difference of total bit T_b in block from provided blocks data. O_d
- $F_x(X)$=Calculate difference of bits X_d from pixelated image block X_{pd}

Now let p and q, height and width of the image.

$$Fr(r) = \sum_{i=0}^{p} [(Odri - rdi])$$ (1)

$$Fc(c) = \sum_{i=0}^{q} [(Odci - cdi])$$ (2)

$$F_b(_b) = |T_b - O_d|$$ (3)

$$Fx(x) = \sum_{i=0}^{p*q} [(Xpdi - xdi])$$ (4)

I. Penalty function

We applied penalty value when each row and column difference go certain extent.

$$\text{Let calculate } R_{davg} = \frac{\sum_{i=0}^{p} [(Odri - rdi])}{p}$$ (5)

$$\text{And } C_{davg} = \frac{\sum_{i=0}^{q} [(Odci - cdi])}{q}$$ (6)

For F_r penalty parameter P_{pr},P_{ppr} and P_{npr}

For each R_{di}

$$F_r(r) = F_r(r) + P_{ppr} \quad \text{If } R_{di} > P_{pr} * R_{davg}$$ (7)

$$F_r(r) = F_r(r) + P_{npr} \quad \text{If } R_{di} < P_{pr} * R_{davg}$$ (8)

Similarly,

For F_c penalty parameter P_{pc},P_{ppc} and P_{npc}

For each C_{di}

$$F_c(c) = F_r(c) + P_{ppc} \quad \text{If } R_{di} > P_{pc} * R_{davg} \qquad (9)$$
$$F_c(c) = F_r(c) + P_{npc} \quad \text{If } R_{di} < P_{pc} * R_{davg} \qquad (10)$$

II. Total fitness

If weight for each fitness function are W_r, W_c, W_b, W_x where $W_r \approx W_c$ and $W_b \ll W_x \ll W_r$
Total fitness $F_t(T) = F_r(r)* W_r + F_c(c)* W_c + F_b(b)* W_b + F_x(X)* W_x$ (11)

III. Selection scheme

We used the tournament method for selection. We run n/2 round tournament each size is n/t_n . And use n/2 round winner as selected individual for crossover.

IV. Crossover and mutation

We used a 3-point crossover in each champion from tournament selection and produce n/2 number of offspring and merge these champions and offspring for new set of population. In this way, the population size always remains constant. We used multi-point which are randomly selected mutation.

7.5. Terminating condition

When last N number of generation has same average fitness, we are terminating the GA, and it shows the output.

8. Result Analysis and Discussion

In figure 11 it is the initial population showed and figure 12 the population started to change and figure 13 we reached a convergence.

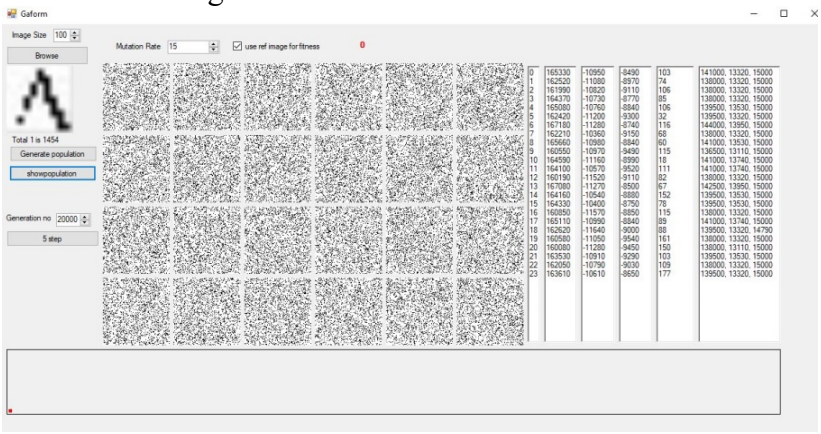


Figure 11. Initial population

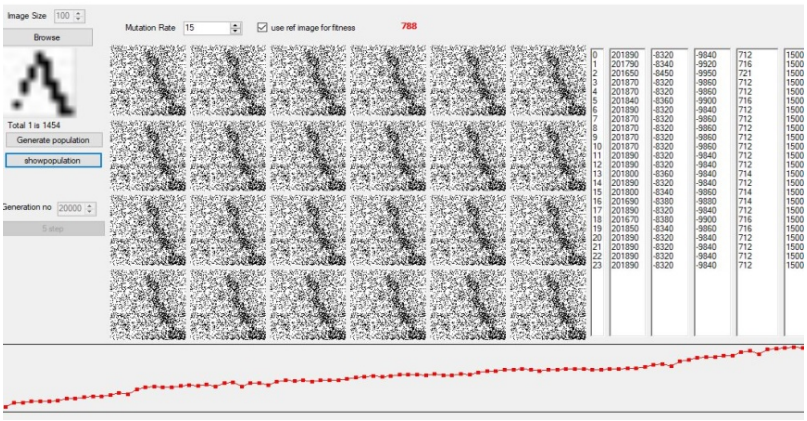


Figure 12. After few generation

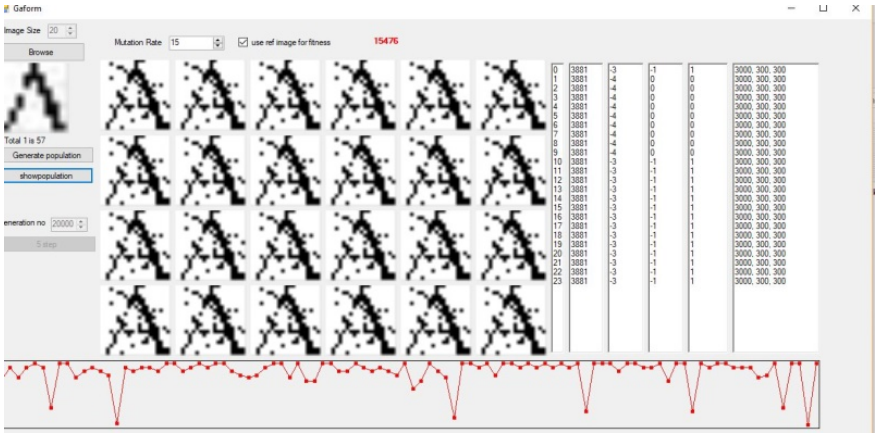


Figure 13. Reached convergence

As we can see, here we were able to recover the lost middle portion of character 'A' using our genetic algorithm. If we can apply some noise filtering technique, the result would be far better. In figure 14, 15 there are another two examples

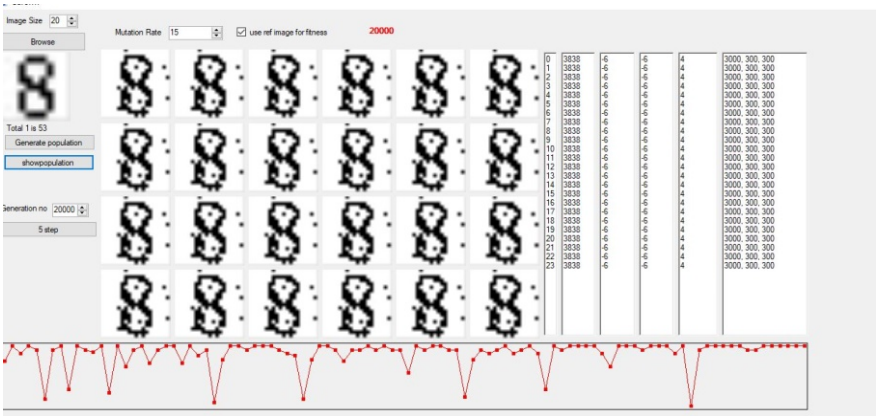


Figure 14. Symbol 8 regeneration

In Figure 15 we were able to generate the symbol H without any help from a small image we used only for row and column data.

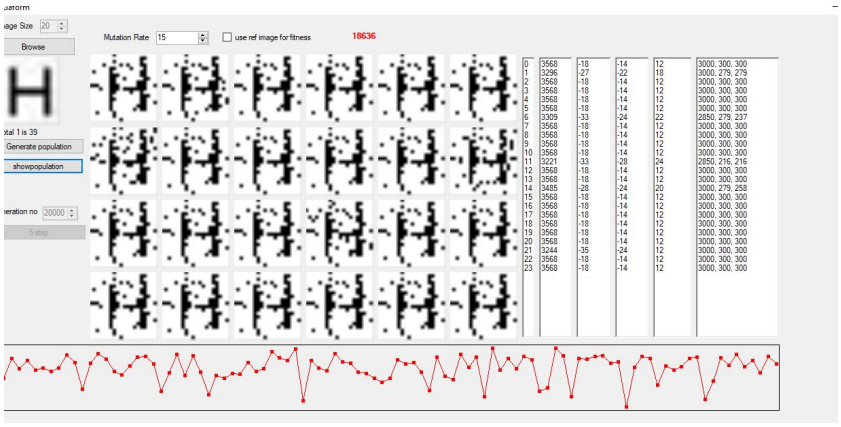


Figure 15. Symbol H regeneration (without reference pixelated image)

From the result analysis, it seems that if we make a smaller block we can produce image more faster and more accurate to original. Also, its seldom that we hit the local optima and stuck

with not good enough version. Changing mutation rate that time gives us better result. It seems we need to apply dynamic penalty method and mutation rate to tackle this issue.

For some images there is a chance to get stuck where fitness function maxed, but we are not near to the original image like in figure 16, both row and column fitness matched. But image lost a key portion from original image, in these cases we should increase the weight of fitness function $F_x(X)$, which will solve the issue.

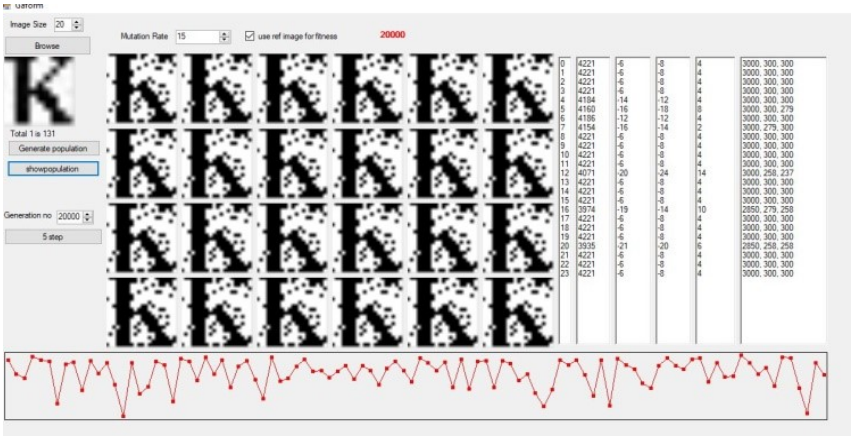


Figure 16. Failed Generations

Also, we used very small data set to test and only used black and white. But as every image can convert in 0 and 1 as binary, our method should have to work for these too.

As every file can converted to 0 and 1, whatever the format the file is, this technique has huge application for not only image files, but for every kind of file. Specially in error bit recovery, this technique could be helpful. If any data block transfer contains its row column bit data as redundant data and data block gets damaged somehow, we could be able to reproduce the data block used this algorithm. As this recovery can be done in smaller parts, it will be very easy and faster to do it in distributed system. Many cloud systems can use this technique to transfer data between each cloud, thus the network congestion will get reduced.

8. Conclusion

We used genetic algorithm to recover an image from its small scaled image. Our method has scalability to deployed in any distributed system and can work faster. Using some more constraints and filtering this algorithm can provide better results. Also, the same technique can also be applied to any file system. This procedure can really improve the reduction of noise in QR code better than the (Gupta, 2018) had done before; when using parallel computing, time overhead cost will be reduced and the process can generate a near perfect reconstruction fast. In the future we will try it to run in distributed computer by dividing it into different blocks and then merge them together. That way it will be faster than now and will be a real application. Also, we will try to implement in Media files such as audio and video, and for binary bits data recovery. If some data gets lost we can recover these data using this algorithm, in that case we will have more guiding parameters as some original data would be present. So, success rate would be higher.

References

55:148 Digital Image Processing. (2017, December 06). Retrieved from 55:148 Dig. Image Proc. Chapter 12: http://user.engineering.uiowa.edu/~dip/lecture/DataCompression.html.

Grow Your Own Picture Genetic Algorithms & Generative Art. (2017, December 06). Retrieved from https://chriscummins.cc/s/genetics/.

Bandyopadhyay, S. a. (2002). Genetic clustering for automatic evolution of clusters and application to image classification. Pattern recognition 35, no. 6 , 1197-1208.

Dipankar Dasgupta, D. R. (1992). Digital image registration using structured genetic algorithm. Proc. SPIE 1766, Neural and Stochastic Methods in Image and Signal Processing.

Igor V. Maslov, I. G. (2001). Gradient-based genetic algorithms in image registration. Proc. SPIE 4379, Automatic Target Recognition XI.

Johansson, R. (2017). Genetic Programming: Evolution of Mona Lisa. Retrieved from https://rogerjohansson.blog/2008/12/07/genetic-programming-evolution-of-mona-lisa/

Gupta, K. D., Ahsan, M., Andrei, S. (2018). Extending the Storage Capacity And Noise Reduction of a Faster QR-Code. Broad Research in Artificial Intelligence and Neuroscience, Vol. 9, Issue 1, 59-71.

Koza, J. R. (1992). Genetic Programming. Cambridge, MA: MIT Press.

Luca Piras, G. G. (2017). Information fusion in content based image retrieval: A comprehensive overview. Information Fusion, Vol. 37, 50-60.

Perret, E. (2017). Here's How Many Digital Photos Will Be Taken in 2017. Retrieved from https://mylio.com/true-stories/tech-today/how-many-digital-photos-will-be-taken-2017-repost.

Torres, R. S., Falcão, A. X., Gonçalves, M. A., Papa, J. P., Zhang, B., Fan, W., Fox, E. A. (2009). A genetic programming framework for content-based image retrieval. Pattern Recognition, Volume 42, Issue 2, 283-292.

Maulik, U., Bandyopadhyay, S. (2000). Genetic algorithm-based clustering technique. Pattern Recognition, Vol. 33, Issue 9, 2000, Pages , ISSN 0031-3203, 1455-1465.

Whitley, D. (1994). A genetic algorithm tutorial. Statistics and Computing, Vol. 4.

Kishor Datta GUPTA was born in Chittagong, Bangladesh in 1989. He received his Bachelor of Science in Computer Science and Engineering (CSE) in 2011 from Khulna University of Engineering and Technology (KUET), Bangladesh and Masters of Science in Computer Science in 2017 from Lamar University, Texas, USA. he is currently working as a research assistant and continuing his Ph.D. in Computer Science Department of University of Memphis, USA. His research interest includes Block chain, Evolutionary Computation

Sajib SEN was born in Chittagong, Bangladesh in 1992. He received his Bachelor of Science in Electrical and Electronic Engineering(EEE) in 2014 from Khulna University of Engineering and Technology (KUET), Bangladesh. He worked as a Lecturer in Electrical and Electronic Engineering Department of Prime University, Bangladesh from 2015 to 2017.Currently, he is working as a graduate assistant and continuing his Ph.D. in Computer Science Department of University of Memphis, USA. His research interest includes machine learning, cyber security and cyber-physical system.