<table>
<thead>
<tr>
<th><strong>Title</strong></th>
<th>Multi-threading based implementation of ant-colony optimization algorithm for image edge detection</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Author(s)</strong></td>
<td>Aslam, Asra; Khan, Ekram; Beg, M.M. Sufyan</td>
</tr>
<tr>
<td><strong>Publication Date</strong></td>
<td>2015-10-17</td>
</tr>
<tr>
<td><strong>Publisher</strong></td>
<td>IEEE</td>
</tr>
<tr>
<td><strong>Link to publisher's version</strong></td>
<td><a href="http://dx.doi.org/10.1109/INDICON.2015.7443603">http://dx.doi.org/10.1109/INDICON.2015.7443603</a></td>
</tr>
<tr>
<td><strong>Item record</strong></td>
<td><a href="http://hdl.handle.net/10379/6895">http://hdl.handle.net/10379/6895</a></td>
</tr>
<tr>
<td><strong>DOI</strong></td>
<td><a href="http://dx.doi.org/10.1109/INDICON.2015.7443603">http://dx.doi.org/10.1109/INDICON.2015.7443603</a></td>
</tr>
</tbody>
</table>

Downloaded 2018-12-11T00:40:28Z

Some rights reserved. For more information, please see the item record link above.
Multi-Threading based Implementation of Ant-Colony Optimization Algorithm for Image Edge Detection

Asra Aslam  Department of Computer Engineering  Aligarh Muslim University  Aligarh-202002, India  Email: asra.aslam.7@gmail.com
Ekram Khan  Department of Electronics Engineering  Aligarh Muslim University  Aligarh-202002, India  Email: ekhan67@gmail.com
M.M. Sufyan Beg  Department of Computer Engineering  Aligarh Muslim University  Aligarh-202002, India  Email: mmsbeg@hotmail.com

Abstract—Ant Colony Optimization (ACO) is a nature inspired algorithm for solving optimization problems and is proved to be a powerful tool in image processing. It works on the principle that an ant while moving leaves pheromones on its path, which is used as guide to be followed by other ants. ACO is complex and time consuming. In this paper, a multi-threading based implementation of ACO is proposed for identifying edges in images. It combines multi-threading with ACO for increasing the randomness among the artificial ants. The algorithm is implemented and its performance is measured in terms of time complexity. Simulation results show that the proposed method has significantly lower execution time as compared to conventional ACO for edge detection.

Index Terms—Multi-Threading, Ant Colony Optimization, Edge Detection.

I. INTRODUCTION

ACO is a nature inspired algorithm based on the observations of real ant colonies. It is motivated by the natural phenomenon that ants deposit pheromone on the ground, in order to mark some favorable path that should be followed by the other members of the colony [1].

The edge detection (ED) in images refers to the process of extracting edges in a digital image. It is the approach which is used most frequently for segmenting images based on abrupt(local) changes in intensity. It generally detects the contour of an image and thus provides important details about an object. This process has its application in image analysis, computer vision applications and machine vision.

Most of the edge detection algorithms depend on threshold. A larger value of threshold results in absence of true edges, whereas a small value leads to presence of many false edges. So, an optimal value of threshold is required for detecting edges. Hence, edge detection is an optimization problem.

The ACO meta-heuristic can be applied to any combinatorial optimization problem for which a constructive heuristic can be defined. Hence ACO can be used for detecting edges [2]-[8]. An ACO based approach has the potential of overcoming some of the limitations of conventional ED methods, like reducing computational time by introducing parallelism, finding more true edges, absence of false edges, etc.

ACO based approach is successful in reducing computational time by introducing parallelism. But from implementation point of view, all artificial ants are not independent, they are dependent on each other as the procedure is iterative in nature. So, these iterative procedures may be replaced with parallel ones. This may be possible through Multi-Threading.

In this paper, a novel edge detection method employing the concept of multi-threading along with ACO is proposed. Multi-threading is the ability of an operating system to concurrently run programs that have been divided into sub-components, or threads. Here ants are implemented using threads, so that they can run in parallel with any number of steps any time. Results are compared with conventional method to demonstrate effectiveness of the proposed method not only in detecting edges but also in terms of computational efficiency.

The remainder of the paper is organized as follows. A brief overview of existing edge detection techniques with and without ACO are given in Section-II. Details of the proposed multi-threading based ant colony optimization are presented in Section-III. Simulation results and discussion are given in Section-IV. Lastly, concluding remarks and avenues for future work appear in Section-V.

II. BACKGROUND

A. Review of Edge Detection method

Edge detection is the approach most widely used for detecting edges and is based on detecting abrupt local changes in the intensity of image. Edge pixels are those pixels at which the intensity of an image function changes abruptly. Various techniques are reported in the literature for edge detection. Some of them are gradient based techniques which includes Sobel, Prewitt and Roberts operators [9]. Other techniques are laplacian based techniques introduced by Marr-Hildreth [10]. This suggest that an edge detection operator should have two features: firstly it should be a differential operator and secondly it should be capable of being "tuned" to act at any desired scale. The most satisfactory operator for these conditions is $\nabla^2 G$ where $\nabla^2$ is the Laplacian operator and G is the 2-D Gaussian function.
Sobel cross-gradient operators, shown in Fig. 1 is one of the most earliest attempts to compute gradient measurement on an image. It is of $3 \times 3$ masks that are symmetric about the center point. The difference between the third and first row approximates the derivative in the x-direction, and the difference between third and first columns approximate the derivative in the y-direction by using a weight of 2 in the center coefficient of the two equations. The gradient in x and y direction are calculated using (1) and (2)

$$g_x = \frac{\delta f}{\delta x} = (z_7 + 2z_8 + z_9) - (z_1 + 2z_2 + z_3)$$  \hspace{1cm} (1)

$$g_y = \frac{\delta f}{\delta y} = (z_3 + 2z_6 + z_9) - (z_1 + 2z_4 + z_7)$$  \hspace{1cm} (2)

then

$$|g(x, y)| = \sqrt{g_x^2 + g_y^2}$$  \hspace{1cm} (3)

a point is considered as edge point if $|g(x, y)|$ is greater than equal to a predefined threshold.

![Sobel mask](image)

![3x3 region](image)

Fig. 1: $3 \times 3$ mask of Sobel Operator and part of image

Most of the edge-detection methods are based on threshold. Once the final threshold($Th$) is obtained (either by basic global thresholding or by adaptive thresholding method), each pixel of gradient image $g(x,y)$ is compared with $Th$. The pixels with gradient higher then $Th$ are considered as edge point and is represented as a white pixel; otherwise it is designated as black. The edge-mapped image $E(x,y)$, thus obtained is:

$$E(x,y) = \begin{cases} 255 & g(x,y) \geq Th, \\ 0 & \text{otherwise.} \end{cases}$$  \hspace{1cm} (4)

That is, a pixel at $(x,y)$ having $g(x,y)$ less than $Th$ is called a background point; otherwise, it is an edge point.

Another successful attempt in the direction of image edge detection is the Canny edge detector [11]. It has three salient characteristics: low error rate, edge points should be well localized, and single edge point response. The canny edge detector first smoothes the image to eliminate noise, and then it finds the image gradient. Also it requires thinning thereafter. Canny’s algorithm also need threshold for detecting edges. Optimal value of threshold plays an important role in finding true edges. Thus edge detection can be categorized as optimization problem which can be handled using ant colony optimization.

### B. Ant Colony Optimization

Ant Colony Optimization (ACO) [1], [12]–[14] is an algorithm which is inspired by the observation of real ant colonies. It is motivated by the natural phenomenon that ants deposit pheromone on the ground in order to mark some favorable path that should followed by the other members of the colony. It can be applicable for solving any optimization problem in which constructive heuristic can be defined. Thus ACO can be used for detecting edges.

As ant colony optimization is an emerging field nowadays but there has been little research in the direction of use of ant colony optimization in image processing. The edge detection using ant algorithms is introduced in [15] where authors introduce the concept of using the pheromone matrix for detection of edges. They perform local and global updates on matrices according to the steps of different ants. The edge detection procedure is further improved in [5]. It is efficient on compensation of broken edges. But it is a post-processing method for edge detection by using conventional edge detection algorithms. Ant Colony Optimization for image edge detection is further explored in [16]. It is an extension from Ant System to Ant Colony System. Authors applied it in tour construction process by varying parameter $q_0$ for controlling the degree of exploration of the ants. Similar efforts are observed in [17] with slight variation in the iteration of algorithm. Similar algorithm is used in this paper for further improvement of that algorithm by multi-threading.

The Ant Colony Optimization algorithm consists of the following main steps:

- **Initialization**

  Distribute $K$ ants randomly on an image i.e. assign random positions to all $K$ ants as there starting position. Form the pheromone matrix and assign $\tau_{init}$ as the initial value of each element in the pheromone matrix. Determine heuristic information $\eta_{i,j}$ at each pixel $(i,j)$ of an image. This heuristic information remain constant through out the algorithm therefore this matrix can be constructed initially. $\eta_{i,j}$ can be determined as:

$$\eta_{i,j} = \frac{V_c(I_{i,j})}{V_{max}}$$  \hspace{1cm} (5)

where $I_{i,j}$ is the intensity value of the pixel at the position $(i,j)$ of the image. $V_c(I_{i,j})$ depends on the variation of the intensity values of the $8$-connectivity neighborhood(see Fig.2), and is given by:

$$V_c(I_{i,j}) = |I_{i+1,j+1} - I_{i-1,j-1}| + |I_{i+1,j-1} - I_{i-1,j+1}| + |I_{i+1,j+1} - I_{i-1,j-1}|$$  \hspace{1cm} (6)

where, $V_{max}$ is the maximum intensity variation in the whole image and it serves as a normalization factor.

- **Construction**

  On every iteration, all ant moves $L$ construction steps across the image, from one pixel to another. A construction step includes single step of ant. An ant moves from one node $(i_0,j_0)$ to its neighborhood node $(i,j)$ according to a transition probability:

$$p^{(n)}_{(i_0,j_0),(i,j)} = \frac{(\tau_{ij}^{(n-1)})^a(\eta_{ij})^\beta}{\sum_{(l,k)\in\omega(x_0,y_0)}(\tau_{lj}^{(n-1)})^a(\eta_{lj})^\beta}$$  \hspace{1cm} (7)
and number of iterations, \( L \) is the number of construction steps only ACO is given in below steps. Let \( \omega_{(i_0,j_0)} \) be the neighborhood node \((i_0,j_0)\) which is of 8-connectivity neighborhood; \( \eta_{i,j} \) is the heuristic information. The constants \( \alpha \) and \( \beta \) represent the influence of the pheromone matrix and the heuristic matrix, respectively.

### Update

The Ant Colony Optimization performs two kind of updates operations: Local update and Global update, for updating the pheromone matrix. The local update is performed each time, when an ant visited a node. The amount of pheromone on the node \((i,j)\) on the \( n^{th} \) iteration, \( \tau_{i,j}^{(n)} \), is updated using the equation:

\[
\tau_{i,j}^{(n)} = (1 - \psi)\tau_{i,j}^{(n-1)} + \psi \tau_{i,j}^{\text{init}}
\]

where \( \psi \epsilon [0,1] \) is the pheromone decay coefficient and \( \tau_{i,j}^{\text{init}} \) is the initial pheromone value. The Global update is carried out after all the ants finish the all the given number of steps. The pheromone matrix is updated according to:

\[
\tau_{i,j}^{n} = (1 - \rho)\tau_{i,j}^{(n-1)} + \rho \sum_{k=1}^{\kappa} \Delta \tau_{i,j}^{(k)}
\]

where, \( \rho \) is evaporation rate and \( \Delta \tau_{i,j}^{(k)} \) is the deposited amount of pheromone, it is equal to the average of heuristic information associated with the pixels that belong to the tour of the \( k^{th} \) ant if the pixel \((i,j)\) was visited by the \( k^{th} \) ant in its current tour; 0 otherwise.

### Decision

Finally, the pheromone matrix is used to classify each pixel either as an edge or a non-edge by applying a threshold. The threshold value is computed based on the method given in [18]. If the pheromone value of a pixel is greater than threshold then consider that pixel to be the edge pixel otherwise discard it as the edge pixel.

C. ACO based Edge Detection

The implementation algorithm [17] for edge detection using only ACO is given in below steps. Let \( I(i,j) \) be the input image and \( g(i,j) \) be the output edge image. \( N \) denotes the number of iterations, \( L \) is the number of construction steps and \( \kappa \) is the number of ants.

1. for \( k=1 \) to \( \kappa \)
2. \{ Generate \( i,j \) randomly
3. Assign \((i,j)\) to starting position of ant \( k \)
4. \}
5. for \( i=1 \) to \( h \)
6. for \( j=1 \) to \( w \)
7. compute \( \eta_{i,j} \) using eq. (5) and (6)
8. for \( i=1 \) to \( h \)
9. for \( j=1 \) to \( w \)
10. \( \tau_{i,j}^{(n)} = \tau_{i,j}^{\text{init}} \)
11. for \( n=1 \) to \( N \)
12. \{ for \( l=1 \) to \( L \)
13. for \( k=1 \) to \( \kappa \)
14. \{
15. for \( m=1 \) to \( 8 \) 8-neighborhood pixels of ant(k)
16. \{ calculate probability \( P_{l,n}^{(n)} \) using eq.(7) \}
17. \}
18. find \( m' \) such that \( P_{l,n}^{(n)} = \max \{ P_{l,n}^{(n)},P_{l,n+1}^{(n)},P_{l,n+2}^{(n)},...\} \)
19. if( pixel \((m',n')\) is path\(h_k \)
20. goto step 18 to find next maximum probability
21. else
22. \{ update pixel’s pheromone \( \tau_{i,j} \) (locally) using eq.8 \}
23. goto step 25
24. \}
25. \}
26. Update all visited pixels’ pheromone \( \tau_{i,j} \) (globally) using eq (9)
27. \}
28. Compute threshold(Th) from pheromone \( \tau \) matrix
29. for \( i=1 \) to \( h \)
30. for \( j=1 \) to \( w \)
31. if(\( \tau_{i,j} > Th \) \) \( g(x,y) = 1 \)
32. else \( g(x,y) = 0 \)

Summary of implementation of ACO is shown in Fig. 3, from where it can be seen that movement of a particular ant is dependent on the other ants as the process is a typically iterative one. In case of the algorithm proposed in this paper, this limitation is overcome as explained in a later section.

III. Multi-Threading based Implementation of ACO

Multi-threading is the ability of an operating system to concurrently run programs that have been divided into subcomponents, or threads. Threads are actually tiny processes. A thread is a basic unit of CPU utilization. If a process has multiple threads of control, it can perform more than one task at a time. It offers better utilization of processors and other system resources. They provide the maximum degree of parallelism. The problem arises in multi-threading when more than one thread try to access the same critical section to proceed their tasks. So there is a requirement of synchronization mechanism for implementation of multi-threading. Semaphore ‘S’ is a popular synchronization tool which can be used for solving critical section problems. Semaphore ‘S’, is an integer variable that
Fig. 3: ACO Algorithm

Fig. 4: (a) Multi-Threading ACO Algorithm, and (b) Structure of Ant thread

IV. SIMULATION RESULTS

The algorithms are implemented using C programming language and run on a PC with Intel Core™ i3-4005U CPU@1.70GHz x 4 and a 4GB RAM. Thread library used is POSIX threads on Ubuntu 12.04.1.

The performance has been tested on six test images Camera­man, Chairs, Lena, Shapes, Peppers and Mandril, as depicted
in Fig. 6 through Fig. 9. Various parameters used in proposed approach are set to their commonly used values as initial pheromone value $\tau_{ini} = 0.0001$. Number of iterations ($N$) and construction steps ($L$) are 3 and 40 respectively whereas number of ants $K = \sqrt{h \times w}$. Constant parameters $\alpha = 1$, $\beta = 0.1$, $\psi = 0.05$ and $\epsilon = 0.1$. Sleep factor for increasing randomness is set to 0 or 1.

The performance of conventional ACO and the proposed algorithm is compared in terms of their edge detection capability and execution time. In Fig. 5 through Fig. 9, sub-figure (a) shows the original image, (b) shows the edges detected using conventional ACO, (c) depicts the edges obtained using the proposed Multi-threaded ACO with the Sleep parameter set to ‘1’, and (d) shows the edges obtained using the proposed Multi-threaded ACO with the Sleep parameter set to ‘0’.

The results of proposed method i.e. by using multi-threading with ACO are much comparable with the results generated by using only ACO, but in the less computation time. The computation time of both methods are shown in Table I with the percentage decrement in time. The factor sleep in multi-threading also plays an important role. It actually increases randomness among ants for movement. If we increase the factor sleep from 0 to 1 then computation time increases but performance in finding true edges increases. But still this increase time due to sleep 1 is less than the computational time of conventional ACO. So it can be concluded that Multi-threading provides a better way for detecting edges and getting similar result with greater efficiency.

V. CONCLUSIONS

Algorithms were presented for detecting edges by using ant colony optimization and multi-threading based ACO. Firstly edges were identified by using conventional methods of edge detection using only ant colony optimization. It found the edges by fixing the order of movement of ants and the order of their steps; whereas the proposed method employed the concept of threads in ant colony optimization. Each step and order of movement of ants was random in that case. It was seen that the results were much improved with lower number of steps and in lesser computation time. Moreover, the number of false edges was significantly reduced in the proposed algorithm. Comparison results of both methods were also provided in the paper by varying the factor of sleep from 0 to 1. Computation time of both methods were also presented for showing the superiority of proposed method over the conventional approach. The multi-threading algorithm concept may be applied on multi-processor systems for reducing the computational time for future research work. Some improved thresholding methods may also be incorporated for enhancement in performance.

REFERENCES


<table>
<thead>
<tr>
<th>Original Image</th>
<th>Conventional ACO</th>
<th>Multi-Threaded ACO (sleep=1)</th>
<th>Multi-Threaded ACO (sleep=0)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Execution Time</td>
<td>Improved Percentage</td>
<td>Execution Time</td>
</tr>
<tr>
<td>Shapes</td>
<td>59.20</td>
<td>53.93 (5.52%)</td>
<td>49.59 (16.23%)</td>
</tr>
<tr>
<td>Lena</td>
<td>421.38</td>
<td>350.70 (16.77%)</td>
<td>305.06 (27.60%)</td>
</tr>
<tr>
<td>Cameraman</td>
<td>354.14</td>
<td>291.76 (17.61%)</td>
<td>251.59 (28.96%)</td>
</tr>
<tr>
<td>Chairs</td>
<td>385.85</td>
<td>304.19 (21.16%)</td>
<td>263.36 (31.75%)</td>
</tr>
<tr>
<td>Pepper</td>
<td>30.33</td>
<td>24.71 (18.53%)</td>
<td>20.91 (31.06%)</td>
</tr>
<tr>
<td>Mandril</td>
<td>42.33</td>
<td>39.19 (7.42%)</td>
<td>32.39 (23.48%)</td>
</tr>
</tbody>
</table>

TABLE I: Execution time (in seconds) using conventional ant-colony optimization and proposed method (ant-colony optimization with multi-threading) by taking sleep=1 and sleep=0.
Fig. 5: (a) Original image (Lena) (b) Edges with conventional ACO (c) Edges with proposed ACO with sleep=1 (d) Edges with proposed ACO with sleep=0

Fig. 6: (a) Original image (Cameraman) (b) Edges with conventional ACO (c) Edges with proposed ACO with sleep=1 (d) Edges with proposed ACO with sleep=0

Fig. 7: (a) Original image (Chairs) (b) Edges with conventional ACO (c) Edges with proposed ACO with sleep=1 (d) Edges with proposed ACO with sleep=0

Fig. 8: (a) Original image (Mandrill) (b) Edges with conventional ACO (c) Edges with proposed ACO with sleep=1 (d) Edges with proposed ACO with sleep=0

Fig. 9: (a) Original image (Peppers) (b) Edges with conventional ACO (c) Edges with proposed ACO with sleep=1 (d) Edges with proposed ACO with sleep=0