Topography Identification Using Multilevel Thresholding and Genetic Algorithm

for the Bachelor of Science (special) Degree

By
O.G.V.V Bandara
SC/2019/10723

Supervisor:
Dr D.M Samarathunga

Department of Mathematics
University of Ruhuna
Matara.

2024

Abstract

The project proposes an innovative approach to image thresholding by employing multilevel thresholding combined with genetic algorithms. The method aims to enhance segmentation accuracy while reducing computational complexicity. The objectives of the project were to perform image thresholding by combining available thresholding methods with genetic algorithms to achieve multilevel thresholding capabilities. The project aimed to develop a thresholding algorithm for topography identification. Aerial images have vast amounts of details and structures and can be overwhelming during analysis and observation. Thresholding reduces the range of color intensities which eases the structural identification process of topography aerial images. A sample aerial image was utilized for algorithm testing and evaluation. The method required the image to be converted to grayscale since most available algorithms require grayscale images. Compared to traditional otsu methods the newly developed algorithm uses the between class variance as the fitness function for the genetic algorithm. The individual selection was done through the roulette wheel selection method. The thresholding was performed for the grayscale histogram and thresholds were applied to the images using the algorithm. Four metrics were used for the evaluation of the algorithm performance namely Peak signal-to-noise ratio, Structural similarity index, execution time, Dice coefficient. The optimal parameters where determined using each metric. Evaluation through the peak signal-to-noise ratio determined that the optimal parameters are population size of 100, generations of 200 and thresholds of 5 which yielded a PSNR of 29.3. The structural similarity index determined the two sets of optimal parameters. One as population size of 100, generations of 100 and thresholds of 5 which yielded SSIM as 0.84. The other as population size of 150, generations of 200, thresholds of 5 and SSIM of 0.84. Based on the execution time the optimal parameters where determined as population size of 50, generations of 100, thresholds of 3 and execution time as 58.7 sec. Parameters can be adjusted to suite the requirement. Either for low computational complexity or more thresholding accuracy or image quality. The final thresholded image is output from the algorithm which can be used for many computer vision tasks.

Acknowledgment

I would like to express my deepest gratitude to my supervisor, Dr D.M Samarathunga, whose expertise, guidance, and unwavering support have been invaluable throughout the course of this project. Your insightful feedback and encouragement have been crucial in shaping this work and driving it to completion.

I am also immensely grateful to my teammate, B.K.V Jayarathna for his dedication, collaboration, and continuous support. Your contributions have been essential to the success of this project. The brainstorming sessions, discussions, and collaborative efforts have greatly enhanced the quality and scope of this project.

Contents

Chapter 1

Introduction

1.1 Background of the study

1.1.1 Image segmentation

Image segmentation is the process of partitioning a digital image into segments/objects to simplify features for analysis. Image segmentation is an important aspect in computer vision studies. The image is required to be partitioned so that the characteristics can be distinguished from groups of pixels. Image segmentation has a variety of applications such as facial recognition, machine vision, object tracking and precision agriculture etc. There are many image segmentation techniques including edge detection based , histogram based ,clustering based and threshold based.
The method used in this project is the thresholding method. Thresholding is a widely used image segmentation approach. An image in grayscale can be segmented into clusters in way to be separated into levels of pixel intensity where one value is attributed to the pixels on the same region. Consider the instance where an image consists of L levels of precision where the image has a RGB color space with each pixel value range from 0-255. The binary thresholding of the image or converting the image to grayscale is as follows

$$g(x,y) = \begin{cases} 1 & paraf(x,y) >= T \\ 0 & paraf(x,y) < T \end{cases}$$

where
$f(x,y)$ - grayscale image
$g(x,y)$ - Threshold image
T - Threshold value
The other instance is multilevel thresholding. In multilevel thresholding a large number of objects are classified.

1.1.2 Thresholding techniques

Mean technique

One technique is the mean technique. This technique uses the mean values of the pixels as the threshold. This thresholding technique works in an instance where

half the pixels belonging to the object and other half belonging to the background. This is a rare instance

p-tile technique

The other method is the p-tile technique. This technique uses the knowledge about the area size of the desired object to threshold the image. p-tile method is one of the earliest methods based on gray level histogram. it assumes that objects in an image are brighter than the background and occupy a fixed percentage of the area. The fixed percentage of the area is known as the 'p percent'.The threshold can be defined as the gray level that mostly corresponds to mapping at least 'p percent' of gray level into object.

Histogram dependent technique(HDT)

The histogram based technique is dependent on the success of the estimating the threshold value that separates the two homogeneous regions of the object and the background of an image. But there is requirement that the image formation should be of two homogeneous regions and there exists a threshold value that separates these regions. The technique can be expressed as

$$C(T) = P_1(T)\sigma_1^2(T) + P_2(T)\sigma_2^2(T) \tag{1.1}$$

$C(T)$ -within-group variance
$P_1(T)$ -probability of group with values less than T
$P_2(T)$ -probability of group with values greater than T
$\sigma_1(T)$ -variance of group of pixels with values less than or equal T
$\sigma_2(T)$ -variance of group of pixels with values greater than T

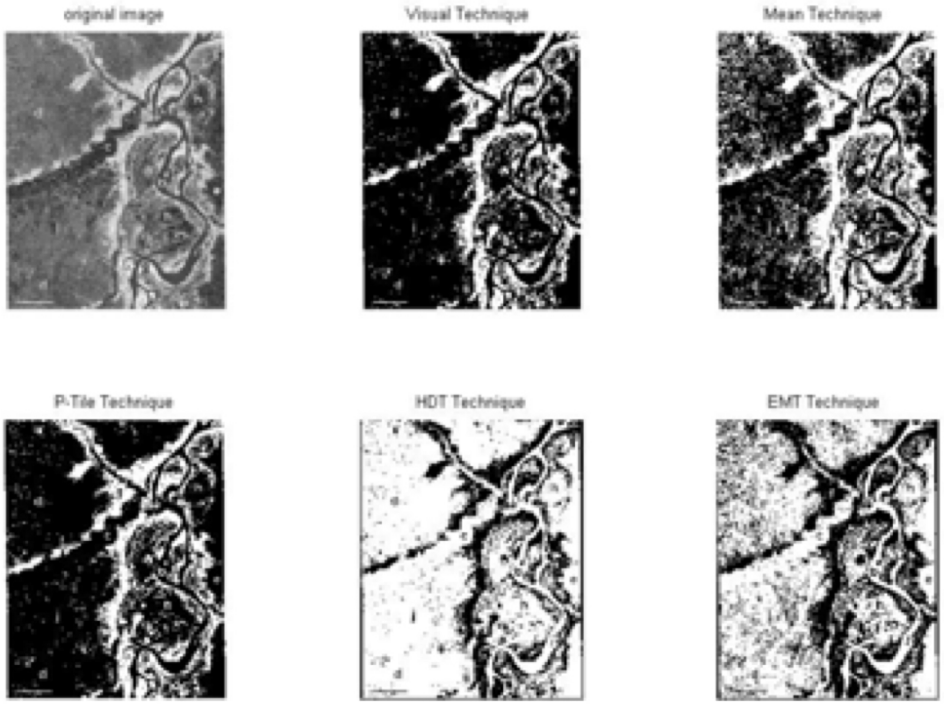
EMT technique

This technique is known as edge maximization technique(EMT). Used in an instance where there is more than one homogeneous region in image or where there is a change on illumination between the object and it's background. In this case a portion of the object is merged with the background or portion of the background may as an object. This automatic threshold selection technique performance becomes much better in large homogeneous well separated regions.

Visual technique

This technique involves using algorithms to improve the recognition ability by a person. The image is thresholded in such a way that a person can identify the segmented parts more clearly. This technique is a bit similar to p-tile technique.

The below figure represents the original image subjected to above mention various thresholding techniques.

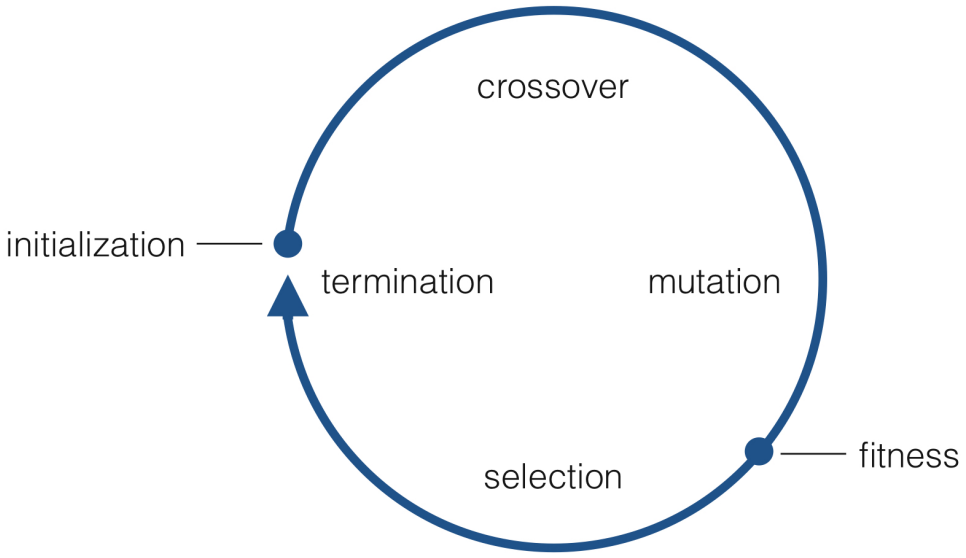


1.1.3 Genetic algorithms

Genetic algorithms are used to solve short optimization problems. optimization is the task of finding optimal solutions. It is often the way to target the optimal solution in the global optimal solution which is the best solution in the whole solution space. This is usually a tedious task since the solution space suffers from constraints, noise , strange fitness function conditions , unsteadiness and large number of local optima. For many hard problems no efficient solution is available and heuristics like genetic algorithms are reasonable to apply.
genetic algorithms are the translation of the biological concept of evolution into algorithmic recipes. Which belong to the area of computer science related to machines and computer programs. Four main streams of genetic algorithm variants have been developed. namely genetic algorithms, evolution strategies, evolutionary programming and genetic programming.

From biology to genetic algorithms

genetic algorithms are biologically inspired algorithms for optimization. Darwinian evolution proposes the explanation for the biological development of species with mating selection and survival of the fittest. Biological creatures comprising DNA are a good example of successful optimization process that has been running for billions of years. Some creatures have a slow and stable evolution and bacteria have a faster evolution because of the structural complexity and fast reproduction rate. The genome is the sequence that induces biological processes in cells and organisms. The following figure visualizes the continuous cycle of artificial evolution that is based on the principal of natural evolution.



Elements of genetic algorithms

There is no rigorous definition of genetic algorithm accepted by all the evolutionary computation community. Most methods known as genetic algorithms have a few methods in common. Population of chromosomes, selection according to fitness, crossover to produce new offspring and random mutation of new offspring. The chromosomes in a GA population takes the form of bit strings. Each locus in the chromosome has two possible alleles 0 and 1. Each chromosome can be thought of as a point in the search space of candidate solutions. GA processes populations of chromosomes successively replacing one such population with another. The GA requires a fitness function that assigns a score/fitness to each chromosome in the current population. The fitness of the chromosome depends on the accuracy of performance of the chromosome. An example of a fitness function can be given by

$$f(y) = \Big\{ y + sin(32y) \quad 0 <= y <= x$$

here the candidate solutions are the values of y. y can be encoded as bit strings representing real numbers. The fitness calculation translates a given bit string x into a real number y and then evaluates the function at that value.

Genetic algorithm operators

The simplest genetic algorithm consists of 3 operators namely Selection, Crossover, Mutation.

- Selection: This operator selects chromosomes in the population for reproduction. The fitter the chromosome the more times it is likely to be selected
- Crossover: This operator randomly chooses a locus and exchanges the sub sequences before and after that locus between two chromosomes to create two offspring.
- Mutation: This operator randomly flips some bits in a chromosome. for example the string 00000100 might be mutated in it's second position to yield 01000100

Applications of genetic algorithms

An interesting approach of research is the optimization of machine learning models using genetic algorithms. They can be used for tuning of parameters of machine learning models. They have been successfully applied in balancing support vector machines and for tuning dimensionality reduction techniques. Among the huge variety of dimensionality reduction techniques, unsupervised learning is a promising one. Such an optimization problem is the process of minimizing the error between this mapping and the original patterns which is also known as data space reconstruction error, as the pattern sin the data space have to be reconstructed by mapping from low dimensional space to high dimensional one which is a quite difficult optimization problem to solve.
Another application is when balancing ensembles.ensemble machine learning methods are known to outperform their counterparts. The predictions from multiple classifiers are combined into one single prediction. The number of ensemble members of each type and their parameters are the variables of the optimization problem. The parameters are bound constrained, as an upper bound on the number of classifiers is reasonable from a computational perspective. For discrete optimization variables NSGA-2 with random setting or Gaussian mutation with rounding can be applied.
Genetic algorithms can also be applied for feature tuning. For the integration of wind into the power grid the precise prediction of wind energy has an important part to play. The question comes up, which surrounding turbines are the best ones for the wind prediction problem. To optimize the influence of the particular turbines in the environment, genetic algorithms can be employed.

1.2 Objectives

- Segmentation of an image into observable regions
- Segmentation of images consisting of non homogeneous separation between objects
- Development of multilevel thresholding based approach using genetic algorithms for image thresholding
- A faster approach for multilevel thresholding
- Topography identification using computer algorithms
- Improving segmentation accuracy using algorithm based thresholding of topographical images

Chapter 2

Problem Statement

The problem we intend to assess is the problem of thresholding a Image of terrain. Terrain tend to have various types of structures and formations of objects and patterns. Using an aerial image, It is quite hard to determine the structures present within a terrain aerial Image. For identification of terrain with better accuracy a thresholded image can be utilized. Thresholding involves computer algorithms and computation power. A model is required that can threshold a image and determine optimum thresholds for that image. Determination of optimum threshold of an image requires a method of obtaining the pixel intensities of the image and finding points in which the image can be separated into classes. Classification into two classes have already been performed using methods such as Otsu's method. But multilevel thresholding require more sophisticated methods. Certain characteristics of an image have to be identified and one of the best choices of characteristics must be utilized for determination of optimum thresholds using algorithms. The derivation of new thresholding algorithms is not a necessity. But it was required that there is a method to utilize already available thresholding methods and combine them with optimization algorithms. Through this it would be possible to obtain multilevel thresholding.

Chapter 3

Methodology

3.1 Multilevel thresholding using Otsu's method

Otsu's method has been formulated for two classes. By the two class formulation it can be expanded to many classes with just a few modification. Let the pixels of a given picture be represented in L gray levels $[1, 2, , L]$. The number of pixels at level i is denoted by n_i and the total number of pixels by $N = n_1 + n_2 + +n_L$ In order to simplify the discussion, the gray-level histogram is normalized and regarded as a probability distribution.

$$p_i = \frac{n_i}{N} \tag{3.1}$$

Now suppose that we dichotomize the pixels into two classes C_O and C_1 (background and objects, or vice versa) by a threshold at level k; C_O denotes pixels with levels $[1, , k]$, and C_1 denotes pixels with levels $[k+1, , L]$. The set of pixel intensities would be separated into two classes at the threshold k. Then the probabilities of class occurrence and the class mean levels, respectively, are given by

$$\omega_0 = Pr(C_0) = \sum_{i=1}^{k} P_i = \omega(k) \tag{3.2}$$

$$\omega_1 = Pr(C_1) = \sum_{i=k+1}^{L} P_i = 1 - \omega(k) \tag{3.3}$$

and,

$$\mu_0 = \sum_{i=1}^{k} iPr(i|C_0) = \sum_{i=1}^{k} \frac{iP_i}{\omega_0} = \frac{\mu(k)}{\omega(k)} \tag{3.4}$$

$$\mu_1 = \sum_{i=k+1}^{L} iPr(i|C_1) = \sum_{i=k+1}^{L} \frac{iP_i}{\omega_1} = \frac{\mu_T - \mu(k)}{1 - \omega(k)} \tag{3.5}$$

where,

$$\omega(k) = \sum_{i=1}^{k} P_i \tag{3.6}$$

$$\mu(k) = \sum_{i=1}^{k} iP_i \tag{3.7}$$

are the zeroth-order and the first-order cumulative moments of the histogram up to the k^{th} level, respectively, and

$$\mu_T = \mu(L) = \sum_{i=1}^{L} iP_i \tag{3.8}$$

is the total mean level of the original picture. We can easily verify the following relation for any choice of k.

$$\omega_0\mu_0 + \omega_1\mu_1 = \mu_T \tag{3.9}$$

$$\omega_0 + \omega_1 = 1 \tag{3.10}$$

The class variances are given by,

$$\sigma_0^2 = \sum_{i=1}^{k}(i-\mu_0)^2 Pr(i|C_0) = \sum_{i=1}^{k}\frac{(i-\mu_0)^2 P_i}{\omega_0} \tag{3.11}$$

$$\sigma_1^2 = \sum_{i=k+1}^{L}(i-\mu_1)^2 Pr(i|C_1) = \sum_{i=k+1}^{L}\frac{(i-\mu_1)^2 P_i}{\omega_1} \tag{3.12}$$

In order to evaluate the "goodness" of the threshold (at level k),the following discriminant criterion measures will be considered, within-class variance:

$$\sigma_w^2 = \omega_0^2\sigma_0^2 + \omega_1^2\sigma_1^2 \tag{3.13}$$

between-class variance:

$$\sigma_B^2 = \omega_0(\mu_0 - \mu_T)^2 + \omega_1(\mu_1 - \mu_T)^2 \tag{3.14}$$

total variance:

$$\sigma_T^2 = \sum_{i=1}^{L}(i-\mu_T)^2 P_i \tag{3.15}$$

The optimization problem is now reduced to a search for thresholds k by optimizing either of the above function. For our project we chose the between class variance. The optimal threshold k^* that maximizes η, or equivalently maximizes σ_B^2 is selected in the following sequential search by using the simple cumulative quantities.

$$\eta(k) = \frac{\sigma_B^2(k)}{\sigma_T^2} \tag{3.16}$$

and the optimal threshold k^* is

$$\sigma_B^2(k^*) = \max_{1<k<L}(\sigma_B^2(k)) \tag{3.17}$$

3.2 Genetic algorithm for optimal threshold Identification

Genetic algorithms are based on the idea of natural selection and genetics. Genetic algorithms tend to replicate the process of speciation. Genetic algorithms tend to use the concept of natural selection where the fittest will survive and go to the next generation to reproduce.

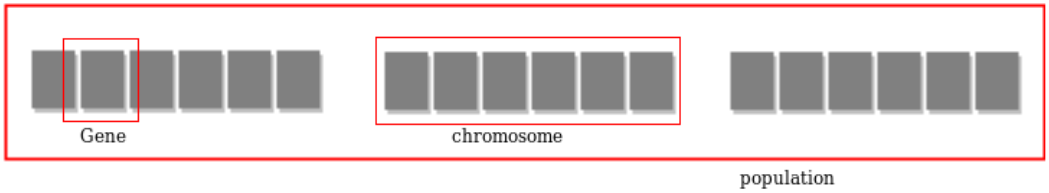


Figure 3.1: Population consisting chromosomes

As shown by the above figure a chromosome consists of genes in which a population is a collection of genes.The below figure represents the flowchart of a continuous genetic algorithm.

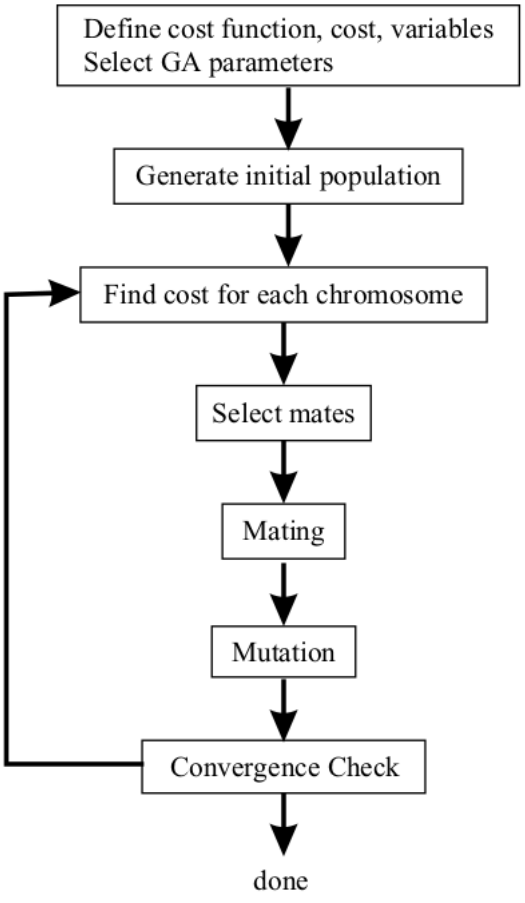


Figure 3.2: Flowchart of genetic algorithm

3.2.1 Variables and Cost function

The goal is to solve some optimization problem where we search for an optimal solution in terms of variables of the problem. A chromosome is defined as an array of variables . If chromosome has N_{var} variables given by $p_1, p_2, ..., p_{Nvar}$ then the chromosome is written as an array with $1 \times N_{var}$ elements so that

$$chromosome = [P_1, P_2, ..., P_{N_{var}}] \tag{3.18}$$

The variables are represented as floating point numbers. Each chromosome has a cost found by evaluating cost function f at the P variables $P_1, P_2, ...P_{Nvar}$.

$$cost = f(chromosome) = f(P_1, P_2, ..., P_{N_{var}}) \tag{3.19}$$

3.2.2 Initial population

The initial population defined N_{pop} chromosomes. A matrix represents the population with each row in matrix being $1 \times N_{var}$ array of continuous values. Given an initial population of N_{pop} chromosomes, the full matrix of $N_{pop} \times N_{var}$ random values is generated by

$$pop = rand(N_{pop}, Nvar) \tag{3.20}$$

The population contains chromosomes which are vectors containing threshold values of images which are randomly generated within the defined range. The chromosomes are passed to the cost function for evaluation. A random array of multiple chromosomes with random values between the defined range of numbers will make up the initial population.

3.2.3 Natural selection

In this stage it is decided which chromosomes are fit to survive to the next generation and reproduce an offspring/child. The N_{pop} costs and associated chromosomes are ranked from lowest cost to highest cost. The rest die off. This process of natural selection occurs at each iteration of the algorithm allowing population of chromosomes to evolve over generations to most fit members as defined by the cost function. Not all survivors are deemed fit enough to mate. only top $Nkeep$ are kept and rest discarded.

3.2.4 Pairing

most-fit chromosomes form the mating pool. Two mothers and fathers pair in some random fashion. Each pair produces two offspring that contain traits from each parent. In addition the parents survive to be part of the next generation. The more similar the two parents, the more likely are the offspring to carry the traits of the parents.

3.2.5 Mating

The simplest methods choose one or more points in the chromosome to mark as the crossover points. Then the variables between these points are merely swapped

between the two parents. consider the two parents to be

$$parent_1 = [p_m1, p_m2, ...p_{mNvar}] \tag{3.21}$$

$$parent_2 = [p_{d1}, p_{d2}, ...p_{dNvar}] \tag{3.22}$$

Crossover points are randomly selected, and then the variables in between are exchanged. Heuristic crossover is a variation where some random number, β, is chosen on the interval $[0, 1]$ and the variables of the offspring are defined by

$$P_{new} = \beta(p_{mn} - pdn) + p_{mn} \tag{3.23}$$

Variations on this theme include choosing any number of variables to modify and generating different β for each variable. It begins by randomly selecting a variable in the first pair of parents to be the crossover point

$$\alpha = roundup(random * N_{var}) \tag{3.24}$$

where the m and d subscripts discriminate between the mom and the dad parent. Then the selected variables are combined to form new variables that will appear in the children

$$p_{new1} = p_{m\alpha} - \beta[p_{m\alpha} - p_{d\alpha}] \tag{3.25}$$

$$p_{new2} = p_{d\alpha} + \beta[p_{m\alpha} - p_{d\alpha}] \tag{3.26}$$

where β is also a random value between 0 and 1. The final step is to complete the crossover with the rest of the chromosome as before

$$offspring_1 = [p_{m1}, p_{m2}, ..., p_{mVar}] \tag{3.27}$$

$$offspring_2 = [p_{d1}, p_{d2}, ..., p_{mNvar}] \tag{3.28}$$

With this procedure new chromosomes or child's are created with swapped threshold values of previous chromosomes leading to a new population of chromosomes with different thresholds.

3.2.6 Mutations

the GA can converge too quickly into one region of the cost surface. If this area is in the region of the global minimum, that is good. However, some functions, such as the one we are modeling, have many local minima. If we do nothing to solve this tendency to converge quickly, we could end up in a local rather than a global minimum. To avoid this problem of overly fast convergence, we force the routine to explore other areas of the cost surface by randomly introducing changes, or mutations, in some of the variables. To the vector of thresholds or in other words the chromosome is mutated by introducing random values within the defined range in to random indices of the chromosome at this stage.

3.2.7 Generations

The process is iterated multiple times until the optimal solution is found for the optimization function. In the case of image thresholding the optimal thresholds can be found by running the algorithm through multiple generations.

3.3 Evaluation

3.3.1 Peak Signal-to-Noise Ratio (PSNR)

It quantifies the level of distortion between a reconstructed image and its original version, making it a key indicator of fidelity and performance for image segmentation algorithms. PSNR is valuable in situations where the aim is to maintain the quality of the original image, especially when images are subjected to processes like compression, filtering, or noise reduction. It helps to quantify how much the reconstructed image deviates from the original image, thus providing an objective measure of quality.
Before calculating PSNR, the Mean Squared Error (MSE) between the original and processed images is determined. MSE is the average of the squares of the differences between corresponding pixel values in the original and reconstructed images.

$$\text{MSE} = \frac{1}{MN} \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} \left(I(i,j) - K(i,j) \right)^2 \tag{3.29}$$

- $I(i,j)$ is the pixel value at position (i,j) in the original image
- $K(i,j)$ is the pixel value at position (i,j)(i,j) in the processed image
- M and N are the image dimensions (height and width).

Once the MSE is calculated, PSNR can be determined using the following formula.

$$\text{PSNR} = 10 \cdot \log_{10} \left(\frac{L^2}{\text{MSE}} \right) \tag{3.30}$$

- L is the maximum possible pixel value of the image
- PSNR is measured in decibels (dB), a logarithmic scale that allows for a more convenient representation of image quality differences.

3.3.2 Structural Similarity Index (SSIM)

Structural Similarity Index (SSIM) is a widely used metric for assessing the similarity between two images, aiming to mimic the human visual perception of image quality. Unlike simpler measures such as Mean Squared Error (MSE) or Peak Signal-to-Noise Ratio (PSNR), SSIM considers changes in structural information, luminance, and contrast. The SSIM index evaluates the visual impact of three characteristics namely luminance, contrast, and structure.

$$\text{SSIM}(I,K) = \frac{(2\mu_I\mu_K + C_1)(2\sigma_{IK} + C_2)}{(\mu_I^2 + \mu_K^2 + C_1)(\sigma_I^2 + \sigma_K^2 + C_2)} \tag{3.31}$$

- I and K: These are the two images being compared. Typically, I is the reference (original) image, and K is the distorted (or compared) image.
- μ_I and μ_K: These represent the mean luminance (average pixel intensity) of images I and K respectively. The mean luminance is a measure of the average brightness of an image.

- σ_I^2, σ_K^2 : These denote the variance (or contrast) of images I and K respectively. The variance reflects the spread or dispersion of pixel intensities, indicating the contrast in the images.

- σ_{IK} :This is the covariance between images I and K. The covariance measures the degree to which the pixel intensities in the two images change together, capturing structural similarity.

- C_1, C_2, C_3 :These are small constants added to stabilize the division operations, especially when the denominator is close to zero. These constants are defined based on the dynamic range of the pixel values

luminance comparison

The luminance component compares the mean intensities of the two images and is given by

$$l(I, K) = \frac{2\mu_I\mu_K + C_1}{\mu_I^2 + \mu_K^2 + C_1} \tag{3.32}$$

This term assesses how similar the brightness (average intensity) of the two images is. If μ_I and μ_K are close, the luminance term will be near 1, indicating similar brightness. C_1 ensures stability in cases where the denominator could become zero. The choice of C_1 is crucial when dealing with images with low contrast.

contrast comparison

The contrast component compares the standard deviations (contrast) of the images. This term measures the contrast similarity between the two images.

$$c(I, K) = \frac{2\sigma_I\sigma_K + C_2}{\sigma_I^2 + \sigma_K^2 + C_2} \tag{3.33}$$

If the contrast (variance) is similar, this term will be close to 1. Similar to C_1 , C_2 is a constant to avoid division by zero. It is often set based on the desired sensitivity to contrast changes.

structure comparison

The structure component compares the structural patterns of the two images. This term evaluates the correlation (or similarity) of the patterns in the images. If the structural details are similar, σ_{IK} will be large, leading to a structure component value near 1. C_3 is derived from C_2 and helps ensure numerical stability, especially in cases where the structural details might be small.

$$s(I, K) = \frac{\sigma_{IK} + C_3}{\sigma_I\sigma_K + C_3} \tag{3.34}$$

3.3.3 Dice Coefficient (Dice Similarity Index)

The Dice Coefficient is a statistical measure used to evaluate the similarity between two sets or binary data. The Dice Coefficient equation measures the overlap between two sets.

$$\text{Dice}(A, B) = \frac{2|A \cap B|}{|A| + |B|} \tag{3.35}$$

- A and B : These are the two sets (or binary images/masks) being compared. In the context of image segmentation, A might represent the set of pixels in a predicted segmentation mask, while B represents the set of pixels in the ground truth mask.
- $|A|$ and $|B|$: These represent the size (or cardinality) of the sets A and B, respectively. This is the total number of elements (or pixels) in each set.
- $|A \cap B|$: This denotes the size of the intersection of sets A and B. It is the number of elements that are common to both sets, representing the overlap between the two.

3.3.4 Execution Time

The Execution Time measures the time taken to complete a task, calculated as the difference between the start and end timestamps.

$$\text{Execution Time} = T_{\text{end}} - T_{\text{start}} \tag{3.36}$$

3.4 Topography identification

Topography identification using thresholding is a technique often employed in digital image processing to analyze and extract meaningful information from terrain data, such as satellite imagery or digital elevation models (DEMs). This approach can help in identifying and classifying various landforms, features, and surfaces in geographical areas by setting thresholds that distinguish between different types of terrains based on pixel intensity values.Application of topography identification namely

- Landform classification: By setting different elevation thresholds, it is possible to classify land into categories such as plains, hills, mountains, valleys, and water bodies. This aids in creating detailed geographical maps for urban planning and development.
- Vegetation mapping: Thresholding can be applied to multispectral images to identify and classify vegetation cover, such as distinguishing between forested areas and grasslands.
- Terrain analysis for construction: Identifying flat terrains and slopes using thresholding helps urban planners determine suitable locations for construction projects, road networks, and infrastructure development.

- Flood risk assessment: By analyzing elevation and slope data, areas prone to flooding can be identified, enabling the planning of flood defenses and risk mitigation strategies.
- Crop suitability analysis: Thresholding can determine land suitability for different crops based on elevation, slope, and soil characteristics. Identifying fertile plains versus rocky terrains helps in efficient land use planning.
- Habitat mapping: Identifying habitats and ecosystems based on elevation and terrain features helps in conservation planning. Thresholding can distinguish between marshlands, forests, and mountainous regions.
- Earthquake impact assessment: Terrain analysis through thresholding aids in understanding how earthquake forces might propagate through different landforms, informing structural design considerations.
- River basin mapping: Thresholding helps in identifying river basins, watershed boundaries, and drainage networks, essential for water resource management and hydrological studies.

Chapter 4

Discussion

4.1 Initial Data

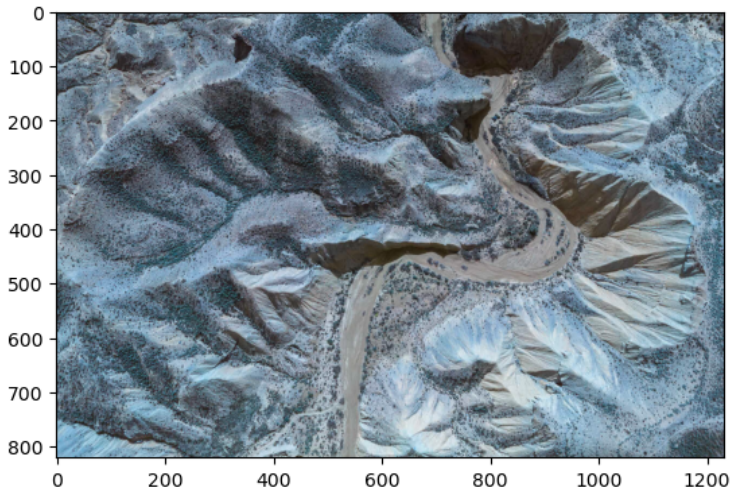


Figure 4.1: Test Image in default color scale

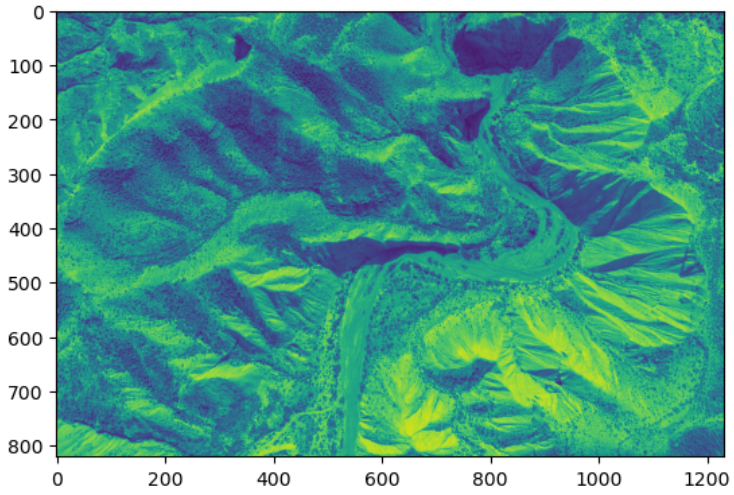


Figure 4.2: Test Image in grayscale

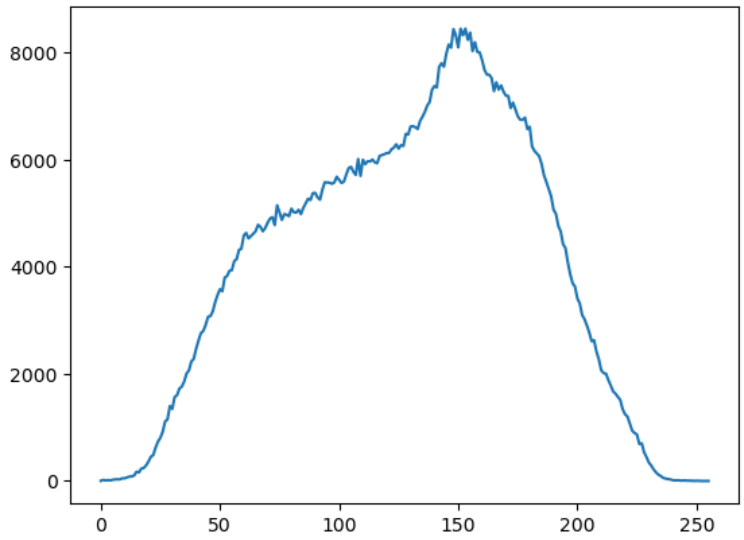


Figure 4.3: Histogram depicting color intensities of test image

4.2 PSNR (Peak Signal-to-Noise Ratio)

parameter set	population size	generations	thresholds	Peak Signal-to-Noise Ratio
set1	50	100	3	28.5
set2	50	150	4	28.9
set3	100	100	5	29.2
set4	100	150	3	28.4
set5	150	100	4	28.7
set6	100	200	5	29.3
set7	150	100	3	28.6
set8	150	150	4	29.1
set9	150	200	5	29.2
set10	200	100	3	28.6
set11	200	150	4	29.1
set12	200	200	5	29.2

Figure 4.4: Peak Signal-to-Noise Ratio for multiple parameter combinations

Peak signal to noise ratio is a metric that is widely used to assess the quality of a processed image compared to the original image. It quantifies the level of signal fidelity by comparing the maximum possible power of a signal to the power of distorting noise. PSNR is particularly useful for assessing image compression, denoising, and image segmentation performance. A higher PSNR indicates image quality. The segmented image is similar in quality to the original. Normally images with PSNR above 30db is considered yo have good image quality but can also be used as a comparative metric. The above table depicts the PSNR values evaluated for multiple combinations of of parameters and the set 6 was selected as the optimal set. Evaluation through the peak signal-to-noise ratio determined that the optimal parameters are population size of 100, generations of 200 and thresholds of 5 which yielded a PSNR of 29.3.

4.3 SSIM (Structural Similarity Index)

parameter set	population size	generations	thresholds	Structural Similarity Index
set1	50	100	3	0.74
set2	50	150	4	0.76
set3	100	100	5	0.84
set4	100	150	3	0.75
set5	150	100	4	0.81
set6	100	200	5	0.83
set7	150	100	3	0.72
set8	150	150	4	0.79
set9	150	200	5	0.84
set10	200	100	3	0.74
set11	200	150	4	0.79
set12	200	200	5	0.81

Figure 4.5: Structural Similarity Index for multiple parameter values

The Structural Similarity Index (SSIM) is a perceptual metric that quantifies image quality by considering changes in structural information, luminance, and contrast. Unlike PSNR, which focuses on pixel-wise differences, SSIM assesses the visual impact of these changes, making it more aligned with human visual perception.SSIM indices range from -1 to 1. where 1 indicates a perfect similarity and 0 means no similarity. Negative values imply inverted similarity. The objective is to attain high SSIM values which would then indicate higher structural similarity to the original image. Two sets of parameters were selected as optimal using the SSIM metric. The structural similarity index determined the two sets of optimal parameters. One as population size of 100, generations of 100 and thresholds of 5 which yielded SSIM as 0.84. The other as population size of 150, generations of 200, thresholds of 5 and SSIM of 0.84.

4.4 Execution Time

parameter set	population size	generations	thresholds	Execution time
set1	50	100	3	58.7
set2	50	150	4	107.5
set3	100	100	5	171.8
set4	100	150	3	180.3
set5	150	100	4	224.4
set6	100	200	5	359.1
set7	150	100	3	196.5
set8	150	150	4	321.1
set9	150	200	5	488.4
set10	200	100	3	226.7
set11	200	150	4	452.4
set12	200	200	5	657.9

Figure 4.6: Execution Time for multiple parameter values

Execution Time measures the computational efficiency of an algorithm. In the context of genetic algorithms for image segmentation, it reflects how quickly the algorithm completes the segmentation process, considering factors like population size, number of generations, and computational resources. lower execution time indicates more efficient segmentation. More complex algorithms may require more

execution time and provide more accurate results. But generally lower execution times are preferred. Based on the execution time the optimal parameters where determined as population size of 50, generations of 100, thresholds of 3 and execution time as 58.7 sec.

4.5 Dice coefficient

parameter set	population size	generations	thresholds	Dice coefficient
set1	50	100	3	0.99
set2	50	150	4	0.99
set3	100	100	5	0.99
set4	100	150	3	0.99
set5	150	100	4	0.99
set6	100	200	5	0.99
set7	150	100	3	0.99
set8	150	150	4	0.99
set9	150	200	5	0.99
set10	200	100	3	0.99
set11	200	150	4	0.99
set12	200	200	5	0.99

Figure 4.7: Dice coefficients for each parameter values

The Dice Coefficient is a statistical measure that quantifies the similarity between two sets, specifically used to evaluate the performance of image segmentation. It calculates the overlap between the segmented image and a ground truth reference. The Dice Coefficient ranges from 0 to 1, where 1 indicates perfect overlap and 0 indicates no overlap. Higher dice coefficients reflect better segmentation accuracy, with more overlap between segmented and ground truth regions. All sets indicate high and almost similar dice coefficients indicating that all sets have good segmentation accuracy.

4.6 Optimal Thresholds

4.6.1 Based on PSNR (Peak Signal-to-Noise Ratio)

set6

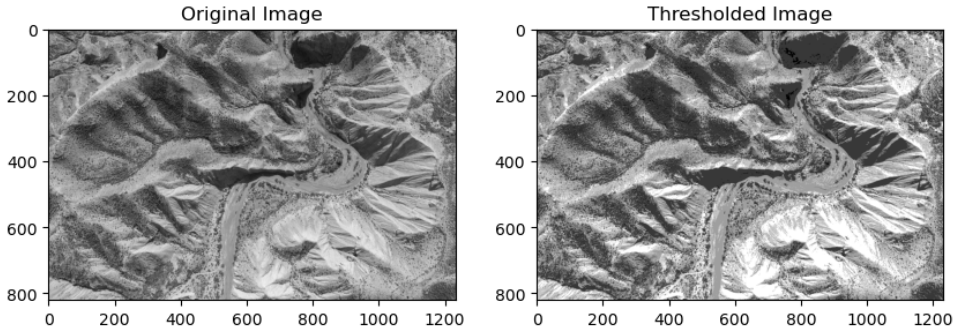


Figure 4.8: population size=100, generations=200, thresholds=5, PSNR=29.3

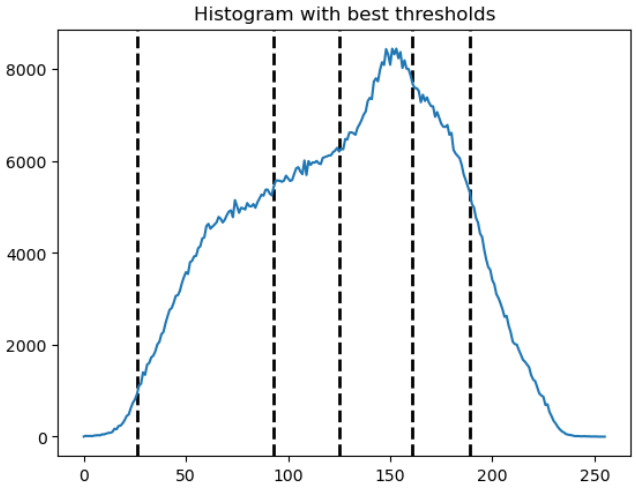


Figure 4.9: Optimal thresholds=45, 79, 130, 161, 176

4.6.2 Based on SSIM (Structural Similarity Index)

set3

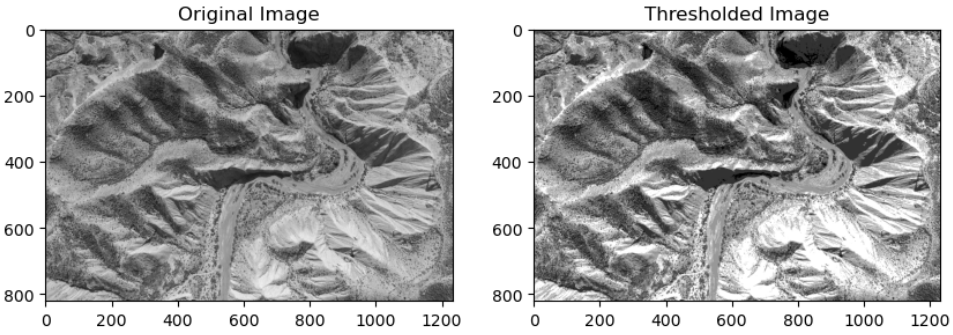


Figure 4.10: population size=100, generations=100, thresholds=5, SSIM=0.84

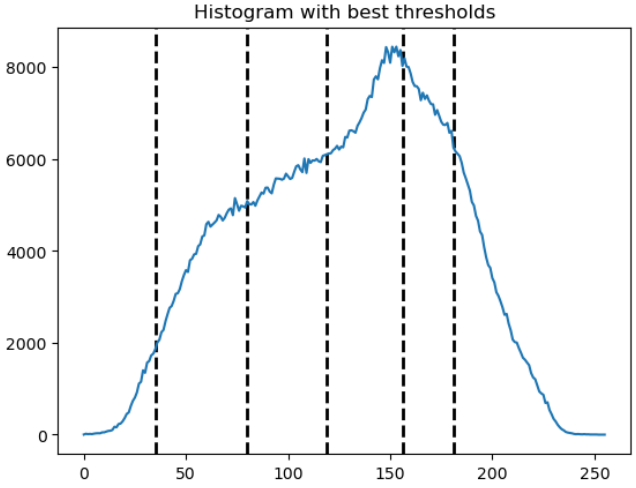


Figure 4.11: Optimal thresholds=35, 80, 119, 156, 181

set9

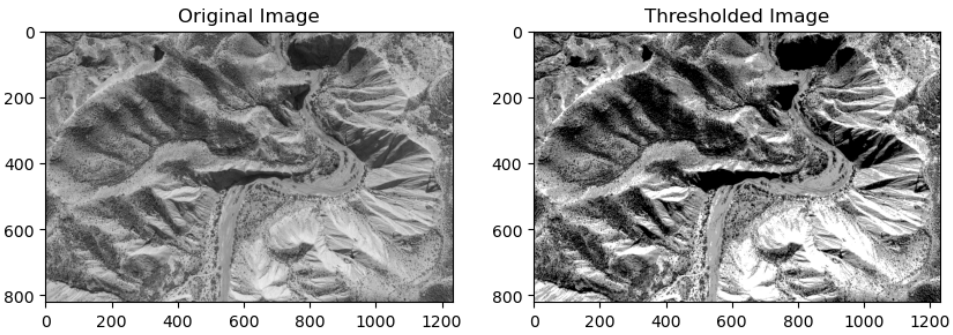


Figure 4.12: population size=150, generations=200, thresholds=5, SSIM=0.84

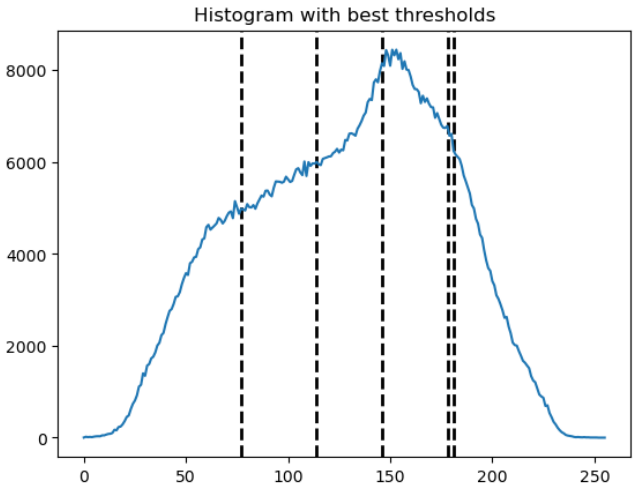


Figure 4.13: Optimal thesholds=77, 114, 146, 178, 181

4.6.3 Based on Execution Time

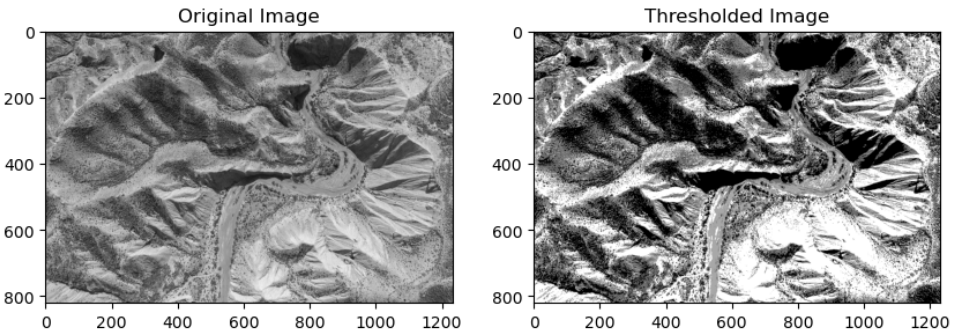


Figure 4.14: population size=50, generations=100, thresholds=3, execution time=58.7

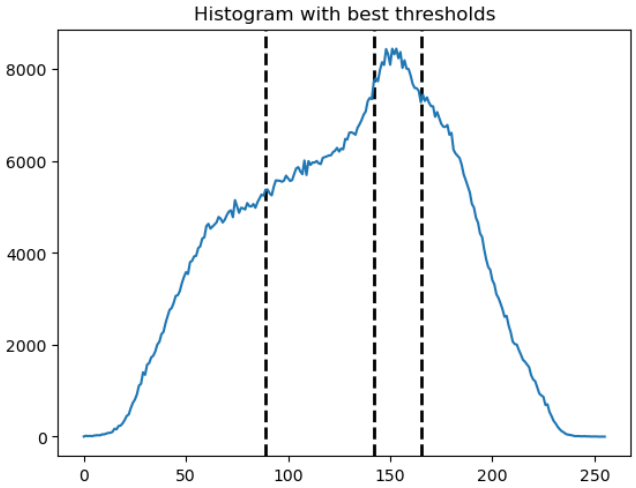


Figure 4.15: Optimal thresholds=$89, 142, 165$

Chapter 5

Conclusion

Through the genetic algorithm and multilevel thresholding a computational algorithm has been developed which has the capability to threshold images with good accuracy. An aerial image of terrain was utilized for algorithm tuning. The algorithm was built using 'python' interpreted programming language and tested for an aerial image. The image was converted to grayscale since multilevel thresholding requires grayscale images. A histogram was obtained for the intensities of pixels of the grayscale image and the peaks were observed. Four metrics were used for the evaluation of the algorithm performance namely Peak signal-to-noise ratio, Structural similarity index, execution time, Dice coefficient. The optimal parameters where determined using each metric. Evaluation through the peak signal-to-noise ratio determined that the optimal parameters are population size of 100, generations of 200 and thresholds of 5 which yeilded a PSNR of 29.3. The structural similarity index determined the two sets of optimal parameters. One as population size of 100, generations of 100 and thresholds of 5 which yielded SSIM as 0.84. The other as population size of 150, generations of 200, thresholds of 5 and SSIM of 0.84. Based on the execution time the optimal parameters where determined as population size of 50, generations of 100, thresholds of 3 and execution time as 58.7 sec. We can conclude that thresholding of an image yeilds the image to be more visually easy to be used for identification of topography. Since terrain images contain a vast amount of features and structures it is not possible to quickly identify the structures with the use of the default terrain images. But with the thresholded image terrain can be separated with ease by observation.

Appendix

Maxima Codes

```
######################### Required libraries ########################
import cv2
import numpy as np
import matplotlib.pyplot as plt
import random

# Load the image in default color scale
image = cv2.imread('/home/vaasala/Desktop/img/top_terrain_1.jpg', 1)
#visulizing the image
plt.imshow(image)



#################### Importing grayscale image ##################

gray_image = cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)#conversion of
    image to grayscale
plt.imshow(gray_image)#visualizing grayscale image


###############grayscale histogram ##########################

#array of histogram values of gray_image
histogram_a, bin_edges = np.histogram(gray_image, bins=256, range=(0,
    256))
#bins=256 , the pixel values range from 0 to 255 which contain 256
    pixel values
#range=(0,256), the range of the pixel values
plt.plot(histogram_a)#visualizing histogram


################## Parameters ############################

population_size = 100
generations = 200
thresholds_count = 5

#####Fitness function: Calculate between-class variance #######

def fitness(thresholds, gray_image):
    thresholds = sorted(thresholds) #sort the thresholds in
        increasing order
```

```
    thresholds = [0] + thresholds + [255] #Threshold list contains
       initial value 0 and final value 255
    total_variance = 0 #object that accumulates within class variance
        for each class

    for i in range(len(thresholds) - 1): #loops through each
       consecutive threshold
        lower, upper = thresholds[i], thresholds[i + 1] #consecutive
           thresholds
        mask = (gray_image >= lower) & (gray_image < upper) #Array of
            boolean values for the specified condition
        if np.sum(mask) == 0: #if the mask array values are NULL
           rejected
            continue
        class_mean = np.mean(gray_image[mask]) #extract only pixel
           values where mask array depicts true and calculate mean
        total_variance += np.sum(mask) * (class_mean - np.mean(gray_
           image)) ** 2
        #np.sum(mask) = count of pixels which are true
        #np.mean(image)=total mean of all pixel values
        #class_mean=mean of a single class
    return total_variance #return the total between-class variance


############### Generate initial population ############

def initialize_population(population_size, thresholds_count):
    threshold_range=range(1,255)#range of the thresholds, pixel
       intensities
    population=[]#empty list to hold population
    for _ in range(population_size): #iterate through population size
        random_thresholds=random.sample(threshold_range,thresholds_
           count)
        #generate random sample of threhsold values within range at
           the threshold count amount
        sorted_thresholds=sorted(random_thresholds) #sort thresholds
           in ascending order
        population.append(sorted_thresholds)#add sorted thresholds to
            the empty population list

    return population #return the population object




########## Selection: Roulette Wheel Selection ##########

def select(population, fitnesses):
    total_fitness = sum(fitnesses)#sum of fitness values in the
       chromosome
    probabilities=[]#initialize null object
    #iterating through the fitness values
    for f in fitnesses:
        probability=f/total_fitness
        probabilities.append(probability)
    num_individuals=len(population)#number of individuals in the
       population
```

```
    selected_index = np.random.choice(num_individuals, p=
        probabilities)#index selected at random based on probabilties
        p
    selected_individual = population[selected_index]#selected
        individual is the population values of selected index

    return selected_individual

###########Crossover ########################
def crossover(parent1, parent2):
    crossover_point = random.randint(1, len(parent1) - 1)#point where
         parents are divided into two
    #generates a random integer between two points
    child1 = parent1[:crossover_point] + parent2[crossover_point:]
    #addition of elements of parent1 elements upto crossover point to
         the parent2 elements beyond crossover point
    child2 = parent2[:crossover_point] + parent1[crossover_point:]
    #addition of parent2 elements upto crossover point to parent1
        elements beyond crossover point

    return child1, child2

########### Mutation ##########################
def mutate(thresholds):
    index = random.randint(0, len(thresholds) - 1) #selection of
        random position in the thresholds list for mutation
    thresholds[index] = random.randint(1, 254)#random integer
        selected within range introduced to index point

    return sorted(thresholds)

########## Main Genetic Algorithm ################

def genetic_algorithm(gray_image, population_size, generations,
    thresholds_count):

    population = initialize_population(population_size, thresholds_
        count)#generating initial random population

    for generation in range(generations):

        fitnesses=[]
        for individual in population:
            fitness_score=fitness(individual, gray_image)
            fitnesses.append(fitness_score)

        new_population = []
        for _ in range(population_size // 2):
            parent1 = select(population, fitnesses)
            parent2 = select(population, fitnesses)
            child1, child2 = crossover(parent1, parent2)
            new_population.extend([mutate(child1), mutate(child2)])
        population = new_population

    #finding set of thresholds with highest fitness values
    best_individual = None #initialization
    best_fitness = -float('inf') #lowest fitness score as
        initialization
```

```
    #iterating over each individual in the population
    for ind in population:
        current_fitness = fitness(ind,gray_image) #calculating
           fitness score for currrent individual
        #updating best fitness score and best individual
        if current_fitness > best_fitness:
            best_fitness=current_fitness
            best_individual = ind #holds individual with highest
               fitness score



    return best_individual

############ Run the genetic algorithm###############

best_thresholds = genetic_algorithm(gray_image, population_size,
   generations, thresholds_count)


########## Apply the best thresholds to the image ########

thresholded_image = np.zeros_like(gray_image) #creating object with
   same shape as gray_image initialized to zero

thresholds = [0] + best_thresholds + [255]#initializing thresholds
   list

for i in range(len(thresholds) - 1):
    lower = thresholds[i] #lower bound for the thresholds
    upper = thresholds[i+1] #upper bound for the threesholds
    mask = (gray_image >= lower) & (gray_image < upper) #create mask
       for pixels within current interval
    midpoint = (lower + upper) // 2 #calculation of midpoint of
       thresholds
    thresholded_image[mask] = midpoint #applyind midpoint to mask of
       thresholded_image


########## Display the result ####################

plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.title('Original Image')
plt.imshow(gray_image, cmap='gray')
plt.subplot(1, 2, 2)
plt.title('Thresholded Image')
plt.imshow(thresholded_image, cmap='gray')
plt.show()



######### visulaizing thresholds #############
#Best thresholds
print(best_thresholds)
#grayscale histogram
histogram_b, bin_edges = np.histogram(thresholded_image, bins=256,
   range=(0, 256))
```

```
#bins=256 , the pixel values range from 0 to 255 which contain 256
   pixel values
#range=(0,256), the range of the pixel values

#vertical lines for the thresholds
for threshold in best_thresholds:
    plt.axvline(x=threshold, color='black', linestyle='--', linewidth
       ='2', label=f'Thresholds')

plt.title('Histogram with best thresholds')
plt.plot(histogram_a)
plt.show



########## PSNR value ###############
import numpy as np

def calculate_psnr(original, thresholded):
    # MSE calculation
    mse = np.mean((original - thresholded) ** 2)

    # PSNR calculation
    if mse == 0:
        return float('inf')  # Infinite PSNR if no error
    max_pixel_value = 255.0
    psnr = 10 * np.log10((max_pixel_value ** 2) / mse)

    return psnr

# Execution
psnr_value = calculate_psnr(gray_image, thresholded_image)
print("PSNR:", psnr_value)


############# SSIM #################
from skimage.metrics import structural_similarity as ssim

def calculate_ssim(original, thresholded):
    # SSIM computation
    ssim_value = ssim(original, thresholded, data_range=thresholded.
       max() - thresholded.min())

    return ssim_value

# Execution
ssim_value = calculate_ssim(gray_image, thresholded_image)
print("SSIM:", ssim_value)



######## execution time #############
import time

def measure_execution_time(gray_image, population_size, generations,
   thresholds_count):
    # Start Timer
    start_time = time.time()
```

```
    # Run Genetic Algorithm
    best_thresholds = genetic_algorithm(gray_image, population_size,
        generations, thresholds_count)

    # Stop Timer
    end_time = time.time()

    # Execution Time calculation
    execution_time = end_time - start_time

    return execution_time

# Execution
execution_time = measure_execution_time(gray_image, population_size,
    generations, thresholds_count)
print("Execution Time:", execution_time, "seconds")


####### dice coefficient ############
def calculate_dice_coefficient(original, thresholded):
    # converting images to binary
    original_bin = original > 0  # Convert to binary based on a
        threshold
    thresholded_bin = thresholded > 0


    intersection = np.logical_and(original_bin, thresholded_bin).sum
        ()
    union = original_bin.sum() + thresholded_bin.sum()

    # Dice coefficient computation
    dice_coefficient = (2.0 * intersection) / union

    return dice_coefficient

# Execution
dice_coefficient_value = calculate_dice_coefficient(gray_image,
    thresholded_image)
print("Dice Coefficient:", dice_coefficient_value)
```

Bibliography

[1] Salem Saleh Al-Amri, Namdeo V Kalyankar, et al. Image segmentation by using threshold techniques. *arXiv preprint arXiv:1005.4020*, 2010.

[2] Pedro Ventura de Oliveira and Keiji Yamanaka. Image segmentation using multilevel thresholding and genetic algorithm: An approach. In *2018 2nd international conference on data science and business analytics (ICDSBA)*, pages 380–385. IEEE, 2018.

[3] John H Holland. Genetic algorithms. *Scientific american*, 267(1):66–73, 1992.

[4] Oliver Kramer and Oliver Kramer. *Genetic algorithms*. Springer, 2017.

[5] Ping-Sung Liao, Tse-Sheng Chen, Pau-Choo Chung, et al. A fast algorithm for multilevel thresholding. *J. Inf. Sci. Eng.*, 17(5):713–727, 2001.

[6] Melanie Mitchell. *An introduction to genetic algorithms*. MIT press, 1998.

[7] Nobuyuki Otsu et al. A threshold selection method from gray-level histograms. *Automatica*, 11(285-296):23–27, 1975.

[8] P Daniel Ratna Raju and G Neelima. Image segmentation by using histogram thresholding. *International Journal of Computer Science Engineering and Technology*, 2(1):776–779, 2012.

[9] Vitorino Ramos and Fernando Muge. Image colour segmentation by genetic algorithms. *arXiv preprint cs/0412087*, 2004.

[6] [4] [3] [5] [1] [8] [2] [9] [7]