GLOWWORM SWARM OPTIMIZATION ALGORITHM FOR MULTI- THRESHOLD IMAGE SEGMENTATION

A Paper
Submitted to the Graduate Faculty
of the
North Dakota State University
of Agriculture and Applied Science

By

Shashi Kanth Kasam

In Partial Fulfillment of the Requirements
for the Degree of
MASTER OF SCIENCE

Major Department:
Computer Science

March 2016

Fargo, North Dakota

North Dakota State University
Graduate School

Title

GLOWWORM SWARM OPTIMIZATION ALGORITHM FOR MULTI-THRESHOLD IMAGE SEGMENTATION

By

SHASHI KANTH KASAM

The Supervisory Committee certifies that this ***disquisition*** complies with North Dakota State University's regulations and meets the accepted standards for the degree of

MASTER OF SCIENCE

SUPERVISORY COMMITTEE:

Dr. Simone Ludwig
Advisor

Dr. Saeed Salem

Dr. Na Gong

Approved:

3/14/2016
Date

Dr. Brian Slator
Department Chair

ABSTRACT

Glowworm Smarm Optimization (GSO), one of the nature-inspired swarm intelligence algorithms is used for finding solutions to optimization problems. Glowworms emit light to attract its mates and mates choosing the brighter member is the basis for this algorithm. We apply this algorithm in conjunction with traditional multi-threshold image segmentation by Otsu method to arrive at our results. The glowworm selection of mates and the movement of these has been mapped to a scientific algorithm to solve the problem and make multi-level threshold image segmentation a relatively efficient one. This algorithm has been implemented in MATLAB and the results are thus presented.

ACKNOWLEDGEMENTS

I would like to thank Dr. Simone Ludwig for her timely guidance and help in every aspect of this study and analysis. Her clear and precise explanation of GSO algorithm and its application to the image segmentation problem has been a vital part of my implementation in MATLAB. I would also like to thank my committee members for their time and invaluable advice which has helped me bring this paper to its closure.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1. INTRODUCTION

Swarm Intelligence is the ability to perform and achieve complex goals by the actions of individuals in the swarm exploiting local information. As observed in the nature among ants, bees, flock of birds, etc., this is very interesting aspect in the way that individuals do not have the knowledge of complex group behavior but still achieve incredible goals [1]; more so, without any centralized command control. Scientists generally describe this as self-organizing decentralized system where the goal is achieved by simple interaction between agents following simple rules. Some of the algorithms that are developed based on this kind of behaviors in the nature are: Particle Swarm Optimization (PSO), Ant Colony Optimization (ACO), Artificial Bee Colony algorithm (ABC), etc. Swarm Intelligence based algorithms can be and are being used in many application areas like robotics, medicine, data mining, communications, etc.

GSO – Glowworm Swarm Optimization is also one such nature inspired swarm intelligence algorithm that tries to follow or simulate the behavior of glowworms. Glowworms that emit light to attract its partner that is at a distance is the main concept of this algorithm. In nature, the female glowworms choose the brightest light emitting glowworm for its perpetuation. Using the same concept we try to find local optima of multi modal functions against a global optima with other algorithms.

The GSO algorithm was introduced by Krishnanand and Ghose in 2005 [2,3] at Guidance, Control and Decision Systems in the Aerospace Engineering Department of the Indian Institute of Science, Bangalore, India. Eventually this algorithm found its way into many application areas, and several papers have been published by various authors in the literature of Swarm Intelligence.

The specific area where we are using this algorithm in this paper is Image Segmentation. Image segmentation is the process of dividing a digital image to multiple segments where each

segment of image holds pixels of some common characteristics. The objective of image segmentation is generally to represent an image in a more meaningful way that is easy to analyze [4]. This particular area of image segmentation has been a classical problem from 1970s and many algorithms have been proposed since then.

In the image segmentation process, the main content that lies in an image is decomposed to simplify its representation. There have been multiple techniques proposed and implemented in the area of image segmentation. Some of those methods are thresholding, clustering, edge detection, histogram-based methods, etc.

Typically, segmentation depends on the two properties of the image – similarity and discontinuity [5]. Based on these two properties, we have two image segmentation categories. They are region based and edge based. In this paper, we use with multi-level thresholding by the Otsu method implementing GSO algorithm to see how effective and efficient this method is as claimed by the authors in [6]. Also, we implemented this algorithm for color images and had our results presented.

2. PROBLEM STATEMENT

The problem identification of this work is defined as "***Multi-level threshold image segmentation by GSO+Otsu algorithm for grey and colored images***".

The main problem of image segmentation by thresholds is that it considers only the luminance values and the relationship between the neighboring pixels is ignored because of which repeated pixels of the same luminance can be easily found and some pixels that are independent can be missed [7]. Hence, a typical image thresholding algorithm is used in conjunction with other optimization algorithm.

One of the most popularly used image segmentation by thresholding is the Otsu algorithm. This algorithm was initially proposed for only two or binary classification of gray area into foreground or background. This was further developed to include multiple gray levels in the image and segment by multiple threshold values to bring in more effective image segmentation.

The traditional image segmentation by Otsu suffers from the efficiency problem [5]. Efficiency in terms of overall effect on the image and the processing time. So, now we evaluate the claim made by the authors [6] by implementing this algorithm in MATLAB and recording the execution times of gray and color images though processing of color images were not mentioned in this paper. So, taking this algorithm little ahead, image segmentation of color images by taking R (red), G (green) and B (blue) values is done in this paper following the same algorithm for which results are recorded and presented.

The current paper does not evaluate other segmentation algorithms practically by any implementation but rather evaluates the claims made by the authors in [6]. And, the results definitely reinforce their claims as per our recording of results.

3. BACKGROUND

3.1. Introduction to Evolutionary Computing

The term of evolutionary computation in the field of computer science is derived on the basis of the Darwinian principles. The rule – "Survival of the fittest" is followed by all the evolutionary computation techniques today.

Evolutionary computing is the name given to the class of problem-solving techniques that are inspired from the biological process of natural selection and inheritance.

Though this area has seen its first people introducing various methods across the globe and calling it by different names like 'Evolutionary Programming' by Lawrence J Fogel of US in 1960 [8], 'Genetic Algorithm' by John Henry Holland in 1970s [34], 'Evolution Strategies' by Ingo Rechenberg and Hans-Paul Schwefel of Germany [35]; it is not until the early nineties that these were recognized as the different forms of same technique called evolutionary computing. However, these three areas developed separately for around 15 years. Since the early nineties, nature inspired algorithms have been a significant part of evolutionary computation [9].

Evolutionary algorithms are about random search and optimization techniques that are applicable to different kind of problems that are multi-modal, non-continuous, multi-objective, noisy and dynamic. Novel methods have been introduced in this field making these one of the best methods in the Derivative Free Optimization for problems of higher dimensions [10].

Algorithm 1. Evolutionary Algorithm

Begin

1. Initialize population with random candidate solutions;
2. Evaluate each candidate;
3. Repeat Until (Termination Condition is satisfied)
4. Do
    a. Select parents;
    b. Recombine pairs of parents;
    c. Mutate the resulting off spring;
    d. Evaluate new candidates;
    e. Select individuals for the next generation;

End do

End

Here are some of the widely used optimization algorithms in the field of evolutionary computing.

3.2. Ant Colony Optimization (ACO)

Marco Dorigo is the person who introduced Ant Colony Optimization in 1992 [11]. In the wider field of swarm intelligence, this is one of the most successful techniques. Inspired by the ants behavior in finding food and laying path from the colony to the food source, computational problems are also reduced to figuring out good paths through graphs by a probabilistic technique. Discrete optimization problems like the travelling salesman is one of the most successful application of Ant Colony Optimization.

In the real world, ants that move randomly (initially) leave pheromone trails on their return path to their colony. When other ants find that path, they would follow the trail instead of moving randomly. When they do so, these ants too leave pheromone trails along the path reinforcing the trail.

Over time, the pheromone trail evaporates. This means that if the ants do not move along that path, it becomes weaker and less attractive for the other ants to follow. The more the time it takes for an ant to travel along the path, the lesser would be its pheromone trail strength along the path. So, by comparison, a path that is shorter between the food source and the colony would have greater pheromone density. Hence an ant that finds a good path leads all others towards that path.

In terms of optimization, the pheromone evaporation avoids the local optima convergence, which is definitely a significant advantage. Had that not been the case, the first chosen path by the ants would be more likely to be followed by the other ants, restricting the exploration of solution search space. The very idea of Ant Colony Optimization algorithm is to simulate this behavior of ants, which starts with random initialization of ants that walk around the search space graph to find a solution to the optimization problem.

Algorithm 2. Ant Colony Optimization Algorithm [13]

1. Initialization
2. An initial pheromone value is assigned to every edge of the search space graph and randomly an ant(s) is located in that search space
3. Looping through population
    a. Probabilistic transition: Move ant over the solution space accordingly to a given probabilistic transition rule
    b. Fitness evaluation: Evaluate the goodness of fit for the solution provided by the ant
    c. Pheromone update: Reinforce pheromone level of the edges for good solutions and reduce (evaporation) for those with not-so-good solutions
4. Repeat step 2 till the criteria of convergence is met.

3.3. Artificial Bee Colony Algorithm (ABC)

Artificial bee colony algorithm (ABC) is one of the swarm based meta-heuristic algorithm introduced by Karaboga in 2005 [14] that mimics the foraging behavior of honey bees. It is used for optimizing numerical problems and is based on the model proposed by Tershko and Loengarov in 2005 [15]. In this ABC algorithm there are three groups of bees -employed bees, onlookers and scouts. Employed bees are the ones that are directly connected to the food source. Onlooker bees are the ones that watch the dance of employed bees and choose the food source. Employed bees whose food source is exhausted then become scout bees searching for new source of food. Going by the exact functioning of these bees, the algorithm is also divided into three phases. Food sources are initially discovered by scout bees. Then, employed and onlooker bees exploit the food source in the local neighborhood till the food source is exhausted and once it is exhausted, employed bees become scout bees searching for new food source.

This algorithm presumes that each employed bee can be associated with only a single food source and so the number of employed bees is always equal to the number of food sources. The location of a food sources represent a possible solution in the search space. The nectar of food source corresponds to the fitness or quality of the solution. With iterative running of the above mentioned phases, randomly discovered initial solution vectors move towards better solutions abandoning the poorer ones. This algorithm has a good search and exploitation capability [16].

Algorithm 3. Artificial Bee Colony Algorithm

1. Initialization Phase
2. Repeat
    a. Employed Bees Phase
    b. Onlooker Bees Phase
    c. Scout Bees Phase
    d. Memorize the best solution achieved so far

   Until (Cycle=Maximum Cycle Number or a Maximum CPU time)
3. End

3.4. Genetic Algorithm

Genetic Algorithm (GA) is a method for solving both constrained and unconstrained optimization problems based on a natural selection process that mimics biological evolution. The algorithm repeatedly modifies a population of individual solutions [17]. At each step, the genetic algorithm randomly selects individuals from the current population and uses them as parents to produce the children for the next generation. Over successive generations, the population "evolves" toward an optimal solution.

You can apply the genetic algorithm to solve problems that are not well suited for standard optimization algorithms, including problems in which the objective function is discontinuous, non-differentiable, stochastic, or highly nonlinear.

At the beginning of a run of a genetic algorithm, a large population of random chromosomes is created. Each one, when decoded will represent a different solution to the problem at hand. Let us say there are N chromosomes in the initial population. Then, the following steps are repeated until a solution is found.

Algorithm 4. Genetic Algorithm

1. Test each chromosome to see how good it is at solving the problem at hand and assign a fitness score accordingly. The fitness score is a measure of how good that chromosome is at solving the problem to hand.
2. Select two members from the current population. The chance of being selected is proportional to the chromosomes fitness. Roulette wheel selection is a commonly used method.
3. Dependent on the crossover rate crossover the bits from each chosen chromosome at a randomly chosen point.
4. Step through the chosen chromosomes bits and flip dependent on the mutation rate.
5. Repeat steps 2, 3, 4 until a new population of N members has been created.

3.5. Genetic Programming

One of the main challenges of computer science is to get a computer to do what needs to be done, without telling it how to do it. Genetic programming addresses this challenge by providing a method for automatically creating a working computer program from a high-level problem statement of the problem [18]. Genetic programming achieves this goal of automatic programming (also sometimes called program synthesis or program induction) by genetically breeding a population of computer programs using the principles of Darwinian natural selection and biologically inspired operations.

Genetic programming is a domain-independent method that genetically breeds a population of computer programs to solve a problem. Specifically, genetic programming iteratively transforms a population of computer programs into a new generation of programs by applying analogs of naturally occurring genetic operations. The genetic operations include crossover (sexual recombination), mutation, reproduction, gene duplication, and gene deletion. It is a specialization

of genetic algorithms (GA) where each individual is a computer program [19]. It is a machine learning technique used to optimize a population of computer programs according to a fitness landscape determined by a program's ability to perform a given computational task.

The main difference between genetic programming and genetic algorithms is the representation of the solution. Genetic programming creates computer programs in the lisp or scheme computer languages as the solution. Genetic algorithms create a string of numbers that represent the solution.

Algorithm 5. Genetic Programming Algorithm

1. Generate an initial population of random compositions of the functions and terminals of the problem (computer programs).
2. Execute each program in the population and assign it a fitness value according to how well it solves the problem.
3. Create a new population of computer programs.
    a. Copy the best existing programs.
    b. Create new computer programs by crossover.
    c. Create new computer programs by mutation.
4. Loop step 3 until exit criteria is met.
5. The best computer program that appeared in any generation, the best-so-far solution, is designated as the result of genetic programming.

3.6. Differential Evolution

The term differential evolution was coined by R. Storn and K. Price in [20] as a product of their search to apply simulated annealing to the Chebyshev polynomial fitting problem. Differential evolution is a stochastic parallel direct search evolution strategy optimization method that is fairly fast and reasonably robust. The method of differential evolution’s functioning is similar to genetic algorithm’s approach and is summarized in the pseudocode at the end.

Differential Evolution like the method of Genetic Algorithms allows each successive generation of solutions to 'evolve' from the previous generations strengths. The method of differential evolution can be applied to real-valued problems over a continuous space with much more ease than a genetic algorithm. The idea behind the method of differential evolution is that the difference between two vectors yields a difference vector, which can be used with a scaling factor to traverse the search space. The strength of differential evolution's approach is that it often displays better results than a genetic algorithm and other evolutionary algorithms for specific domain problems and can be easily applied to other wide variety of real valued problems despite noisy, multi-modal, multi-dimensional spaces, which usually make the problems very difficult for optimization.

Differential evolution is capable of handling non-differentiable, nonlinear and multimodal objective functions [21]. It has been used to train neural networks having real and constrained integer weights. In a population of potential solutions within an n-dimensional search space, a fixed number of vectors are randomly initialized, then evolved over time to explore the search space and to locate the minima of the objective function. At each iteration, called a generation, new vectors are generated by the combination of vectors randomly chosen from the current population (mutation). The out-coming vectors are then mixed with a predetermined target vector. This operation is called recombination and produces the trial vector. Finally, the trial vector is accepted for the next generation if and only if it yields a reduction in the value of the objective function.

Algorithm 6. Differential Evolution Algorithm

Begin

1. Randomly generate an initial population of solutions.
2. Calculate the fitness of the initial population.
3. Repeat
    a. For each parent, select three solutions at random.
    b. Create one offspring using the differential evolution operators.
    c. Do this a number of times equal to the population size.
    d. For each member of the next generation
    e. If offspring(x) is more fit than parent(x), parent(x) is replaced.

    Until a stop condition is satisfied.

End

3.7. Particle Swarm Optimization (PSO)

Particle swarm optimization (PSO), another stochastic optimization technique that is population based, was established by Dr. Eberhart and Dr. Kennedy in 1995 [22]. It was motivated by social actions of bird flocking or fish schooling. PSO shares many resemblances with evolutionary computation methods such as Genetic Algorithms (GA).

PSO is a metaheuristic approach that goes by few or no presumptions about the problem being enhanced and can look for very huge areas of applicable alternatives. However, metaheuristics such as PSO do not assure if the optimal result would be ever discovered. More particularly, PSO does not require the problem be differentiable as needed by traditional optimization techniques such as gradient descent and quasi-newton techniques [23].

A fundamental version of the PSO criteria performs by having a populace (called a swarm) of solution candidates (called particles). These particles are shifted around in the search-space according to update equations. The motions of the particles are governed by their own best known

place in the search-space as well as the whole swarm's best known place. When enhanced positions are being found these will then come to help the motion of the swarm. The procedure is recurring and by doing so it is expected, but not assured, that an effective solution will ultimately be found.

Algorithm 7. Particle Swarm Optimization Algorithm [13]

1. For each particle
   Initialize particle
   End
2. For each particle
   If the fitness value is better than the best fitness value (pbest*) in history
   Set current value as the new pbest
   End
3. Choose the particle with the best fitness value of all the particles as the gbest**
4. For each particle
   a. Calculate particle velocity according to a predefined formula
   b. Update particle position according to a predefined formula
   End
5. While maximum iterations or minimum error criteria is not attained

* ‘pbest’ is the particle's best value

** ‘gbest’ is the global best among all the particles in the swarm

4. GLOWWORM SWARM OPTIMIZATION (GSO)

Glowworm swarm optimization (GSO) is a novel algorithm for the simultaneous computation of multiple optima of multimodal functions. The algorithm shares a few features with some better known swarm intelligence based optimization algorithms, such as ant colony optimization and particle swarm optimization, but with several significant differences [2]. The agents in GSO are thought of as glowworms that carry a luminescence quantity called luciferin along with them. The glowworms encode the fitness of their current locations, evaluated using the objective function, into a luciferin value that they broadcast to their neighbors. The glowworm identifies its neighbors and computes its movements by exploiting an adaptive neighborhood, which is bounded above by its sensor range. Each glowworm selects, using a probabilistic mechanism, a neighbor that has a luciferin value higher than its own and moves toward it. These movements which are based only on local information and selective neighbor interactions enable the swarm of glowworms to partition into disjoint subgroups that converge on multiple optima of a given multimodal function.

In GSO, a swarm of agents are initially randomly distributed in the search space. Agents are modeled after glowworms and will be called glowworms in the following of this paper. Accordingly, they carry a luminescent quantity called luciferin along with them. The glowworms emit a light whose intensity is proportional to the associated luciferin and interact with other agents within a variable neighborhood. In particular, the neighborhood is defined as a local-decision domain that has a variable neighborhood range r^i_d bounded by a radial sensor range r_s ($0 < r^i_d \leq r_s$). A glowworm i considers another glowworm j as its neighbor if j is within the neighborhood range of i and the luciferin level of j is higher than that of i. The decision domain enables selective neighbor interactions and aids in formation of disjoint sub-swarms. Each glowworm is attracted

by the brighter glow of other glowworms in the neighborhood. Agents in GSO depend only on information available in their neighborhood to make decisions. For instance, in Figure 1(a), agent i is in the sensor range of (and is equidistant to) both j and k. However, j and k have different neighborhood sizes, and only j uses the information of i. Figure 1(b) shows the directed graph based on the relative luciferin level of each agent and on the availability of only local information. Each glowworm selects, using a probabilistic mechanism, a neighbor that has a luciferin value higher than its own and moves toward it.

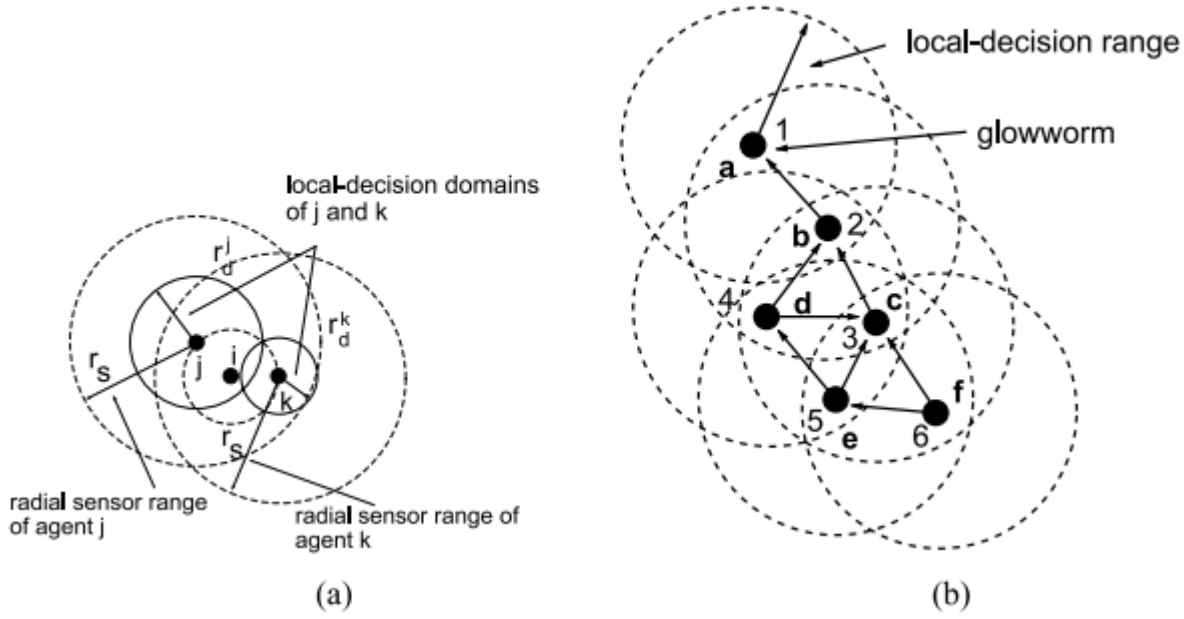


Figure 1. Glowworms movement in the search space [2]
(a) $r^k_d < d(i,k) = d(i,j) < r^j_d < rs$. Agent i is in the sensor range of (and is equidistant to) both j and k. However, j and k have different neighborhood sizes, and only j uses the information of i.
(b) Directed graph based on the relative luciferin level of each agent and availability of only local information. Agents are ranked according to the increasing order of their luciferin values. For instance, agent 'a', whose luciferin value is highest, is ranked '1' in the figure.

4.1. Steps in GSO algorithm

GSO algorithm [2, 3] starts by placing a population of n glowworms randomly in the search space so that they are well dispersed. Initially, all the glowworms contain an equal quantity of luciferin l_0. Each iteration consists of a luciferin-update phase followed by a movement phase based on a transition rule.

1. Initialize parameters: n individuals are randomly placed in feasible region, l_0 accounts for initial luciferin value, r_0 for dynamic decision domain, s for step, n_t for threshold in domain, ρ for luciferin elimination coefficient, γ for fluorescein update coefficient, β for update coefficient of domain, r_s for maximal searching radius, and t for iteration number.
2. Luciferin-update phase: The luciferin update depends on the function value at the glowworm position. During the luciferin-update phase, each glowworm adds, to its previous luciferin level, a luciferin quantity proportional to the fitness of its current location in the objective function domain. Also, a fraction of the luciferin value is subtracted to simulate the decay in luciferin with time. The luciferin update rule is given by

$$l_i(t) = (1-\rho)\, l_i(t-1) + \gamma J(x_i(t)) \tag{1}$$

where $l_i(t)$ represents the luciferin level associated with glowworm i at time t, ρ is the luciferin decay constant ($0 < \rho < 1$), γ is the luciferin enhancement constant, and $J(x_i(t))$ represents the value of the objective function at agent i's location at time t.

3. In each r^i_d (t), select higher luciferin value individuals forming a set of $N_i(t)$ neighborhood. Hence,

$$N_i(t) = \left\{ j : \left\| x_j(t) - x_i(t) \right\| \le r^i_d(t) ; l_i(t) \le l_j(t) \right\} \tag{2}$$

4. Movement phase: During the movement phase, each glowworm decides, using a probabilistic mechanism, to move toward a neighbor that has a luciferin value higher than its own. That is,

glowworms are attracted to neighbors that glow brighter. Figure 1(b) shows the directed graph among a set of six glowworms based on their relative luciferin levels and availability of only local information. For instance, there are four glowworms (a, b, c, and d) that have relatively more luciferin than glowworm e. Since e is located in the sensor-overlap region of c and d, it has only two possible directions of movement. For each glowworm i, the probability of moving toward a neighbor j is given by

$$p_{ij}(t) = \frac{l_j(t) - l_i(t)}{\sum_{k \in N_i(t)} l_k(t) - l_i(t)} \tag{3}$$

5. The position of individual i can be updated as

$$x_i(t+1) = x_i(t) + s\left(\frac{x_j(t) - x_i(t)}{\left\|x_j(t) - x_i(t)\right\|}\right). \tag{4}$$

6. The dynamic decision domain can be updated as

$$r_d^i(t+1) = \min\left\{r_s, \max\left\{0, r_d^i(t), \beta\left(n_t - \left|N_i(t)\right|\right)\right\}\right\} \tag{5}$$

Algorithm 8. Glowworm Swarm Optimization Algorithm [3]

GLOWWORM SWARM OPTIMIZATION (GSO) ALGORITHM

Set number of dimensions $= m$
Set number of glowworms $= n$
Let s be the step size
Let $x_i(t)$ be the location of glowworm i at time t
deploy_agents_randomly;
for $i = 1$ to n do $\ell_i(0) = \ell_0$
$r_d^i(0) = r_0$
set maximum iteration number $=$ *iter_max*;
set $t = 1$;
while ($t \leq$ *iter_max*) do:
{
  for each glowworm i do: % Luciferin-update phase
    $\ell_i(t) = (1-\rho)\ell_i(t-1) + \gamma J(x_i(t))$

  for each glowworm i do: % Movement-phase
  {
    $N_i(t) = \{j : d_{ij}(t) < r_d^i(t); \ell_i(t) < \ell_j(t)\}$;
    for each glowworm $j \in N_i(t)$ do:
      $p_{ij}(t) = \frac{\ell_j(t)-\ell_i(t)}{\sum_{k\in N_i(t)} \ell_k(t)-\ell_i(t)}$;
    $j = select\_glowworm(\vec{p})$;
    $x_i(t+1) = x_i(t) + s(\frac{x_j(t)-x_i(t)}{\|x_j(t)-x_i(t)\|})$
    $r_d^i(t+1) = \min\{r_s, \max\{0, r_d^i(t) + \beta(n_t - |N_i(t)|)\}\}$;
  }
  $t \leftarrow t+1$;
}

5. IMAGE SEGMENTATION

Image Segmentation is the division of an image into meaningful structures. Image segmentation, is often an essential step in image analysis, object representation, visualization, and many other image processing tasks.

Perfect image segmentation, i.e., each pixel is assigned to the correct object segment– is a goal that cannot usually be achieved. Indeed, because of the way a digital image is acquired, this may be impossible, since a pixel may straddle the “real” boundary of objects such that it partially belongs to two (or even more) objects. Most methods presented here –indeed most current segmentation methods– only attempt to assign a pixel to a single segment, which is an approach that is more than adequate for most applications. Methods that assign a segment probability distribution to each pixel are called probabilistic. This class of methods is theoretically more accurate, and applications where a probabilistic approach is the only approach accurate enough for specific object measurements can easily be named. However, probabilistic techniques add considerable complexity to segmentation –both in the sense of concept and implementation– and as such are still little used.

Perfect image segmentation is also often not reached because of the occurrence of over-segmentation or under-segmentation. In the first case, pixels belonging to the same object are classified as belonging to different segments. A single object may be represented by two or more segments. Generally, a "good" complete segmentation must satisfy the following criteria [24]:

1. All pixels have to be assigned to regions.
2. Each pixel has to belong to a single region only.
3. Each region is a connected set of pixels.
4. Each region has to be uniform with respect to a given predicate.

5. Any merged pair of adjacent regions has to be non-uniform.

A great variety of segmentation methods has been proposed in the past decades, and some categorization is necessary to present the methods properly here. A disjoint categorization does not seem to be possible though, because even two very different segmentation approaches may share properties that defy singular categorization.

Broadly, segmentation techniques can be classified into these categories [24], however in practicality hybrids of these methods can also exist in specific algorithms. Histogram thresholding and slicing techniques are used to segment the image. They may be applied directly to an image, but can also be combined with pre- and post-processing techniques.

5.1. Edge based segmentation

With this technique, detected edges in an image are assumed to represent object boundaries, and used to identify these objects.

5.2. Region based segmentation

Where an edge based technique may attempt to find the object boundaries and then locate the object itself by filling them in, a region based technique takes the opposite approach, e.g., by starting in the middle of an object and then "growing" outward until it meets the object boundaries.

5.3. Clustering techniques

Although clustering is sometimes used as a synonym for segmentation techniques, we use it here to denote techniques that are primarily used in exploratory data analysis of high-dimensional measurement patterns. In this context, clustering methods attempt to group together patterns that are similar in some sense. This goal is very similar to what we are attempting to do when we segment an image, and indeed some clustering techniques can readily be applied to image segmentation.

5.4. Threshold Based Segmentation

Thresholding is probably the most frequently used technique to segment an image. The thresholding operation is a grey value remapping operation g defined by,

$$g(v) = \begin{cases} 0 & \text{if} \quad v < t \\ 1 & \text{if} \quad v \geqslant t, \end{cases} \tag{6}$$

where 'v' represents a grey value, and t is the threshold value. Thresholding maps a grey-valued image to a binary image. After the thresholding operation, the image has been segmented into two segments, identified by the pixel values 0 and 1, respectively. If we have an image which contains bright objects on a dark background, thresholding can be used to segment the image.

Generally, the non-contextual thresholding may involve two or more thresholds as well as produce more than two types of regions such that ranges of input image signals related to each region type are separated with thresholds. The question of thresholding is how to automatically determine the threshold value [24]. There are many kinds of threshold detection methods like P-tile thresholding, Optimal thresholding, Mixture modelling, adaptive thresholding etc.,

See Figure 2 on the next page for an example. Since, in many types of images the grey values of objects are very different from the background value, thresholding is often a well-suited method to segment an image into objects and background. If the objects are not overlapping, then we can create a separate segment from each object by running a labelling algorithm on the thresholded binary image, thus assigning a unique pixel value to each object.

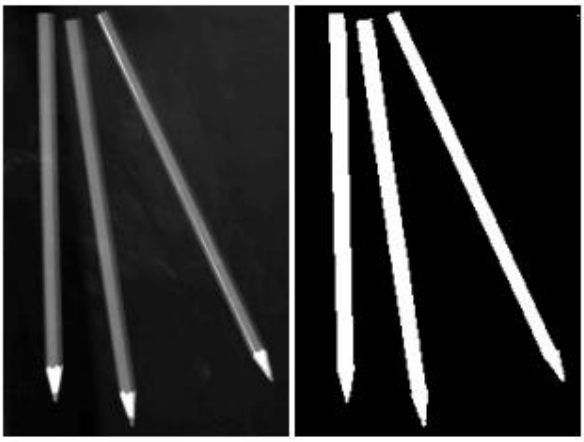


Figure 2. Example of segmentation by thresholding[1].

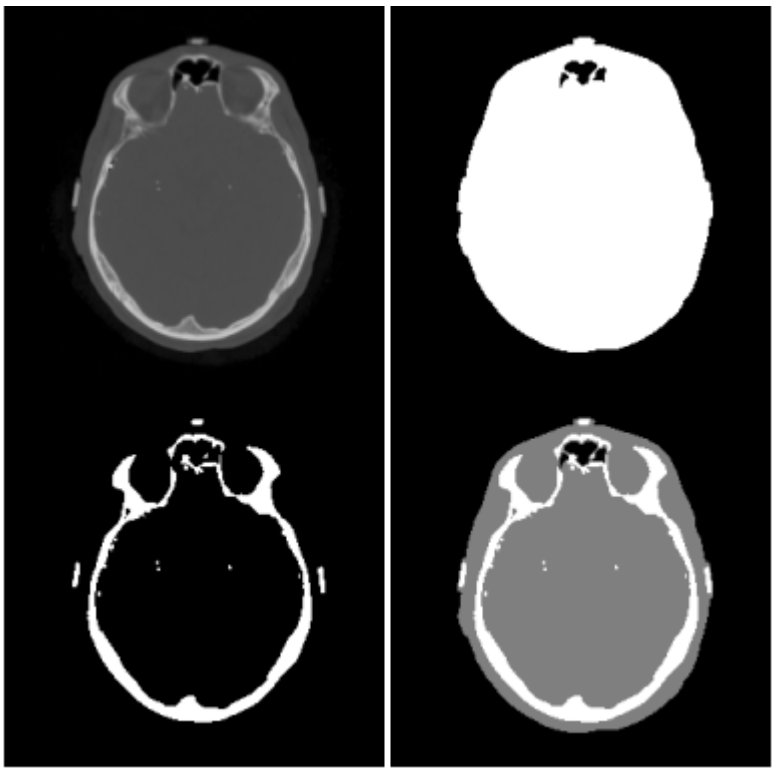
On the left in Figure 2 is the original image with bright objects (the pencils) on a dark background. On the right is the segmented image where an appropriate threshold value is used to segment the image into objects (segment with value1) and background (segment with value 0). Many methods exist to select a suitable threshold value for a segmentation task. Perhaps the most common method is to set the threshold value interactively; the user manipulating the value and reviewing the thresholding result until a satisfying segmentation has been obtained. The histogram is often a valuable tool in establishing a suitable threshold value.

When several desired segments in an image can be distinguished by their grey values, threshold segmentation can be extended to use multiple thresholds to segment an image into more than two segments: all pixels with a value smaller than the first threshold are assigned to segment 0, all pixels with values between the first and second threshold are assigned to segment 1, all pixels with values between the second and third threshold are assigned to segment 2, etc. If n thresholds (t1, t2,...,tn) are used, then

[1] Reprinted and referenced from "http://www.cs.uu.nl/docs/vakken/ibv/reader/chapter10.pdf

$$g(v) = \begin{cases} 0 & \text{if} \quad v < t_1 \\ 1 & \text{if} \quad t_1 \leqslant v < t_2 \\ 2 & \text{if} \quad t_2 \leqslant v < t_3 \\ \vdots & \vdots \quad \vdots \\ n & \text{if} \quad t_n \leqslant v. \end{cases} \tag{7}$$

(a) (b)



(c) (d)

Figure 3. Example of multiple thresholds for segmentation[2]
(a) Original image (b) Thresholding result after using a low threshold value to segment the image into head and background pixels. (c) Result after using a higher value to segment the bone pixels. (d) Result after using both thresholds at once.

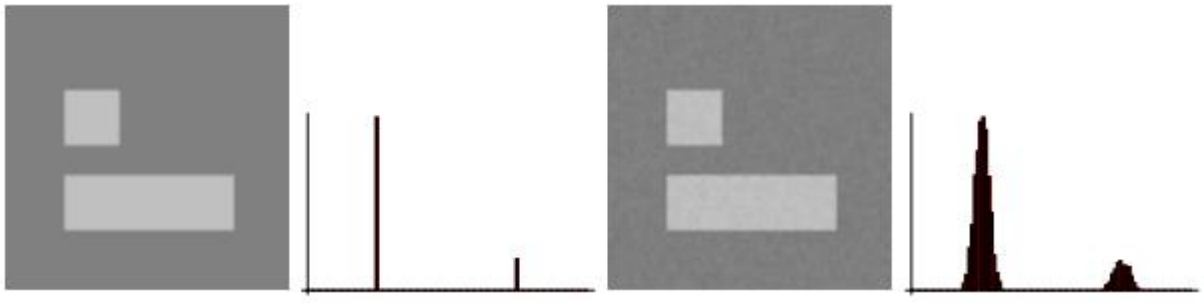
[2] Reprinted and referenced from "http://www.cs.uu.nl/docs/vakken/ibv/reader/chapter10.pdf

5.5. Threshold Selection

Many methods exist to find a suitable threshold for segmentation. The simplest method is the interactive selection of a threshold by the user, possibly with the aid of the image histogram - a method that is usually accompanied by a graphical tool which lets the user immediately assess the result of a certain choice of threshold. Automatic methods often make use of the image histogram to find a suitable threshold. Some of these methods are detailed below.

5.5.1. Histogram extrema

A typical objects-on-a-background image as shown in Figure 4 will have a bimodal histogram. The peaks of the histogram will not generally be as sharp as we would like them to be, because of the influence of image degrading factors such as noise, illumination artifacts and partial volume effects. In practice, the curves in the histogram corresponding to certain objects may overlap. When this happens, an errorless segmentation based on global thresholding is no longer possible.



(a) (b)

Figure 4. Bimodal histogram[3]

In the Figure 4 (a), an image containing only two grey values is shown. The histogram (next to it) shows only two peaks. In practice, image artifacts will cause the image and its histogram

[3] Reprinted and referenced from “http://www.cs.uu.nl/docs/vakken/ibv/reader/chapter10.pdf

to degrade from this ideal situation. In particular, the histogram blurs as shown in the image, Figure 4 (b) on the right. In practice, it is quite possible that the two curves in the histogram start to overlap.

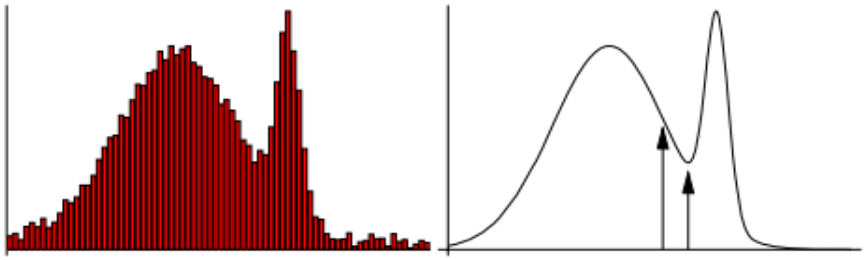
Regardless of whether or not there is overlap in the histogram, we can use the maxima (peaks) of the histogram to establish a segmentation threshold. This threshold t may be midway between two peaks with grey values p_1 and p_2 [25]:

$$t = \frac{p_1 + p_2}{2}, \tag{8}$$

or better, it may be the grey value at the minimum between the two peaks:

$$t = \arg \min_{v \in [p_1, p_2]} H(v) \tag{9}$$

where H(v) gives the histogram value at grey value v, and we assume that p_1 is smaller than p_2. Figure 5 shows why the minimum choice for t is usually better than the midway choice. The histogram is often not as smooth as we would like, as the figure shows. For many histogram analysis tasks (such as finding the global extrema) it is therefore useful to smooth the histogram beforehand. This can be done by convolution with a Gaussian or a discrete averaging kernel.



(a) (b)

Figure 5. Comparing a threshold midway between peaks and at the minimum between peaks[4]

[4] Reprinted and referenced from "http://www.cs.uu.nl/docs/vakken/ibv/reader/chapter10.pdf

Figure 5 (a) shows a bimodal histogram. The right image, Figure 5 (b) shows a smoothed version of this histogram, together with an arrow showing the threshold midway between the peaks (left arrow) and showing the threshold at the minimum (right arrow). The threshold at the minimum will misclassify fewer pixels.

5.5.2. Minimum variance within segments

If we assume that a segment should have relatively homogeneous grey values, it makes sense to select a threshold that minimizes the variance of the original grey values within segments. Alternatively, we may select a threshold that maximizes the variance between objects and background, or one that attempts to optimize both these "within" and "between" variances.

We will give an example using one threshold, i.e., two segments [25]. For simplicity, we normalize the histogram H(v) of the image to a proper distribution h(v), i.e., $\sum h(v)=1$. This can be achieved by setting $h(v) = H(v)/n$ for all v, where n is the total number of pixels in the image. The variance of the grey values σ^2 in the image then (by definition) equals

$$\sigma^2 = \sum_v (v-\mu)^2 h(v), \tag{10}$$

where $\mu = \sum v\, vh(v)$ is the mean grey value of the image. If we segment the image with threshold value t into a background segment 0 and an object segment 1, then the variance of grey values within each segment (respectively σ_0 and σ_1) are the mean grey values of the respective segments 0 and 1.

$$\sigma_0^2 = \sum_{v<t} (v-\mu_0)^2 h(v) \tag{11}$$

$$\sigma_1^2 = \sum_{v\geqslant t} (v-\mu_1)^2 h(v), \tag{12}$$

where,

$$\mu_0 = \frac{1}{h_0} \sum_{v<t} vh(v) \tag{13}$$

$$\mu_1 = \frac{1}{h_1} \sum_{v \geqslant t} v h(v) \tag{14}$$

The probabilities h_0 and h_1 that a randomly selected pixel belongs to segment 0 or 1 are

$$h_0 = \sum_{v<t} h(v) \tag{15}$$

$$h_1 = \sum_{v \geqslant t} h(v). \tag{16}$$

Note that $h_0 + h_1 = 1$. The total variance within segments σ^2_w is

$$\sigma_w^2 = h_0 \sigma_0^2 + h_1 \sigma_1^2. \tag{17}$$

This variance only depends on the threshold value t: $\sigma^2_w = \sigma^2_w(t)$. This means that we can find the value of t that minimizes the variance within segments by minimizing $\sigma^2_w(t)$. The variance between the segments 0 and 1, σ^2_b, is

$$\sigma_b^2 = h_0(\mu_0 - \mu)^2 + h_1(\mu_1 - \mu)^2. \tag{18}$$

Again, this variance is only dependent on the threshold value t. Finding the t that maximizes σ^2_b, maximizes the variance between segments. A hybrid approach that attempts to maximize σ^2_b while minimizing σ^2_w is to find the threshold t that maximizes the ratio σ^2_b / σ^2_w.

If more than two segments are required, the method described above can be extended to use multiple thresholds. The variances σ^2_w and σ^2_b will then be functions of more than one threshold, so we need multi-dimensional optimization to find the set of optimal thresholds. Since this is especially cumbersome if the number of segments is large, a more practical algorithm that minimizes the variances within segments is often used, an iterative algorithm known as K-means clustering.

5.5.3. K-means clustering

The objective of the K-means clustering algorithm is to divide an image into K segments (using K − 1 thresholds), minimizing the total within-segment variance [25]. The variable K must be set before running the algorithm. The within-segment variance σ^2_w is defined by

$$\sigma_w^2 = \sum_{i=0}^{K-1} h_i \sigma_i^2, \tag{19}$$

where $h_i = \sum_{v \in S_i} h(v)$ is the probability that a random pixel belongs to segment i (containing the grey values in the range S_i), $\sigma_i^2 = \sum_{v \in S_i} (v - \mu_i)^2 h(v)$ is the variance of grey values of segment i, and $\mu_i = \sum_{v \in S_i} v h(v)$ is the mean grey value in segment i. All definitions are as before in the case with a single threshold.

Algorithm 9. K-Means Clustering Algorithm for Image Thresholding

1. Initialization: Distribute the K − 1 thresholds over the histogram; e.g., in such a way that the grey value range is divided into K pieces of equal length. Segment the image according to the thresholds set. For each segment, compute the 'cluster center',i.e., the value midway between the two thresholds that make up the segment.
2. For each segment, compute the mean pixel value μi.
3. Reset the cluster centers to the computed values μi.
4. Reset the thresholds to be midway between the cluster centers, and segment the image.

5.5.4. Otsu Algorithm for Image Segmentation

Otsu's method executes thresholding that works on shape-based image by converting grey level image to binary image. This methods divided the pixels of image into two categories or histograms; e.g. for fore and back ground. After this division it assesses the optimal threshold for each category so that it can make their intra class variance to a minimum level. This extension of basic level to several levels of thresholding is referred to as 'Multi Otsu Method' [26].

The grey level histogram provides an efficient tool for developing thresholding algorithms In thresholding, binary images are created from grey-level binary images, by setting the value for pixels less than threshold to zero, and for pixels somewhat similar to and above threshold to one. If g(x, y) is a threshold version of f(x, y) at some global threshold T, it can be defined as [27]:

$$g(x, y) = 1 \text{ if } f(x, y) \geq T$$
$$= 0 \text{ otherwise} \tag{20}$$

Thresholding operation is defined as

$$T = M[x, y, p(x, y), f(x, y)] \tag{21}$$

where T represent threshold, f(x,y) represents the gray value of point(x, y), p(x, y) represents a local property of point such as the average gray value of the neighborhood centered on point (x,y). On the basis of links which above equation defines, threshold methods can be further classified into two types:

Global Thresholding: When T depends only on f(x, y) (in other words, only on gray-level values) and the value of T solely relates to the character of pixels, this thresholding technique is called global thresholding

Local Thresholding: If threshold T depends on f(x, y) and p(x, y), this thresholding is called local thresholding. This method divides an original image into several sub regions, and chooses various thresholds T for each sub region reasonably [7].

The Otsu method is a type of global thresholding in which it depend only gray value of the image. Otsu method was proposed by Scholar Nobuyuki Otsu in 1979. Otsu method is global thresholding selection method, which is widely used because it is simple and effective [4] which requires computing a gray level histogram before running.

The Otsu algorithm [29] is known as the maximum class square error method. Otsu's method is used to automatically perform histogram shape-based image thresholding, or the reduction of a gray level image to a binary image. The algorithm assumes that the image to be thresholded contains two classes of pixels or bi-modal histogram (e.g. foreground and background) then calculates the optimum threshold separating those two classes so that their combined spread (intra-class variance) is minimal. The extension of the original method to multi-level thresholding is referred to as the multi Otsu method. We assume that L is the total number of the gray levels of the image, f(m, n) represents the gray value of the point (m, n) in a M ×N image. Here assume the range of f(m, n) is bounded in [0, L−1]. Let g(k) be the sum of all pixels that gray values are k in the image, and p(k) represents the probability.

$$P(k) = \frac{g(k)}{M \times N} \tag{22}$$

Assume t is the threshold of a given image, the target and background can be expressed to {f (m, n) < t} and {t ≤ f(m, n) < l}, respectively. The probability the object of interested and the background are calculated by the following formula, respectively:

$$\omega_0(t) = \sum_{i=0}^{t-1} P_i, \tag{23}$$

$$\omega_1(t) = 1 - \omega_0(t). \tag{24}$$

Thus, the means of the two classes can be computed as

$$\mu_0(t) = \frac{\sum_{i=0}^{t-1} iP_i}{\omega_0(t)}, \tag{25}$$

$$\mu_1(t) = \frac{\sum_{i=t}^{L-1} iP_i}{\omega_1(t)}, \tag{26}$$

The total mean of pixels are defined as follows:

$$\mu = \omega_0(t)\mu_0(t) + \omega_1(t)\mu_1(t), \tag{27}$$

Otsu method is to obtain maximum between-clusters variance. Otsu's method of thresholding gray level images is efficient for separating an image into two classes where two types of fairly distinct classes exist in the image. The best threshold is the corresponding threshold when the fitness function value is maximum, the simpler formula for obtaining optimal threshold is as follows:

$$T^* = Arg \max \left[\omega_0(t)(\mu_0(t) - \mu)^2 + \omega_1(t)(\mu_1(t) - \mu)^2\right], \tag{28}$$

The larger the between-class variance between the two classes, the greater the difference and the higher the accuracy. So we can quickly extracted information from the gray level image.

Algorithm 10. Otsu Algorithm

1. Calculate the probability of the histogram and each pixel with different brightness
2. Initialize the probability and mean of every pixel
3. Let the range of threshold be in 1 and maximum gray value, then update the probability and mean of every pixel, calculate the total mean and the corresponding maximum between-class variance
4. Repeat Step 3 until maximizing the between-class variance, and the corresponding threshold is the maximum value

The extension of the original method to multi-level thresholding is according to the theory of the Otsu method [14, 15]. Let k be the number of the thresholds:

$$\begin{aligned}\omega(t_1, t_2, ..., t_k) &= n_0 n_1(\mu_0 - \mu_1)^2 + ... + n_0 n_k(\mu_0 - \mu_k)^2 + n_1 n_2(\mu_1 - \mu_2)^2 + ... \\ &\quad + n_1 n_k(\mu_1 - \mu_k)^2 + ... + n_{k-1} n_k(\mu_{k-1} - \mu_k)^2 \end{aligned} \quad (29)$$

Where the number of pixels at the i^{th} object block be n_i , i = 1, 2, ..., k respectively. μi is the corresponding mean. When the between-cluster variance is maximum, the corresponding threshold t_i^* is the optimum value

$$(t_1^*, t_2^*, ..., t_k^*) = Arg \max_{0 \le t_1 \le t_2 \le ... \le t_k \le l-1} (\omega(t_1, t_2, ...t_k)) \quad (30)$$

The Otsu method was one of the better threshold selection methods for general real world images with regard to uniformity and shape measures. However, because of the one-dimensional nature of this algorithm which only considers the gray-level information, it does not give better segmentation result sometimes. So, for that two dimensional Otsu algorithms was proposed which works on both gray-level threshold of each pixel as well as its Spatial correlation information within the neighborhood. As the number in classes of an image increases, Otsu's method takes too

much time to be practical for multilevel threshold selection [10]. Many techniques thus were proposed to reduce time spent on computation and still maintain reasonable thresholding results. In [9], proposed a fast recursive technique that can efficiently reduce computational time. And, now in this paper we are going to see the results of GSO algorithm fused with Otsu as proposed in [6].

Following are some of the papers (Table 1 in the next page) that study and propose different variations of the Otsu algorithm. Also, in [12] it is concluded that in order to improve the performance of Otsu algorithm, it has to be combined with other algorithms like K-means, improved histogram, etc.

Table 1. Otsu with other algorithms

Paper	**Method**	**Segmentation Result**	**Limitation**
Otsu Thresholding based on Improved Histogram [30]	Otsu + Improved Histogram	Good	Does not give better result with Gaussian noise
Comparative Research on Image Segmentation Algorithm [31]	Otsu + Global Thresholding	Good/Stable	High complexity / low processing rate
Otsu and K-Means Method [32]	Otsu + K-Means	Good	Increased complexity and time
Image Segmentation based on Improved Otsu Algorithm [33]	Otsu + Entropy based	Stable	When the global distribution of target image and background vary widely, the performance degrades.

5.5.5. Multi-level threshold image segmentation using GSO and Otsu algorithm

In GSO, the dimension of the glowworm individual is equal to the number of the thresholds. Then, randomly initialize of the glowworm individual position between 0 and 255 and calculate the fitness value according to Equation (30). The best threshold can be obtained through many iterations. For detailed information on the GSO algorithm, you may refer to Chapter 4.

Algorithm 11. GSO + Otsu Algorithm [6]

1. Initialization: Randomly generate initial population, and set the parameters l_0, r_0, r_s, S, ρ, γ, $x_i(0)$, the maximum number of iterations and the number of the thresholds
2. For each glowworm:
    a. Let luciferin value be equal to the objective function value
    b. Choose its neighbor within decision-making domain
    c. Determine the direction of movement
    d. Move and update the location
    e. Update the fitness value of each glowworm
    f. Update the dynamic decision-making domain radius
3. Reach the maximum number of iterations, turn to Step 4; Otherwise, turn to Step 2;
4. Display the optimal solution.

6. RESULTS

The experimental results in the reference paper [6] are from a desktop with a minimum configuration for XP running on 1 GB RAM with a 1.73 GHz Intel T5300 processor. But, the current results presented in the paper are taken from the system that has 2.4 GHz Intel core i5 dual core processor running the latest Windows OS (as of date) ie, Windows 10 on a 6 GB RAM. So, we can definitely expect a huge performance difference between the numbers presented here and the numbers in [6].

Here are some of the test runs carried out on the current GSO+Otsu algorithm implemented using MATLAB R2015b

Input 1:

Table 2. Test run input – Lena

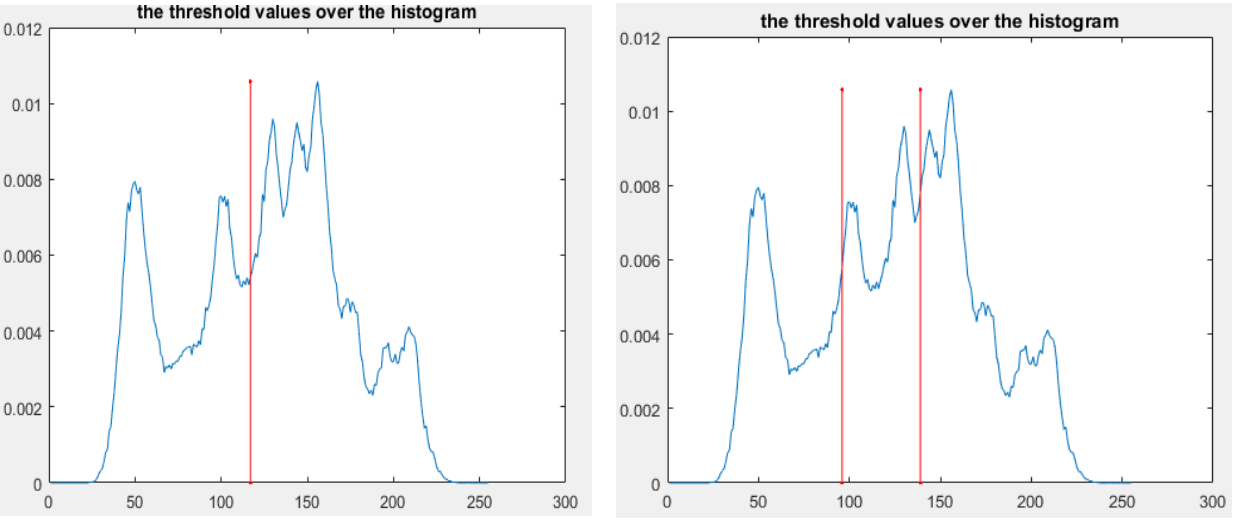
Image	Size	Color Scale
Lena	512 * 512	Gray

Figure 6. Input image – Lena

Output:

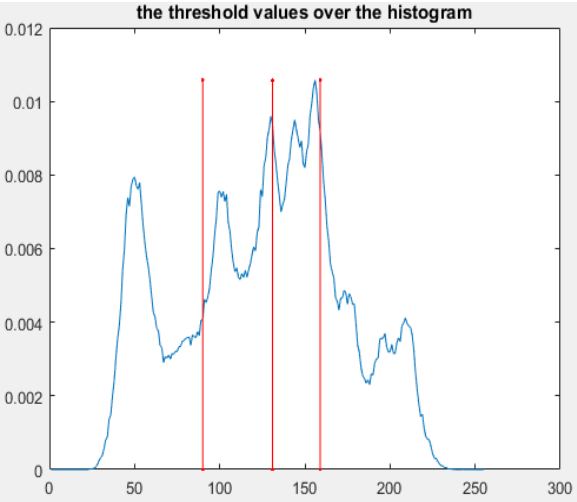
(a) Single threshold (b) Double thresholds (c) Three thresholds

Figure 7. Output for Lena gray image



(a) Single Threshold (b) Double Thresholds

Figure 8. Output for Lena gray image - Single and Double Thresholds



(c) Three thresholds

Figure 9. Histogram output for Lena gray image – Three thresholds

Table 3. Output for Lena image

Number of thresholds	Calculated threshold values	Elapsed Time (seconds)
1	117	0.35
2	96, 139	0.61
3	90, 131, 159	0.65

From the output pictures, we can clearly identify that quality of segmented image increases with the number of thresholds. The gray levels in the images are finely segmented into clearer regions with the increase in threshold levels. Though we observe that there is approximately 2 times increase in execution time from single threshold (0.35 seconds) to double thresholds (0.61 seconds), it is not much of a difference from two to three thresholds (0.65 seconds). So, we observe that the execution time is not in a linear relationship with the number of thresholds.

Input 2:

Table 4. Test run input – Camera man

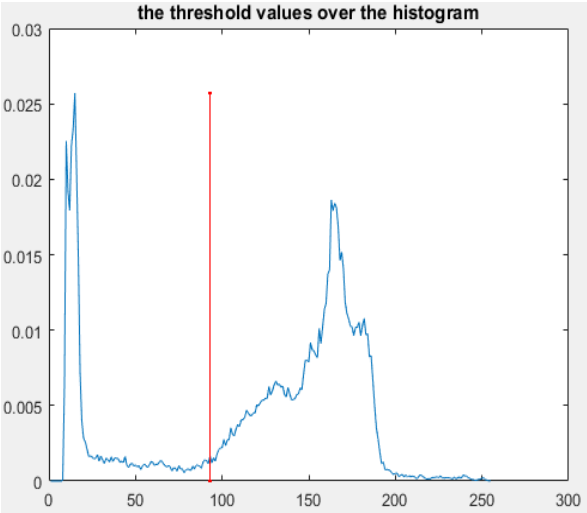
Image	Size	Color Scale
Cameraman	256 * 256	Gray

Figure 10. Input image – Camera man

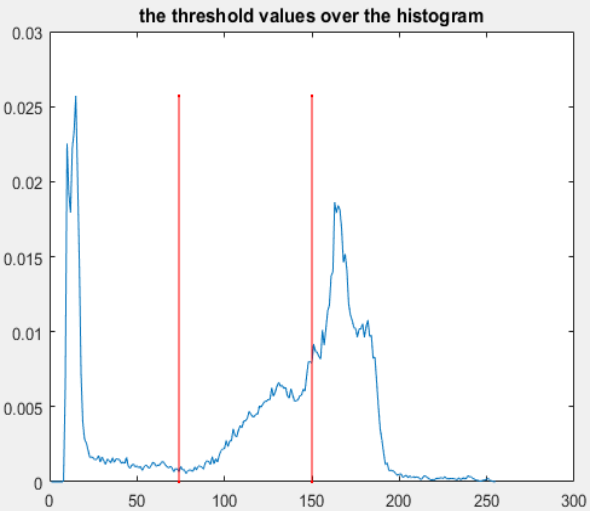
Output:

(a) Single threshold (b) Double thresholds (c) Three thresholds

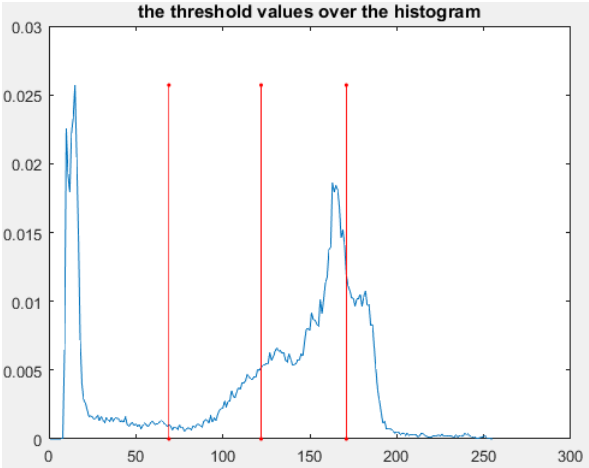
Figure 11. Output for cameraman gray image



(a) Single Threshold



(b) Double Thresholds



(c) Three thresholds

Figure 12: Histogram output for Camera man gray image

Table 5. Output for Cemera man image

Number of thresholds	Calculated threshold values	Elapsed Time
1	93	0.32
2	74, 150	0.60
3	61, 124, 178	0.61

Here again, we see that region segmentation improved with the increase in thresholds. Observe the buildings behind the cameraman that became clearer at higher number of threshold levels. Execution time again increased at a linear pace from one to three thresholds, but then almost flat from two to three thresholds. However, an interesting observation is that the resolution of this input image is half the resolution of first image – Lena, but still there is not much difference observed in the execution times.

Input 3:

Table 6. Test run input – Lena_rgb

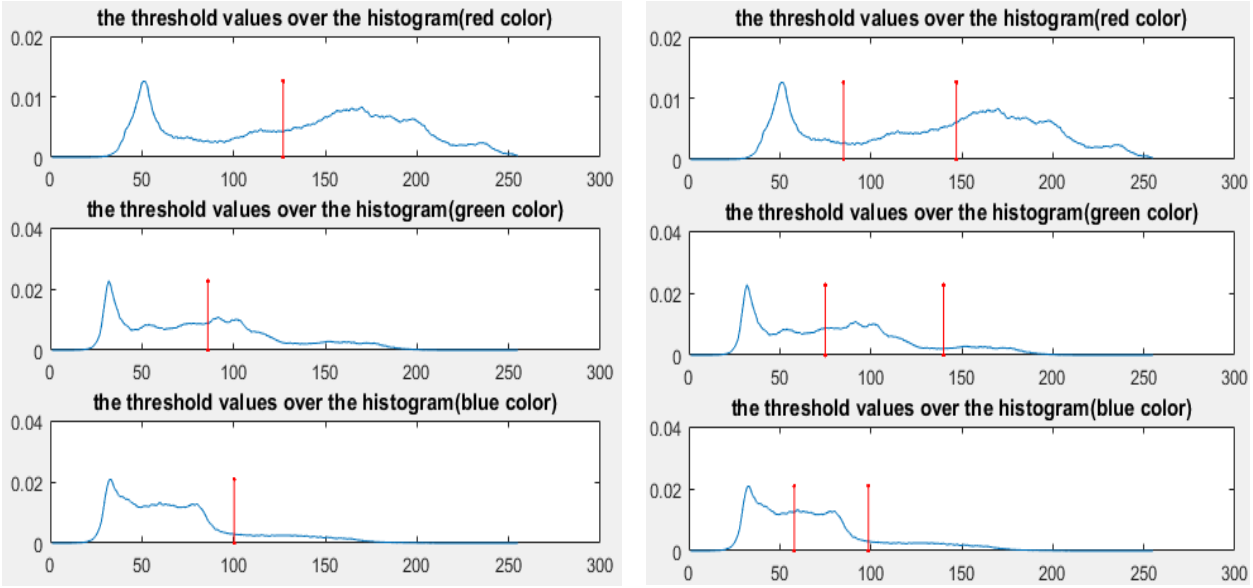
Image	Size	Color Scale
Lena_rgb	512 * 512	Color

Figure 13. Input image – Lena_rgb

Output:

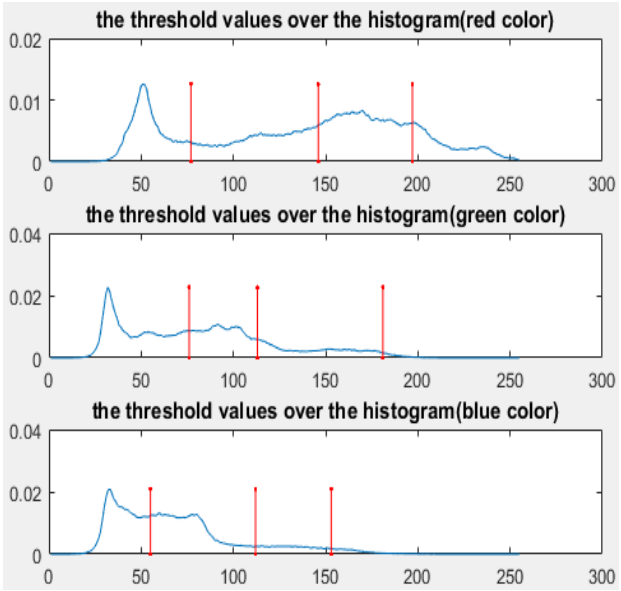
(a) Single threshold (b) Double thresholds (c) Three thresholds

Figure 14. Output Lena_rgb color image



(a) Single Threshold (b) Double Thresholds

Figure 15. Histogram outputs for Lena_rgb color image – Single and Double thresholds



(c) Three thresholds

Figure 16. Histogram outputs for Lena_rgb color image – Three thresholds

Table 7. Output for Lena_rgb image

Number of thresholds	Calculated threshold values	Elapsed Time
1	Red: 127 Green: 86 Blue: 100	0.91
2	Red: 85, 147 Green: 75, 140 Blue: 58, 98	1.69
3	Red: 77, 146, 197 Green: 76, 113, 181 Blue: 55, 112, 153	1.82

The input we used here is same as the first input (Lena) except that this is a color image this time. We also maintained same resolution so that it remains comparable. A single threshold for a color image mean a single level of each color – R (Red), G (Green) and B (Blue). You can

observe the figure 18 (a) where we calculated a threshold level for each of the color level that is sampled from the image. And, for the very obvious reasons of color level separations in the image, one can except to have a longer processing times than its counter part gray image.

There is approximately 3 times increase in the execution time of this color image when compared to the gray image for the respective thresholds. And, here again we see that increase in threshold levels do not linearly increase the execution times.

Input 5:

Table 8. Test run input – Peppers_rgb

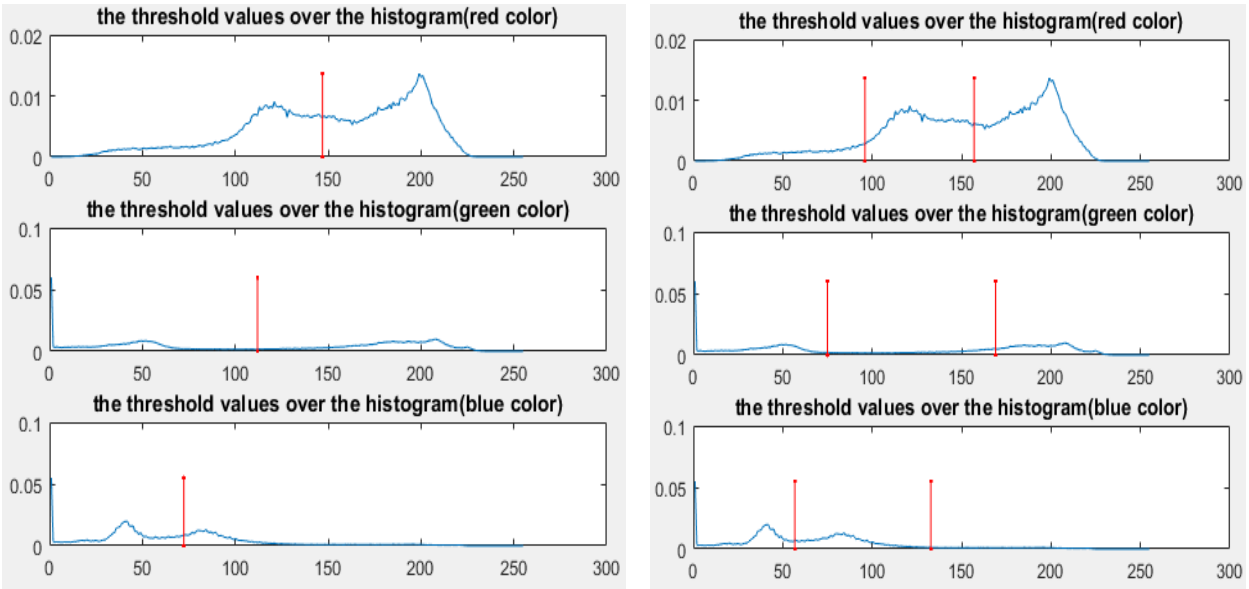
Image	Size	Color Scale
Peppers_rgb	512 * 512	Color

Figure 17. Input image – Peppers_rgb

Output:

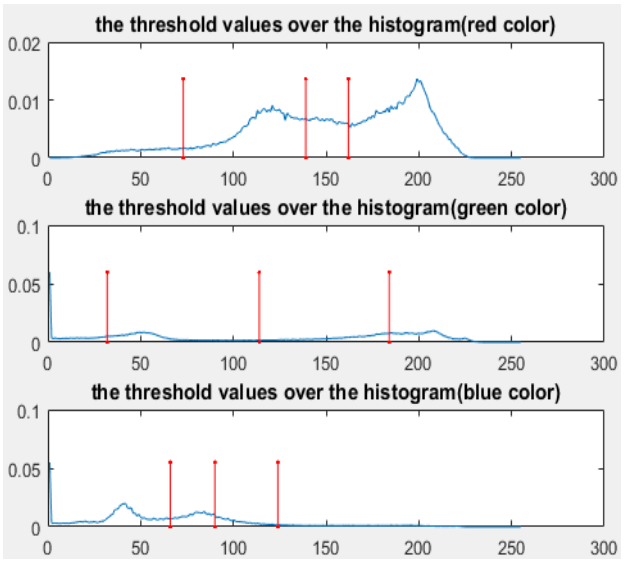
(a) Single threshold (b) Double thresholds (c) Three thresholds

Figure 18. Output Peppers_rgb color image



(a) Single Threshold (b) Double Thresholds

Figure 19. Histogram outputs for Peppers_rgb color image – Single and Double thresholds



(c) Three thresholds

Figure 20. Histogram outputs for Peppers_rgb color image – Three thresholds

Table 9. Output for Peppers_rgb image

Number of thresholds	Calculated threshold values	Elapsed Time
1	Red: 147 Green: 112 Blue: 72.3	0.91
2	Red: 96, 157 Green: 75, 169 Blue: 56.7, 132.82	1.68
3	Red: 73, 139, 162 Green: 32, 114, 184 Blue: 66, 90, 124	1.77

Execution times differed very less between two and three thresholds of image segmentation when compared between one and two thresholds of segmentation. The colors R (Red), G (Green) and B (Blue) had better color region separations with the increase in thresholds. Comparing the Figure 21(a) and Figure 21(c) gives the clear distinction of color contours in both. Figure 21(a)

had some indistinguishable or blacked out areas which became clearer by using higher number of thresholds for the image segmentation process.

Table 10. Results Comparison

Algorithm	**Number of Thresholds**	**From [6]**		**From Current results**	
		Lena	**Camera man**	**Lena**	**Camera man**
Otsu	Single threshold	1.78	0.45	NA [#]	NA[#]
GSO + Otsu		4.45	1.39	0.35	0.32
Otsu	Double thresholds	45.41	15.37	NA[#]	NA[#]
GSO + Otsu		7.15	2.03	0.61	0.60
Otsu	Three thresholds	2583.03	953.98	NA[#]	NA[#]
GSO + Otsu		9.21	2.5	0.65	0.61

[#] NA – Not Available

We cannot really comment on a comparison basis between Otsu and GSO + Otsu on the current setup as we did not have Otsu standalone algorithm for image segmentation. But going by the credibility of the figures in the reference paper, we can very well assume to have a huge improvement in performance by fusing GSO and Otsu for the image segmentation. And, a better quality with multi-threshold image segmentation.

Also, the other objective of this paper is to evaluate and assess the claim of GSO+Otsu algorithm numbers against the standalone Otsu algorithm and it is very evident from the results here that GSO+Otsu algorithm performs far better on higher number of thresholds in the image segmentation.

The important thing to note here was that Otsu algorithm was proposed on the gray scale images or gray level histogram, but now is extended to the color images in this paper in conjunction with GSO.

7. CONCLUSION

One of main problems with native Otsu algorithm is the performance and there have been many algorithms that were used in conjunction with Otsu to make it better. And, one such algorithm in the same direction is GSO+Otsu, which clearly outperformed the native standalone algorithm by a large difference. Native Otsu method required computing a gray level histogram before running and because of this one dimensional nature, the segmentation results were not really great. Now, with GSO+Otsu we get a two dimensional proposal where it is dependent on both gray levels in the image and also spatial correlation information in the neighborhood.

Native Otsu method performs exhaustive search to evaluate the criterion for maximizing the between-class variance. As the number in classes of an image increases, Otsu's method takes too much time to be practical for multilevel threshold selection. And, the fusion of Otsu with GSO now proves otherwise.

The experimental results show that the algorithm not only has a good effect on image segmentation with clear image contours, it also reduces the calculation time, and improves the quality of image segmentation.

The credence and the significance of the numbers presented in this paper would assume lot of importance if we could make a comparative analysis with standalone Otsu algorithm under the current configuration of the system. Also, the numbers on the color images could not compared with any benchmark numbers as of now. So, a comparative study of the color image segmentation by RGB values can provide good insights into the algorithm's performance. Also, with the capability of MATLAB to process by parallel computation leaves a huge scope of further improvement in the performance, which if exploited can give even more interesting numbers.

8. BIBLIOGRAPHY

[1] E. Bonabeau, M. Dorigo, and G. Theraulaz (1999), "Swarm Intelligence: From Natural to Artificial System". Oxford University Press, New York.

[2] K.N. Krishnanand and D. Ghose (2005), "Detection of multiple source locations using a glowworm metaphor with applications to collective robotics", IEEE Swarm Intelligence Symposium, Pasadena, California, USA, 84- 91.

[3] K.N. Krishnanand and D. Ghose (2009), "Glowworm swarm optimization for simultaneous capture of multiple local optima of multimodal functions. Swarm Intelligence", 87- 124.

[4] Linda Shapiro and George Stockman (2001), "Computer Vision", Prentice-Hall, 279-325.

[5] H. J. Vala (2013), "A Review on Otsu Image Segmentation Algorithm", International Journal of Advanced Research in Computer Engineering & Technology, 2(2), 387-389.

[6] Qifang Luo, Zhe Ouyang, Xin Chen, Yongquan Zhou (2011), "A Multilevel Threshold Image Segmentation Algorithm Based on Glowworm Swarm Optimization", Journal of Computational Information Systems 10:1621–1628.

[7] Nirpjeet kaur and Rajpreeet kaur (2011), "A review on various method of image thresholding", IJCSE.

[8] L.J. Fogel, A.J. Owens, M.J. Walsh, (1996), "Artificial Intelligence through Simulated Evolution", John Wiley.

[9] A.E. Eiben, J.E. Smith (2003), "Introduction to Evolutionary Computing".

[10] G.S. Hornby and J.B. Pollack (2002), "Creating high-level components with a generative representation for body-brain evolution. Artificial Life", 8(3):223–246.

[11] M. Dorigo and T. Stutzle (2003), "The ant colony optimization metaheuristic: algorithms, applications, and advances," in Handbook of Metaheuristics, Eds. F. Glover and G. Kochenberger, New York: Springer, 250-285.

[12] A. Colorni, M. Dorigo et V. Maniezzo (2003), "Distributed Optimization by Ant Colonies, actes de la première conférence européenne sur la vie artificielle", Paris, France, Elsevier Publishing, 134-142.

[13] S.M. Thampi (2009), "Swarm intelligence", arXiv preprint arXiv:0910.4116.

[14] D. Dervis Karaboga (2005), "An Idea Based On Honey Bee Swarm for Numerical Optimization", Technical Report-TR06, Erciyes University.

[15] Tereshko V. Loengarov (2005), A, “Collective decision-making in honey bee foraging dynamics, Computing and Information Systems”, University of the West of Scotland, UK. 9 (3): 1-7.

[16] D. Karaboga (2005), “An idea based on honey bee swarm for numerical optimization”, Technical Report TR06, Erciyes University, Engineering Faculty.

[17] Mitchell, Melanie (1996) “An Introduction to Genetic Algorithms”, Cambridge, MA: MIT Press.

[18] H.J. Bremermann (1962), “Optimization through evolution and recombination”, Self-Organizing systems, 93, 106.

[19] L. Spector et al., (1999), “Finding a Better-than-Classical Quantum AND/OR Algorithm using Genetic Programming,” Proc. 1999 Congress Evolutionary Computation, IEEE Press, Piscataway, N.J, 2239-2246.

[20] R. Storn, K. Price (1997), “Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces”, Journal of Global Optimization, 11: 341–359.

[21] C. Sun, H. Zhou and L. Chen (2012), “Improved differential evolution algorithms,” Computer Science and Automation Engineering (CSAE), 2012 IEEE International Conference on, Zhangjiajie, 142-145.

[22] Kennedy. J. Eberhart, R (1995), “Particle Swarm Optimization”, Proceedings of IEEE International Conference on Neural Networks, 1942–1948.

[23] I.C. Trelea, (1995), “The Particle Swarm Optimization Algorithm: convergence analysis and parameter selection”, Information Processing Letters, 85 (6): 317–325.

[24] Nick Efford (2000), “Digital Image Processing: A Practical Introduction Using JavaTM”, Pearson Education.

[25] A.Marion (1991), “An Introduction to Image Processing”, Chapman and Hall.

[26] P.S. Liao, P.C. Chung (2001), "A fast algorithm for multilevel thresholding", Journal of Information Science and Engineering, September, 17 (5): 713-727.

[27] R.C. Gonzalez, R.E. Woods, B.R. Masters (2009). Digital Image Processing, Third Edition. J. Biomed. Opt. Journal of Biomedical Optics, 14(2), 029901.

[28] Nobuyuki Otsu (1979), "A threshold selection method from gray-level histograms". IEEE Trans. Sys., Man., Cyber, 9 (1): 62–66.

[29] Z. Ningbo, W. Gang, Y. Gaobo, D. Weiming (2009), "A fast 2D otsu thresholding algorithm based on improved histogram," 2009 Chinese Conference on Pattern Recognition.

[30] W. X. Kang, Q. Q. Yang, R. R. Liang (2009), "The Comparative Research on Image Segmentation Algorithms", IEEE Conference on ETCS, 703-707.

[31] L. Dongju and Y. Jian (2009), "Otsu method and k-means in Hybrid Intelligent Systems", 2009 Ninth International Conference on Hybrid Intelligent Systems, 1(1): 344–349.

[32] Zhong Qu and Li Hang (2010), "Research on Iimage Segmentation Based on the Improved Otsu Algorithm.", 2010 Second International Conference on Human-Machine Systems and Cybernetics.

[33] J.H Holland (1975), "Adaptation in Natural and Artificial Systems", The University of Michigan Press, Ann Arbor.

[34] Ingo Rechenberg (1971), "Evolutionsstrategie – Optimierung technischer Systeme nach Prinzipien der biologischen Evolution (PhD thesis)".