Optimization Techniques for Image Processing

Prerak Chapagain
University of New Orleans

Follow this and additional works at: https://scholarworks.uno.edu/honors_theses

Part of the Electrical and Computer Engineering Commons

Recommended Citation
https://scholarworks.uno.edu/honors_theses/133

This Honors Thesis-Unrestricted is brought to you for free and open access by the Undergraduate Showcase at ScholarWorks@UNO. It has been accepted for inclusion in Senior Honors Theses by an authorized administrator of ScholarWorks@UNO. For more information, please contact scholarworks@uno.edu.
OPTIMIZATION TECHNIQUES FOR IMAGE PROCESSING

An Honors Thesis

Presented to

The Department of Electrical Engineering

of the University of New Orleans

In Partial Fulfillment

of the Requirements for the Degree of

Bachelor of Science, with University High Honors

and Honors in Electrical Engineering

by

Prerak Chapagain

April 2019
Acknowledgements:

Foremost, I would like to thank my advisor Dr. Dimitrios Charalampidis for helping me narrow down my focal interest and for letting me work on optimization in image processing while realizing that I had no background on these areas. His constant guidance and valuable feedback helped me gain insights that were essential in completing this thesis. His open door policy and his frankness made it convenient for me to talk to him anytime regarding any issues I had with my code. Despite his position as the Department Chair, he did not hesitate putting aside time for discussions which helped me progress with my thesis.

I would also like to thank Dr. Huimin Chen for being a valuable resource in helping me realize the applications of my work clearer. It was when I took his Controls class that I realized my interest in Optimization techniques. His willingness to let me borrow his books and always being open to talking about my thesis progress was a great support.

I would also like to thank Dr. Edit Bourgeois and Erin Sutherland for constantly asking me about my progress and motivating me to work harder. Finally, I would like to thank my parents and my sister for their constant encouragement to strive for academic excellence that has helped me come this far, and which will help me to achieve even more in the future.
# Table of Contents

Acknowledgements: ii  
List of figures: iv  
Abstract: vi  
Introduction: 1  

Convex Optimization: 3  
  Introduction to Convex Optimization: 3  
  Mathematics of Convex Optimization: 4  

Genetic Algorithms: 7  
MATLAB application using Genetic Algorithm: 9  
  Roulette Wheel Selection method: 11  
  Crossover: 13  
  Mutation: 14  

Stage 1 Results: 17  
Object matching in MATLAB using GA: 21  
Stage 2 Results: 25  
Stage 3 Results: 30  
Stage 4 Results: 34  
Conclusion/Future work: 37  
Bibliography: 39
List of figures:

1. Figure 1: Image blurring (Esser)
2. Figure 2: Convex Graph (Esser)
3. Figure 3: Roulette Wheel concept
4. Figure 4: MATLAB script snippet for Roulette Wheel selection
5. Figure 5: MATLAB script snippet for crossover
6. Figure 6: Local and Global maxima before mutation
7. Figure 7: MATLAB script snippet for mutation
8. Figure 8: Local and global maxima after mutation
9. Figure 9: Stage 1 result: Simulation 1
10. Figure 10: Stage 1 result: Simulation 2
11. Figure 11: Stage 1 result: Simulation 3
12. Figure 12: Stage 1 result: Simulation 4
13. Figure 13: Stage 1 result: Simulation 5
14. Figure 14: Original image
15. Figure 15: Image transformation
16. Figure 16: Stage 2: Process of every iteration
17. Figure 17: Stage 2 result: Simulation 1
18. Figure 18: Stage 2 result: Simulation 2
19. Figure 19: Stage 2 result: Simulation 3
20. Figure 20: Stage 2 result: Simulation 4
21. Figure 21: Stage 2: Original image with noise
22. Figure 22: MATLAB snippet for decaying mutation

23. Figure 23: Stage 3: Process of every iteration

24. Figure 24: Stage 3 result: Simulation

25. Figure 25: Stage 3: Overlap of the best solution with the original image

26. Figure 26: Stage 4: Original image with noise and other objects

27. Figure 27: Stage 4: Best parent of every iteration shown in blue asterisks

28. Figure 28: Stage 4: Overlap of the best solution with the original image
Abstract:

This research thesis starts off with a basic introduction to optimization and image processing. Because there are several different tools to apply optimization in image processing applications, we started researching one category of mathematical optimization techniques, namely Convex Optimization. This thesis provides a basic background consisting of mathematical concepts, as well as some challenges of employing Convex Optimization in solving problems. One major issue is to be able to identify the convexity of the problem in a potential application (Boyd). After spending a couple of months researching and learning Convex Optimization, my advisor and I decided to go on a different route. We decided to use Heuristic Optimization techniques instead, and in particular, Genetic Algorithms (GA). We also conjectured that the application of GA in image processing for the purpose of object matching could potentially yield good results.

As a first step, we used MATLAB as the programming language, and we wrote the GA code from scratch. Next, we applied the GA algorithm in object matching. More specifically, we constructed specific images to demonstrate the effectiveness of the algorithm in identifying objects of interest. The results presented in this thesis indicate that the technique is capable of identifying objects under noise conditions.

Key Words: Optimization, Convex Optimization, Genetic Algorithms, Image Processing, Object Matching
Introduction:

Optimization, as the name suggests, is a way to solve a problem by tuning a set of parameters in order to achieve an optimal solution towards a defined goal. For example, one may be interested in finding the maximum or the minimum value of a function. The function could be associated with a real-life problem, and the possible applications are endless. For instance, the function could represent the profit of a company, which one would like to maximize, or it could represent the expenses, which would need to be minimized. The purpose of using optimization techniques would be to tune certain parameters on which the profit and the expenses depend, so that the function maximization or minimization is possible. For our application, we decided to use optimization for an image processing application, namely pattern matching.

Image processing is a large area, which includes many different research subfields. Image processing essentially refers to a large collection of techniques and algorithms which, among other goals, attempt to extract useful information from images, find points and areas of interest within images, convert images to more efficient representations, and improve visualization of information. To be more specific, popular subfields of image processing include pattern recognition, object matching, image blurring, image compression, edge detection, and image restoration. For instance, an application of image processing is the processing of pictures obtained from satellites which could be partially damaged and may be missing information (Mary). In general, images can be analog or digital. Most recent applications involve the use of digital images. Digital images can be thought of as rectangular arrays consisting of a number of values, and each location in the array is called a pixel.
There is a vast number of different optimization techniques ranging from heavily mathematical optimization techniques to Heuristic Optimization methods. Most of these techniques find applications in image processing. One of the mathematical optimization techniques that interested me was Convex Optimization. After spending a couple of months studying Convex Optimization and trying to identify a potential image processing application, my advisor and I decided to go on a different route and use Genetic Algorithms (GAs) to solve an object matching problem. We were curious as to how successful results GAs can yield. The algorithms were written from scratch using MATLAB. After implementing the basic GA algorithm, we then formulated the object matching problem so that GAs can be employed.

In what follows, this thesis first presents a review of Convex Optimization, followed by a review of GAs. Then, the method used in this thesis is introduced, and experimental results are presented. Finally, the thesis closes with some concluding remarks and possibilities of future work.
Convex Optimization:

Introduction to Convex Optimization:

Optimization can be used in fields such as controls, signal processing, circuit design, communications, and machine learning. Besides engineering, optimization can be used in other areas, such as in finance and economics with the purpose of finding the maximized profits and returns. Basically, it can be used in any problem where we are interested in finding the minimum cost in a function, or where we want to know what course of action is the best in order to solve or analyze a problem.

In particular, Convex Optimization (CO) is one of the popular and heavily mathematical optimization techniques employed in engineering applications. Working with CO requires knowledge of advanced calculus, linear algebra, and some probability theory. The first step in the process of applying CO is usually the identification of the problem. Then, it is necessary to express the problem mathematically. In particular, a cost function associated with the problem should be obtained, so that one can check if the problem is convex or not. In general, the cost function may be convex, concave, or affine. In order to apply CO, the cost function of the particular problem should be convex, as will be described in more detail later in this thesis in the mathematics of Convex Optimization section. Although there are some non-convex optimization techniques, the first part of this thesis only focuses on functions which satisfy or which at least approximately satisfy convexity. Even within CO, there are special subclasses of optimization problems, such as linear programming and least squares (Boyd).
Since we decided on exploring an image processing topic, we started looking at how optimization is used in different image processing applications. One such application is image blurring. Figure 1 illustrates how CO-based method was used in order to obtain an approximation of the original image from a blurred version of the image (Esser).

![Figure 1: Image blurring (Esser)](image)

**Original image**  **Blurry/Noisy**  **Recovered**

Mathematics of Convex Optimization:

This section goes in some detail about the background and mathematics of CO that were gathered from our study through online resources. Basically, as was described earlier in this thesis, the objective of optimization is to minimize or maximize a function as much as possible, while considering any restrictions that might be present. The mathematical way to describe the CO concept is presented next:

Minimize $f_0(x)$ subject to

$$f_i(x) \leq b_i, \quad i = 1, \ldots, m$$

(Boyd)
where $m$ is number of constraints. Here we want to minimize our function $f_o$ where $x$ is a vector consisting of the optimization variables or a single optimization variable.

A function is convex if for any straight line drawn between two points of the function starting at $x_1$ and ending at $x_2$, all points of the function between $x_1$ and $x_2$ are below the line. Mathematically, this concept is expressed as follows:

$$f((1-g)x_1 + g x_2) \leq (1-g)f(x_1) + g f(x_2) \quad \text{(Boyd)}$$

The concept is also illustrated in Figure 2.

![Convex Graph](image)

Figure 2: Convex Graph (Esser)

One of the obstacles that may arise while identifying an optimization problem is if the function is not linear and also not known to be convex. Therefore, convexity is often compromised, and it is
assumed that the problem is convex around the location of the best solution. This assumes that we need an initial guess, or multiple initial guesses, which may lead us close to the best solution.
Genetic Algorithms:

Genetic Algorithms (GAs) is a class of heuristic optimization techniques. A characteristic of heuristic optimization techniques, when compared to other optimization techniques, such as CO, is that they are based on randomness. Randomness may improve the capability of searching for the overall (global) best solution by avoiding locally good solutions in compromise of accuracy. There are different types of heuristic optimization techniques, such as Swarm Intelligence, and Tabu Search, among others (Heuristic Algorithms).

GA is one of those algorithms which, as the name suggests, attempts to model optimization using concepts from genetics. These algorithms assume that many candidate solutions are available, and each candidate solution is a member of the available population. Each member in the population is a parent, and each parent has an associated chromosome of a particular size. The chromosomes are essentially vectors containing the optimization parameters.

GAs are iterative methods. In each iteration step, the parents are modified using crossover and mutation operations. Crossover essentially combines two different parents to produce new off-springs (which in turn become the parents in the next iteration). Mutation alters a parent in some random way. Each parent also has a “fitness”. The fitness indicates how “good” the solution is. Based on the fitness, a selection method is applied in order to pick the better fit parents more often. Then crossover and mutation is applied to get another set of “healthier” parents, and the process continues for a number of iterations, until a desired result is obtained.

One advantage of GAs and other heuristic algorithms is that they do not require as strict mathematical formulation of a problem as with some other methods. Arguments against using GAs
state that the method is not reliable, not exactly accurate, and requires significant computation time, especially as the dimensions increase. Additionally, using a large population, or using a large number of iterations might slow down the process of obtaining the solution even further. Nevertheless, GAs have been found to be really useful tools for many scenarios.

We used MATLAB’s scripting language to implement GAs. Although MATLAB offers some pre-built functions with the GA toolbox, we decided to build our own code from the scratch. The following section provides more details regarding our GA implementation.
MATLAB application using Genetic Algorithm:

We first decided on writing a genetic algorithm in MATLAB from scratch without using any GA or any other optimization toolbox. The problem statement is to find the maximum value of any given function with any given parameters. For example, if we have a first order function with 2 variables, \( f(x_1, x_2) = 3x_1 - 2x_2 \), where \( x_1 \) and \( x_2 \) are valued between 0 and 1, then the maximum value of that function is going to be 3, and this is for when \( x_1 = 1 \) and \( x_2 = 0 \). This seems obvious when using such a simple function. However, for a higher order polynomial, for more variables, and for a wider range of these variables, the situation becomes more complicated. The maximum of a function such as \( f(x_1,x_2,x_3,x_4,x_5) = (3-x_1^2+2x_2-3x_3-x_4^3+x_5^2) \) where \( x_1, x_2, x_3, x_4, x_5 \) range from -5 to 5 might not seem obvious at the first sight. However, upon close inspection, we can realize that the best solution for obtaining the maximum value of the function is when \( x_1=0, x_2=5, x_3=-5, x_4=-5, x_5 = 5 \) (or -5).

We tried obtaining a known solution by using the GA in order to confirm the correctness of the algorithm, so that we could then apply the GA to obtain the maximum of any other given function. First, we started with the function which has only 2 parameters, and then we also tried to find the solution for the function which has 5 parameters.

Firstly, we initialized all variables needed for the function:

\[
 f(x_1,x_2,x_3,x_4,x_5) = (3-x_1^2+2x_2-3x_3-x_4^3+x_5^2)
 \]

Since there are 5 variables, the size of each chromosomes in the GA is 5. The order of the equation does not affect the size of the chromosomes in the GA. Then, we choose a number of parents. Parents refer to randomly chosen numbers for the values of the 5 variables. A total of \( n \)
parents was chosen. This implies that we randomly generate \( n \) vectors in MATLAB (each of size 5) to form a matrix of size \( n \times 5 \) (for \( n \) parents and for chromosomes of size 5). Now, the objective lies in manipulating this randomly generated \( n \) parents in such a way so that we can search for the best solution that provides the best (maximum value) for the function defined above. By iteration we refer to how many times we “manipulate” the parents to produce a new generation of parents. This process imitates the natural hereditary process where the best off-springs are chosen based on the “survival of the fittest” concept, after going through some mutations and crossovers. In the following sections, we will go through every step along with references to the code written for this thesis.

After choosing the parents, the next step is to calculate the level of fitness for each parent. The higher the fitness, the higher the chances should be that the parents can provide a new generation of parents with high fitness. Since fitness for our example refers to the maximum value of the function, it is simply calculated by plugging in values for those \( n \) randomly chosen parents (chromosomes of size 5) to get \( n \) corresponding numbers, namely their fitness. Note that we keep the maximum fitness considering all parents and store it for each iteration. Then, the challenge lies in how exactly to choose parents based on the fitness. There are different selection methods which can be employed. Some of the popular ones are Rank Selection, Tournament Selection, Boltzmann Selection, and Roulette Wheel selection (Saini). We chose the Roulette Wheel selection method for this thesis.
**Roulette Wheel Selection method:**

Like the name suggests, this method is an accurate representation of the roulette wheel we see in Casinos. The Pi chart in Figure 3 presents a roulette wheel with bias (preference) on Parent 1 and Parent 2.

![Figure 3: Roulette Wheel concept](image)

As expected, when the wheel is spun, it is more likely for the pointer to fall on Parent 1 or Parent 2 compared to rest of the parents. The percentage of likeliness or bias for the pointer to be pointed at the “fit” parents is determined by the fitness, as we described it earlier. A larger fitness corresponds to a higher chance for that particular parent to be chosen. “The principle of roulette selection follows a linear search through a roulette wheel with the slots in the wheel weighted in proportion to the individual’s fitness values. The probability of an individual being selected as a parent for crossover is given by” (Saini),

\[ p(i) = \frac{f(i)}{\sum f} \]
where $f$ represents the fitness function. This implies we need to normalize the $n$ fitness values we had, and then apply the cumulative sum (cumsum) to obtain the roulette wheel values. The code snippet in Figure 4 shows how we have implemented the Roulette Wheel Selection in MATLAB after finding the cumulative sum.

```matlab
for j=1:n
    random_num_again=rand; % generating a random number
    fitness_prob_index=find(fitness_prob>random_num_again);
    fitness_prob_index=min(fitness_prob_index);
    for roulette_index=1:chromosomes
        new_population(j, roulette_index)=b(fitness_prob_index, roulette_index);
    end
end
```

Figure 4: MATLAB script snippet for Roulette Wheel selection

We generate a random number (between 0 and 1) which acts as the pointer in the Roulette Wheel. After that, we compare that number with the cumsum array to find where it lies in the Roulette Wheel. Since the parents with bigger fitness correspond to a larger wedge in the wheel, the chances that the fit parents will be chosen are higher. The same process is repeated over $n$ times (the number of parents) to get a new population. It is apparent that the parents with “higher fitness percentage”, as shown in Figure 5, will be chosen more frequently than the ones with low fitness. Now we have a new population of $n$ parents. The next step is the genetic modification of crossover.
**Crossover:**

Genes of one parent crossover with the ones of the other parent to produce off-springs, as we know from biology. We use the same concept to crossover the new parents that we obtained through the Roulette method with one another. Crossover is one of the manipulations, as mentioned earlier in this thesis. The ultimate goal is to get the best fit solution by trying different possible ways of mixing the parents randomly.

For crossover to happen, a part of the parent needs to be crossed-over with a part of another parent. We have 5 genes in each chromosome and \( n \) new parents, and we wrote the code in such a way that every new parent is randomly crossed-over. Figure 5 shows the code snippet of the crossover algorithm.

```matlab
for i=1:n
    i_r_c=zeros(chromosomes, 1);
    for i_r=1:chromosomes
        i_r_c(i_r)=round((n-1)*rand+1);
    end
    for new_population_index=1:chromosomes
        c(i, new_population_index)=new_population(i_r_c(new_population_index), new_population_index);
    end
end
```

Figure 5: MATLAB script snippet for crossover

The number of times crossover is going to take place is defined by \( n \), which is the number of parents. Then, another for loop (i_r=1:chromosomes) is included to create an array of random indexes for each gene in the chromosome. Essentially, each new parent is created by the random assignment of genes in a chromosome, taken from different parents randomly. When the crossover is done, we move along to apply the mutation.
Mutation:

Mutation has been an essential evolutionary trait to help to provide enough variability in a population. We use the same hereditary concept in our GA to manipulate our parents. We use mutation by adding or subtracting some values from the parents. Those numbers are generated randomly. One might wonder if adding a random number to a set of fit solutions might compromise lowering their fitness. However, the advantages of mutation can be understood by observing the example of Figures 6 and 8.

Figure 6: Local and Global maxima before mutation
As we can observe in Figure 6, all current solutions (parents) have concentrated around the peak to the left. The left peak is what we would call a local maximum, because there is another peak of a higher value. If we do not apply mutation, all our solutions will be concentrated around this local maximum. However, applying some mutation allows the parents to move away from the local maximum and to “realize” that there could be a better solution. Implementing mutation in the MATLAB script of our problem statement is fairly simple, as can be seen in the code snippet of Figure 7.

```
mutation_raw=rand(n, chromosomes);
mutation=random_lower_m+((random_upper_m-random_lower_m).*mutation_raw);
mutated=c+mutation;
mutated(mutated>random_upper)=random_upper;
mutated(mutated<random_lower)=random_lower;
b=mutated;
```

Figure 7: MATLAB script snippet for mutation

Randomly generated numbers are scaled between a certain range to add to the parents, after crossed-over has been applied, to account for mutation. In ensure that the new gene values do not go out of range after mutation, the values are capped using upper and lower limits.

After applying mutation, we might reduce the risk of having our solutions concentrated just around the local maxima, and we can observe a scenario similar to the one presented in Figure 8. In Figure 8, it is apparent that the global maximum peak has been detected by at least some of the possible solutions.
Figure 8: Local and Global maxima after mutation
Stage 1 Results:

After the parents are mutated and a new population is obtained, the processes of Roulette Wheel, crossover, and mutation are repeated for \( n\_iterations \) (number of iterations) to get a new population in each iteration. Every best fitness is stored in an array for each iteration, in order to find the best of all the best finesses and its corresponding parent at the end.

After trying different values of \( n \) (parents) and \( n\_iterations \) (iterations), it became apparent that increasing \( n \) and \( n\_iterations \) also increased the accuracy of solution while, however, increasing the computational time too. The results for different values of \( n \) and \( n\_iterations \) are presented in the figures next. The expected highest fitness is 178, and the best parent which provides this fitness is \( x = [0, 5, -5, -5, 5 \text{ (or } -5) ] \).

1. When \( n=10, n\_iterations=10 \)

\begin{verbatim}
9 - chromosomes=5;
10 - n=10;               % number of parents
11 - b_raw=rand(n, chromosomes); % array for randomly
12 - b=random_lower+((random_upper-random_lower).*b_raw); %array for randomly
13 - n_iterations=10;         %number of iterations

Command Window
New to MATLAB? See resources for Getting Started.

best_fitness =

164.1459

best_parent =

1.8435  -0.2279  -5.0000  -5.0000  -5.0000
\end{verbatim}

Figure 9: Stage 1 result: Simulation 1
2. When \( n=10, \text{ } n\_iterations=100 \)

\[
\begin{align*}
0 & \quad \text{n}=10; \quad \% \text{ number of parents} \\
1 & \quad \text{b\_raw} = \text{rand(n, chromosomes)}; \quad \% \text{ array for randomly} \\
2 & \quad \text{b} = \text{random\_lower} + ((\text{random\_upper} - \text{random\_lower}).*\text{b\_raw}); \\
3 & \quad \text{n\_iterations} = 100; \quad \% \text{number of iterations}
\end{align*}
\]

Command Window

New to MATLAB? See resources for Getting Started.

\[
\begin{align*}
\text{best\_fitness} &= \\
&= 171.4540 \\
\text{best\_parent} &= \\
&= 1.3172 \quad 5.0000 \quad -5.0000 \quad -5.0000 \quad -4.4932
\end{align*}
\]

Figure 10: Stage 1 result: Simulation 2

3. When \( n=100, \text{ } n=10 \)

\[
\begin{align*}
9 & \quad \text{chromosomes}=5; \\
10 & \quad \text{n}=100; \quad \% \text{ number of parents} \\
11 & \quad \text{b\_raw} = \text{rand(n, chromosomes)}; \quad \% \text{ array for randomly} \\
12 & \quad \text{b} = \text{random\_lower} + ((\text{random\_upper} - \text{random\_lower}).*\text{b\_raw}); \\
13 & \quad \text{n\_iterations} = 10; \quad \% \text{number of iterations}
\end{align*}
\]

Command Window

New to MATLAB? See resources for Getting Started.

\[
\begin{align*}
\text{best\_fitness} &= \\
&= 177.6933 \\
\text{best\_parent} &= \\
&= 0.5538 \quad 5.0000 \quad -5.0000 \quad -5.0000 \quad -5.0000
\end{align*}
\]

Figure 11: Stage 1 result: Simulation 3
4. When $n=100$, $n_{\text{iterations}}=100$

```
9 - chromosomes=5;        % number of parents
10 - n=100;                % array for randomly
11 - b_raw=rand(n, chromosomes); % array for randomly
12 - b=random_lower+((random_upper-random_lower).*b_raw); % number of iterations
13 - n_iterations=100;     %number of iterations
```

**Command Window**

New to MATLAB? See resources for Getting Started.

```
best_fitness =
177.9995

best_parent =
0.0229  5.0000  -5.0000  -5.0000  -5.0000
```

Figure 12: Stage 1 result: Simulation 4

5. When $n=1000$, $n_{\text{iterations}}=1000$

```
9 - chromosomes=5;        % number of parents
10 - n=1000;               % array for randomly
11 - b_raw=rand(n, chromosomes); % array for randomly
12 - b=random_lower+((random_upper-random_lower).*b_raw); % number of iterations
13 - n_iterations=1000;    %number of iterations
14 - max_fitness=zeros(n_iterations, 1);
```

**Command Window**

New to MATLAB? See resources for Getting Started.

```
best_fitness =
178.0000

best_parent =
-0.0000  5.0000  -5.0000  -5.0000  -5.0000
```

Figure 13: Stage 1 result: Simulation 5
As mentioned earlier, clearly, increasing $n$ and $n_{\text{iterations}}$ provides a better chance to the algorithm to find a solution (parent) with a better fitness, but at the expense of increased computational time. In this example, the best result was obtained when both $n$ and $n_{\text{iterations}}$ were chosen to be 1000, but anything higher than 100 for $n$ and $n_{\text{iterations}}$ yielded acceptable results. We can also observe by comparing results 2 and 3 that using more parents and less iterations yielded better results than that from using less parents with more iterations.

Since the code is written in a generic way with generalized variables, it is easy and practical to tweak the parameters and adapt the code for other applications. This flexibility made it really easy for us to move on to the next step which was to apply the GA to an image processing application. Since we have already explained the GA concepts in some detail, we will skip the basics and just describe the particulars associated with the image processing application.
Object matching in MATLAB using GA:

We use a similar concept for object matching. The purpose is to determine if we can use GA to match or identify objects. Firstly, we created a simple binary image of 1000 x 1000 pixels. Then we created a triangle by assigning some pixel values equal to 1 in such a way that a triangle is produced. The original image, $I$, is presented in Figure 14.

![Figure 14: Original image](image.png)
Then, we cropped out the triangle from $I$ to create a pattern, and we subjected the pattern
to different transformations to obtain a number of transformed triangles. Each triangle was
essentially associated with a parent. For our examples, we used two different transformations,
namely rotation and scaling. Therefore, the parameters to be optimized were size and angle of
rotation. Two additional parameters were $x$ and $y$, namely the coordinates of the center of the
transformed patterns (triangles).

The transformed patterns were compared to the location in the original image, as specified
by the pattern center $(x, y)$, to find the fitness. The goal was to find the best parent and its
 corresponding chromosome consisting of the location, size, and angle that best matches the triangle
in the original image $I$. We tried to achieve this goal using the GA method that we used for the
basic function as we described earlier in this thesis. As in the previous example, it was important
to identify what our parents, chromosomes, and fitness function look like. It is important to
emphasize again that the chromosomes consist of the following parameters:

1. $y, x$ (center of the generated triangle to be compared with the original image),
2. angle, and
3. scale

For $n$ parents, we have $n$ differently transformed triangles. The code snippet below shows how
image transformation is performed in MATLAB. For transformation purpose, we had to use the
Image Processing toolbox in MATLAB.
The transformation can be expanded by including a variable angle scaling and shearing, but we just resorted to using scaling and angle for simplicity. As can be observed in Figure 15, this code snippet generates \( n \) randomly transformed patterns. Each randomly transformed pattern needs to be compared to the original image based on the \( x \) and \( y \) values of each parent. This part needs to be done carefully, as we have to make sure the size of the transformed pattern matches that of the portion in the original image which is compared. After ensuring that the size of the pattern and the corresponding portion of image match, the fitness is calculated based on the difference of the pixels between the two. In order to be consistent with our definition of fitness (that a large fitness implies a better parent), we use the inverse of the distance, so that a higher fitness value corresponds to better object or pattern matching. The fitness function can be expressed as:

\[
\text{fitness}(i) = \frac{1}{\sqrt{\text{sum}(\text{sum}((I(x_A:x_B,y_A:y_B)-\text{smallerImage})^2))}}
\]

where \( I \) is the original image, \( \text{smallerImage} \) is the transformed pattern, and \( x_A, x_B, y_A, y_B \) are chosen in such a way that the transformed pattern lies in the desired center \((y, x)\) in the original image. The fitness values are normalized similarly to the previous example so that the Roulette
method can be used. After that, crossover is applied, and then mutation. The mutation in this example is performed a little differently. In the previous example, for the crossed-over parents, a randomly generated mutation was employed for all genes in the chromosome. However, in this application, each gene corresponds to a different type of variable (location, angle, size). Therefore, the range of each gene in the chromosome is different, and they have to be differently mutated. As in the previous example, we repeat this process for $n_{iterations}$ and choose the best fitness and the corresponding parent.
Stage 2 Results:

We know that increasing \textit{n\_iterations} or \textit{n} would result in increasing computational time. Since we are using image processing along GA, the processing time increases even further. To show the ongoing process and see how parents are getting modified in each iteration, we chose 200 parents for 9 iterations. The asterisks represent the location of the transformed pattern \((y, x)\) within the chromosome of each parent. Since we have 200 parents chosen, we will have 200 of those points dispersed randomly the first time. After each iteration, we expect to see them converging around the triangle in the original image to finally yield the best fit solution and the corresponding parent for the purpose of object matching.
The purpose is to have the blue asterisks getting closer around the center of the white triangle with every iteration. In Figure 16, we can clearly see the blue asterisks converging to the region around the white triangle. Obviously, increasing the number of iterations gather the asterisks around the triangle more accurately. Though we had only 200 parents and just 9 iterations,
the resulting solution was still impressive. To check what the best solution is, we store the best fitness and parent from each iteration and, at the end, we choose the best fitness of all the iterations and its respective parent. Our desired best parent is supposed to be [300, 700, 0, 1], where 300 and 700 are the center coordinates, 0 is the angle, and 1 is the scaling. These numbers refer to the original triangle in the original image, \( I \).

We tested our simulation by setting \( n \) as 200 and \( n\_iterations \) as 10, and we obtained the following results:

\[
\text{best_fitness} = 0.9044 \\
\text{best_parent} = \begin{bmatrix} 297 & 0.0000 \\ 689 & 0.0000 \\ 0.1000 \\ 0.8016 \end{bmatrix}
\]

Figure 17: Stage 2 result: Simulation 1

The result was close to the desired solution. However, when we changed \( n \) to 1000, while keeping \( n\_iterations \) at 10, we obtained better results. The result is shown below in Figure 18.
It can be observed that we had better results than before. We also wanted to see what happens when we set \( n \) as 1000 and \( n_{\text{iterations}} \) as 100.

This result was significantly better, and very close to the best solution. Not only the location, but also the angle and magnification size was almost the same as our original pattern found in \( I \). However, the computational time was significantly higher. The trade-off between accuracy and computational time is thus apparent.
We also tried to go one step further and try for $n$ set to 1000 and $n_{\text{iterations}}$ set to 1000 as well. Though the computational time skyrocketed, the resulting solution matched our desired solution with having only a negligible error. The obtained triangle almost perfectly overlapped the original one.

\begin{verbatim}
best_fitness =
   0.9972

best_parent =
   300.0000  700.0000  0.1000  0.9998
\end{verbatim}

Figure 20: Stage 2 result: Simulation 4
Stage 3 Results:

Next, we added some noise to the original image to see if we get similar results. To add the noise, we used the noise function with “salt and pepper” in MATLAB. The figure below shows the original image with added noise.

Figure 21: Original image with noise

Our original function did not work to get us our desired results, and thus we had to use a different fitness function defined as:
fitness\(i\) = \(0.01/(0.01+\sum(\sum((I(x_A:x_B,y_A:y_B)-\text{smallerImage})^2))/(s_{\text{smallerImage}}x \times s_{\text{smallerImage}}y))^2\)

where we biased the fitness values based on the size and distance. Increasing the power helped us eliminate candidate solutions which were far from the triangle. We also changed our mutation range to give us a larger set of possibilities to work with. However, to avoid having the candidate solutions moving around a lot throughout the iterations, we added the following piece of code to start decaying the range of the mutation values after a certain number of iterations. This decay allows the solutions to converge towards the correct solution as the iterations progress.

```
if j>n_iterations/10
    random_lower_m=random_lower_m*0.98;
    random_upper_m=random_upper_m*0.98;
    random_lower_l=random_lower_l*0.98;
    random_upper_l=random_upper_l*0.98;
end
```

Figure 22: MATLAB snippet for decaying mutation

After these adjustments to the code, we were able to get better results. Figure 23 below shows the parents’ adaptation in every iteration. The results are presented in a similar way as in Stage 2 to see if parents converge around our desired solution or not.
After we confirmed the convergence towards our expected solution, we ran a simulation with \( n \) as 1000 and \( n\text{\_iterations} \) as 100, and the result is shown below.

Figure 23: Stage 3: Process of every iteration
The figure below shows the chosen solution on top of the original image. We can confirm that it’s a good match, even though a substantial amount of noise has been added.

```
best_fitness = 0.0685

best_parent =
300.0000 700.0000 0.1000 0.9970
```

Figure 24: Stage 3 result: Simulation

Figure 25: Stage 3: Overlap of the best solution with the original image
Stage 4 Results:

The next step was to add other objects to the original image on top of the noise to see if the algorithm is able to identify the desired triangle as the best solution. The new original image after adding noise and other objects is shown in figure below.

![Image of Stage 4: Original image with noise and other objects](image)

Figure 26: Stage 4: Original image with noise and other objects

We had to use a big number of particles and a high number of iterations to get a successful result. Also, we were not able to get the desired solution accurately each time. We then realized that our code needs further improvement to be able to work well in this scenario. To provide a better idea
of how the algorithm performed, we provide in Figure 27 below the location of the best parents obtained in all iterations using red asterisks.

![Figure 27: Stage 4: Best parent of every iteration shown in red asterisks](image)

As we can see, there are more asterisks on our desired triangle. We can also see a few asterisks in the other shapes as well. This result does not necessarily imply that the algorithm has failed. It still shows that we are able to detect all different objects in the grayscale image. As part of future work, the GA method could be employed around all areas identified by these best parents. It can be
conjectured that working locally around the different shapes (one location at a time), would identify the triangle as the best matched object.

Figure 28: Stage 4: Overlap of the best solution with the original image
Conclusion/Future work:

Optimization is one of the most useful tools that can be used in image processing and in particular in areas such as object matching. While approaching this application in a mathematical way, by using Convex Optimization, it may be possible to obtain similar results through the use of a heuristic algorithm like GA. This is especially true when the problem is not truly convex. Many works have been presented in the literature which have demonstrated that GAs have been used effectively to solve many different problems. The objective of this thesis is to investigate how well GAs can be used for solving the object matching problem.

Our future work should move in the direction of testing additional and more realistic images for object matching, such as by using color (RGB: Red, Green Blue) instead of grayscale images. Moreover, the shear effect can be added to the geometrically transformed objects. Moreover, we need to try different variations of selection, crossover, and mutation methods to determine which combination of methods may yield the best solution. For example, in Stage 4, we could potentially improve our code by biasing our selection function to keep the best 10% of the parents in a particular iteration so that they are also members of the next generation for the next iteration. This can ensure we never lose the best set of parents. Similar tweaking in mutation and crossover algorithms can be performed to achieve better results for real-life images, which we would like to work on in the future.

It is however clear that increasing the number of parents or increasing the number of iterations increases the chances of finding the best fit solution. However, at the same time, the computation time requirements impose an unfortunate trade-off.
Finally, it can be said that for cases where one is interested in getting a desired solution for an optimization problem in non-real time and without getting acquainted with too much of mathematics, GA seems to be a really useful tool.
Bibliography:


Esser, Ernie. Convex Optimization in Image Processing. 5 May 2010,

“Heuristic Algorithms.” Facility Location Problems - Optimization, 8 June 2014,
optimization.mccormick.northwestern.edu/index.php/Heuristic_algorithms.

Introduction, Basics, Types & Applications, www.engineersgarage.com/articles/image-
processing-tutorial-applications.

Engineering And Computer Science, vol. 6, no. 12, 12 Dec. 2017, pp. 1–3.,