University of Stavanger

Faculty of Science and Technology

MASTER'S THESIS

Study program/ Specialization: Information Technology/ Computer Science	Spring semester, 2011 Open
Writer: Vegard Foldøy Thorsen	…… (Writer's signature)
Faculty supervisor: Tom Ryen	
Title of thesis: Meteorite Impact Crater Crack Extraction using Artificial Ants	
Credits (ECTS): 30	
Key words: Crack Extraction Edge Detection Ant Colony Optimization (ACO) Direction Control Mathematical Morphology	Pages: 57 + enclosure: CD/DVD Stavanger, 15.06.2011 Date/year

Front page for master thesis
Faculty of Science and Technology
Decision made by the Dean October 30th 2009

Summary

This Master's thesis deals with extracting crack information from images of rocks taken from a meteorite impact crater. By the use of artificial ants, and mathematical morphology, it becomes possible to detect the edges in the images, or more specifically, the cracks outlines.

Ant Colony Optimization (ACO) is a method inspired by nature and the social behavior of some ant species. ACO impersonates real ants and their foraging behavior where a chemical substance, pheromone, is deposited on the ground in order to mark a favorable path between the colony and a food source. This pheromone trail ensures that other members of the colony follows this exact same path. When working with image edge detection it is possible to make use of this phenomenon. By exploiting a similar mechanism and letting artificial ants traverse the images, they can detect the edges in the images by mimicking the colony behavior of real ants. Furthermore is the original ACO method enriched by a new direction control feature making the ants more adept at detecting edges belonging to a crack. Experiments show the effectiveness of directed artificial ants in detecting edges in a digital image.

Mathematical Morphology (MM) is a technique that builds on mathematical set theory and the studies of sets. The basic idea in mathematical morphology is to probe an image with a pre-defined shape, a structuring element, drawing conclusions on how this shape fits or misses the shapes in the image. A logical operation is performed between this structuring element and the underlying image, resulting in different effects depending on the morphological operator being used. When working with image edge detection it is possible to utilize this technique. By adjusting the structuring element to the features in the images, the edges in the images can be detected using various morphological operators. Experimental results demonstrates the usefulness of mathematical morphology in detecting edges in a digital image.

Based on the obtained edge information, a novel approach utilizing statistics and shape properties are developed in order to fill the gap between two corresponding edges. That way, the actual cracks becomes highlighted. Experimental results demonstrates the practicability of the proposed approach in extracting crack information from a digital image.

Preface

There are several ways of selecting a Master's thesis project. The faculties provides their own projects which they want students to undertake, or one can consult companies and investigate whether there are possibilities for writing your thesis in cooperation with them. This thesis is an individual assignment, given by the Faculty of Science and Technology at the University in Stavanger, that completes my studies as a computer scientist. The work started out as a preliminary project [1] in computer science the fall of 2010.

Working with the assignment was a rewarding personal experience. The amount of research involved was a true challenge that has without a doubt given me much valuable knowledge and insight into conducting self-studies. Also, having a supervisor to relate to over a longer period of time, as well as keeping up with deadlines, provides experience one truly cannot get through traditional teaching.

I am personally really enthusiastic about the end results, and I am confident that what is achieved here has value. I hope that time will be given to test the actual implementation. The algorithm works very well considering the focus of the project. Besides, there is great potential for further development which is put to an end only by the thesis natural time frame.

I would like to thank my faculty supervisor, associate professor Tom Ryen, at the University in Stavanger. He has been of great help and undoubtedly a valuable resource to have during the assignment.

I would also like to thank geologist Fridtjof Riis for taking the time to meet with me and for providing constructive feedback regarding the results. His knowledge on the subject has truly been beneficial as far as the end results are concerned.

Stavanger June 15, 2011

Vegard Foldøy Thorsen

Contents

List of Algorithms

List of Figures

List of Tables

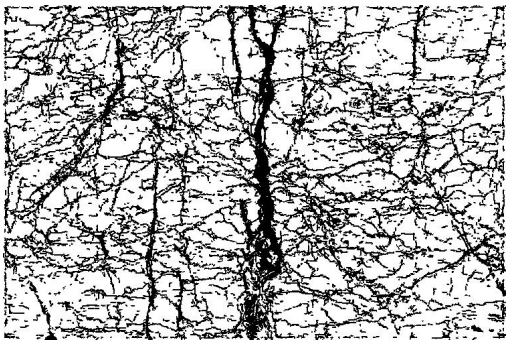
1 Introduction

The introduction concerns the project background, the problem description as well as the project goal.

1.1 Background

Recently (2000), a meteorite impact crater was discovered in Ritland, Norway. The meteorite made its impact with earth's surface roughly 500 million years ago, resulting in an approximately 400 m deep and 2.5 km wide impact crater. Today, there are areas with a lot of cracks in the crater rocks (Figure 1a). Geologists are interested in analyzing these cracks with respect to their length, width and orientation.

(a) Original crack image.



(b) Extracted crack image.

Figure 1: Image from the Ritland meteorite impact crater.

1.2 Problem Description

This project is about extracting crack information from images of the impact crater rocks (Figure 1b). Investigate whether artificial ants, as well as mathematical morphology, are capable of detecting the edges in the images, hence, providing a complete and clear edge trace of the cracks outline. Then, based on the highlighted edges, the crack itself should become identifiable by filling the gap between two corresponding edges.

1.3 Project Goal

The goal of this project is to generate (binary) images showing the cracks in the crater rocks. Based on these images, the cracks' length, width and orientation can be automatically measured and quantified. Considering the fact that these cracks mostly is long and narrow openings, special attention is given to the more linear trends in the image. Obviously, a result image free of noise and false cracks is unattainable. After all, not all edges in the images belongs to a true crack. Ultimately, extracting as much real cracks as possible, both big and small, is desired in order to better determine the trends in the images. Therefore, noise and false cracks is to be expected.

1.4 Report Outline

The outline of the report is as follows. Section 2 deals with artificial ants and ant colony optimization in image edge detection. It gives a brief introduction to both concepts, as well as providing a new ACO-based approach to image edge detection. Section 3 concerns the use of mathematical morphology in image edge detection. Basic morphological operators are explained and combined into morphological edge detectors. Section 4 introduces a novel approach that describes how the detected edge information can be used to extract crack information. Experimental results are presented in Section 5. Then, Section 6 concludes the report with Section 7 discussing future work on the subject.

2 Ant Colony Optimization (ACO)

Ants communicate with each other using pheromones. They leave pheromone trails on the ground in order to mark a path between their colony and a food source for other members in the colony to follow. The more ants following the same path, the higher its pheromone concentration becomes. Over time pheromone trails evaporate. The longer it takes for an ant to walk back and forth, the more time the pheromone has to evaporate. Hence, pheromone density remains higher at shorter and more favorable paths where pheromone is deposited at a much higher rate. This behavior helps ants successfully establish, and follow, the better paths. ACO [4] is inspired by this foraging behavior.

There exists several ant colony optimization algorithms [4], whereas the original algorithm, known as Ant System (AS), originates from the early nineties. Since then, a number of other algorithms, among the more successful variants MAX-MIN Ant System (MMAS) and Ant Colony System (ACS), were introduced. Several ACO-based approaches have been proposed to the edge detection problem [5, 6].

In this project, a new ACO-based approach is applied to image edge detection. The approach makes use of improvements introduced in ACS, with the addition of a new *direction control* feature. The reason for introducing direction control is to make the ants better suited for detecting edges belonging to long and narrow openings, hence, edges belonging to cracks.

2.1 ACO in Image Edge Detection

Image edge detection deals with extracting edges in an image by identifying pixels where the intensity variation is high. There are many well-known edge detection algorithms [7]. Prewitt, Sobel and Canny, to mention a few. Although originating from the early days of computer vision, some are still considered state-of-the-art edge detectors.

Ant Colony Optimization introduces a different approach to image edge detection. In ACO, artificial ants «walk on» the image depositing pheromone where the intensity variation is high. A $h \times w$ two-dimensional image can be represented as a two-dimensional graph with the image pixels as its nodes (Figure 2). A pixel is connected to all adjacent pixels in an 8-connectivity neighborhood (Figure 5b on page 11).

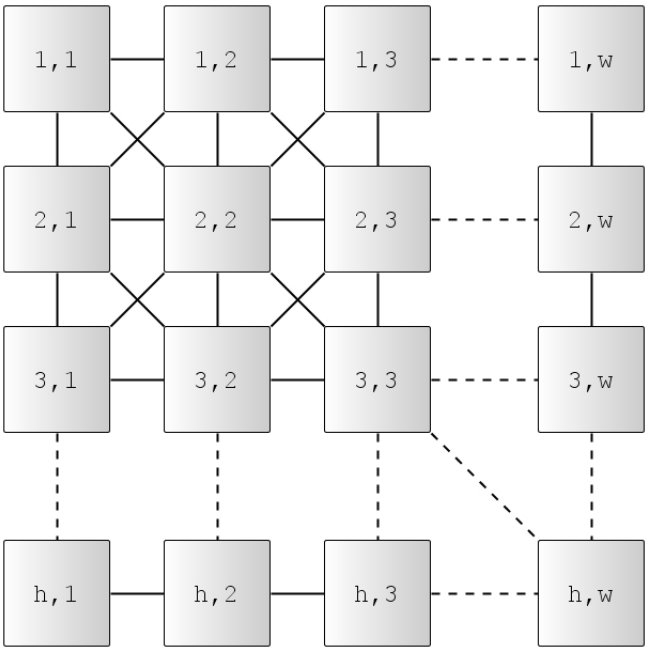


Figure 2: A graph representation of a $h \times w$ two-dimensional image.

ACO is an iterative probabilistic algorithm where ants are guided, by pheromone information, towards optimal paths in a graph. At each iteration, a number of artificial ants are considered. Each ant

Algorithm 1 The proposed ACO-based approach to image edge detection.

1. **Initialization Phase**

   Initialize a gray scale intensity value matrix, a pheromone matrix, a normalized intensity variation matrix (the heuristics) and a set of eight polar angles.

2. **Construction Phase**

   for construction step (round) $n = 1 : N$
   Randomly position all ants.
   for movement step $l = 1 : L$
   for ant $k = 1 : K$
   Select, and move ant to, next pixel.
   Immediate local update of the pixel's pheromone (pheromone decay).
   end
   end
   Offline update of all visited pixels pheromone (pheromone evaporation).
   end

3. **Decision Phase**

   The solution is made based on the values in the final pheromone matrix.

incrementally builds a (complete) solution by moving from pixel to pixel in the graph. An ant can only move to adjacent pixels with the constraint of not visiting any pixel more than once within the same iteration. The movement of the ants is affected by local variations in pixel intensity values. Ants select the following pixel to visit through a stochastic mechanism influenced by pheromone and heuristic information. The heuristic information is only dependent on the specific problem. Additionally, ants deposit a certain amount of pheromone on the traversed pixels. The actual amount deposited depends on the quality of the pixel. Subsequent ants make use of this pheromone information as a guide steering them towards the more promising regions in the graph. The goal is to construct a final pheromone matrix that reflects the edge information in the image. Each element in the pheromone matrix corresponds to a pixel in the image and indicates whether the pixel is on an edge or not.

The proposed ACO-based approach to image edge detection can be thought of as a three-phased process as described in Algorithm 1: The first phase is an initialization phase. The second, which is the construction phase, covers all ant movement. During this phase the pheromone information is continuously updated as the ants «walk on» the image. In the last phase, the decision phase, the solution is made from the values of the elements in the final pheromone matrix.

2.1.1 Initialization Phase

First, an *intensity value* matrix is made. Every element in the intensity value matrix has a value based on the gray scale intensity level related to the corresponding image pixel.

Secondly, a *pheromone* matrix is constructed. Every element in the pheromone matrix is assigned a small nonzero value τ_{init}, stating an initial pheromone level.

The heuristic information, one of the main aspects in the following construction phase, may be calculated already in the initialization phase. This is due to the fact that the heuristic only depends on the intensity values of the pixels in the image. In other words, the heuristic information is a (normalized) *intensity variation* matrix which is fixed for every construction step:

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

- $I_{i,j}$ is the intensity value of the pixel (i, j).
- $V_c(I_{i,j})$ reflects the intensity variation between the pixel (i, j) and a local group of surrounding pixels *c,* called a *clique* (Figure 3).
- V_{max} represents the maximum intensity variation in the whole image and serves as a normalization factor.

The value of $\eta_{i,j}$ is large for pixels located in regions of the image containing sharp intensity variations. Hence, pixels representing edges in the image.

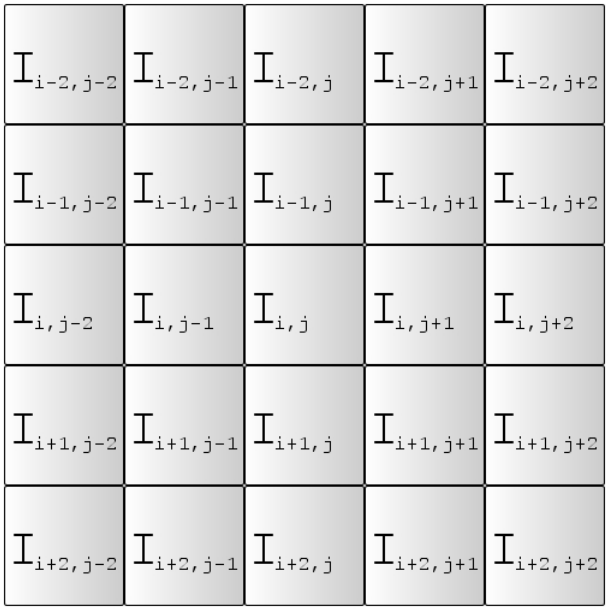


Figure 3: The clique at pixel *(i,j).*

Polar Angle = Value (in radians)		
$\theta_6 = \frac{5\pi}{4}$	$\theta_7 = \frac{3\pi}{2}$	$\theta_8 = \frac{7\pi}{4}$
$\theta_5 = \pi$	$(i\,,\,j)$	$\theta_1 = 0$
$\theta_4 = \frac{3\pi}{4}$	$\theta_3 = \frac{\pi}{2}$	$\theta_2 = \frac{\pi}{4}$

Table 1: The eight polar angles θ_u.

Calculating the intensity variation at pixel (i, j) is done as follows:

$$V_c(I_{i,j}) = \begin{vmatrix} |I_{i-2,j-2} - I_{i+2,j+2}| + |I_{i+2,j-2} - I_{i-2,j+2}| + |I_{i-1,j-1} - I_{i+1,j+1}| + |I_{i+1,j-1} - I_{i-1,j+1}| + \\ |I_{i-2,j-1} - I_{i+2,j+1}| + |I_{i+2,j-1} - I_{i-2,j+1}| + |I_{i-1,j-2} - I_{i+1,j+2}| + |I_{i+1,j-2} - I_{i-1,j+2}| + \\ |I_{i-2,j} - I_{i+2,j}| + |I_{i,j-2} - I_{i,j+2}| + |I_{i-1,j} - I_{i+1,j}| + |I_{i,j-1} - I_{i,j+1}| \end{vmatrix} \tag{2}$$

Large differences in intensity values between pixels located 180 degrees of each other, relative to pixel (i, j), gives a large variation in intensity. In other words, pixels located at edges have large intensity variations, hence, large values for $V_c(I_{i,j})$.

Note however, that the borders of an image (the two outermost pixels) does not satisfy a complete clique. For instance, the pixel located at the top left corner $(1, 1)$ does not have any west- nor north-laying neighbor pixels. Hence, no (normalized) intensity variation is calculated on the image borders. Instead are all border pixels initialized with a very small nonzero value. A small value is chosen for the simple reason of making the image borders less attractive to ants.

A set of eight *polar angles* is defined (in radians). Each angle θ_u is relative to the movement, in a horizontal right direction, of an ant located at pixel (i, j) and its adjacent neighboring pixels. Hence, there are only eight possible values for θ_u (Table 1).

Note that the use of polar angles is not originally a part of ACO. However, by introducing polar angles it becomes possible to implement direction control where ants favor pixels laying in their current moving direction. Like the heuristic information, the polar angles are fixed for every construction step and may be defined already in the initialization phase.

2.1.2 Construction Phase

Ant Movement

Ants execute a predefined number of rounds (construction steps).

In every round n each ant k is consecutively moved a fixed number of movement steps l. An ant moves from its current pixel (i, j) to one of its, not previously visited, neighboring pixels (x, y). Which neighbor pixel to go to is determined by a *transition probability* matrix:

$$p^{(n)}_{(i,j),(x,y)} = \frac{\left[\left(\tau^{(n-1)}_{x,y}\right)^{\alpha}(\eta_{x,y})^{\beta}\right] + [r(\cos(\theta_u - \varphi_{i,j}) + 1)]}{\Sigma_{(x,y)\in\Omega_{(i,j)}}\left[\left(\tau^{(n-1)}_{x,y}\right)^{\alpha}(\eta_{x,y})^{\beta}\right] + [r(\cos(\theta_u - \varphi_{i,j}) + 1)]} \tag{3}$$

- $\tau^{(n-1)}_{x,y}$ is the pheromone information at the neighbor pixel (x, y).
- α determines the influence of the pheromone information.
- $\eta_{x,y}$ is the (normalized) intensity variation at the neighbor pixel (x, y).
- β determines the influence of the intensity variation.
- r determines the influence of the direction control.
- θ_u is the polar angle given the ants current pixel (i, j) and its neighbor pixel (x, y).
- $\varphi_{i,j}$ is the angle given the ants current pixel (i, j) and its starting pixel (i_{start}, j_{start}) (Figure 4).
- $\Omega_{(i,j)}$ represents the set of neighborhood pixels for the ant currently located at pixel (i, j).
- The denominator represents a normalization of the transition probability $p_{(i,j)}$.

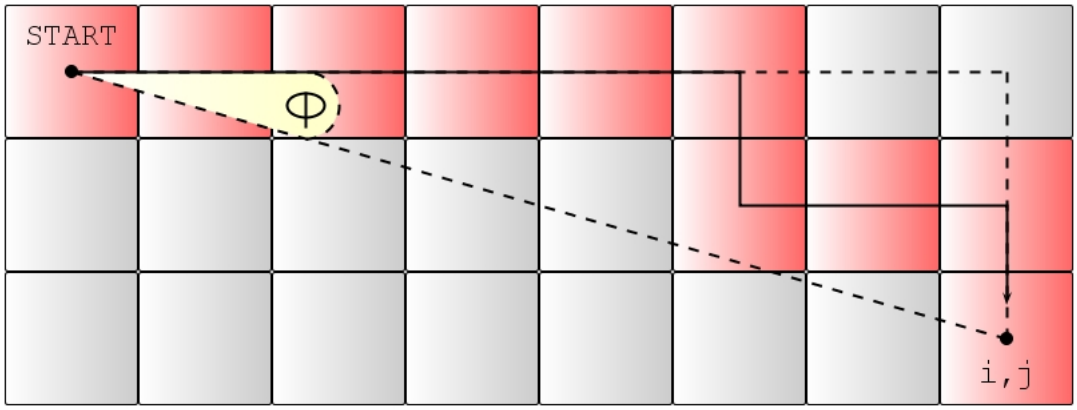


Figure 4: The angle $\varphi_{i,j}$ given the ants current pixel (i, j) and its starting pixel (i_{start}, j_{start}).

Calculation of the angle $\varphi_{i,j}$ is done as follows:

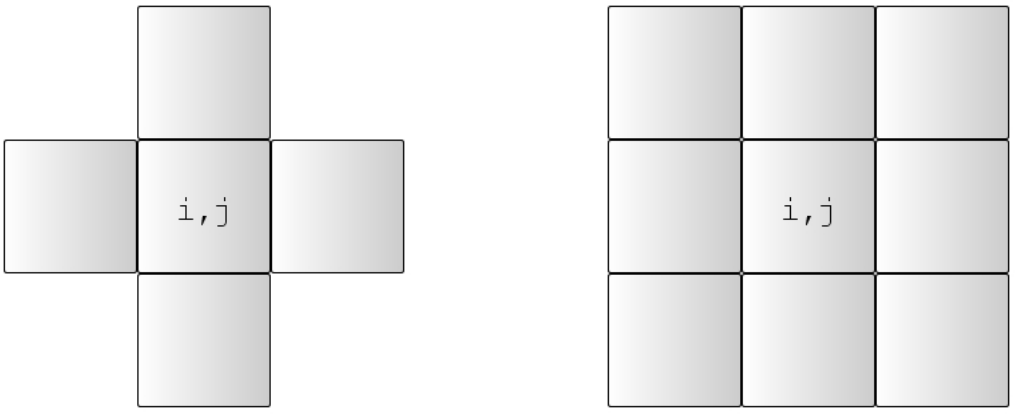
$$\varphi_{i,j} = \begin{cases} \arctan(\frac{i-i_{start}}{j-j_{start}}) & if\ j > j_{start} \\ \arctan(\frac{i-i_{start}}{j-j_{start}}) - \Pi & if\ j < j_{start}\ and\ i < i_{start} \\ \arctan(\frac{i-i_{start}}{j-j_{start}}) + \Pi & if\ j < j_{start}\ and\ i \geq i_{start} \\ -\frac{\Pi}{2} & if\ j = j_{start}\ and\ i < i_{start} \\ \frac{\Pi}{2} & if\ j = j_{start}\ and\ i > i_{start} \end{cases} \tag{4}$$

The value of $\varphi_{i,j}$ solely depends on the ants current pixel location (i, j) with respect to its starting pixel (i_{start}, j_{start}). It is not (directly) affected by any of the intermediate movement steps.

Note however, that φ is neither a part of the original ACO, but introduced as a part of the new direction control feature.

During ant movement, there are two issues which are considered crucial. The first is related to the already discussed (normalized) intensity variation (1), more specifically $\eta_{x,y}$ in (3). The second is about defining the permissible range of their movement, namely $\Omega_{(i,j)}$ in (3).

Ants may only move to adjacent (neighboring) pixels (Figure 5). In other words, it cannot move to a pixel which is not directly connected with the pixel where it is currently located. Various neighborhoods have been proposed in the literature [5, 6], whereas the one adopted in this report is the 8-connectivity version (Figure 5b).



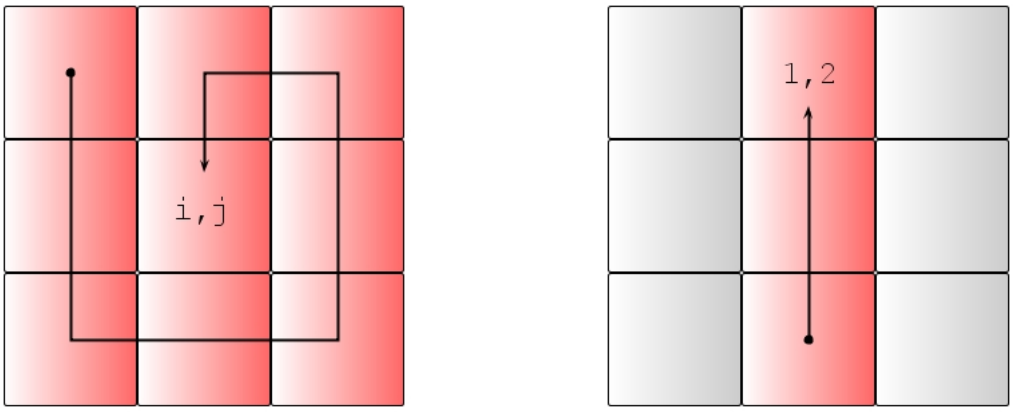
(a) 4-connectivity neighborhood. (b) 8-connectivity neighborhood.

Figure 5: Adjacent pixels using various neighborhoods.

Ant movement is further restricted by the condition that it is not permitted to visit any pixel more than once within the same round. This prevents ants from moving back and forth between the same set of pixels. Hence, ants are steered towards regions in the image unknown to them, resulting in a better coverage of the image as a whole. In order to keep track of the recently visited pixels each ant has a (round-based) memory. For next round, the ants memory are cleared.

However, refusing ants to revisit pixels leads to possible *deadlocks* (Figure 6a). A deadlock occurs when an ant, located at pixel (i, j), has visited all its neighboring pixels in the current round rendering the ant immobile.

Ants running into deadlocks are most likely to find themselves in less interesting (edgeless) areas of the image. Lack of pixels with sharp intensity variations guiding the ants forward can, and most probably will, cause some of the ants to deadlock. All ants that deadlock will be refused to move any further during the given round and forced to terminate without completing their remaining movement steps. The actual amount of ants deadlocking is highly dependent on several aspects of ACO. Parameters such as the total number of rounds, the total number of movement steps (per round), the total number of ants, as well as the random factor in connection with the ants starting positions, are all affecting the occurrence of deadlocks. Furthermore, the direction control feature has a great impact on the matter.



(a) Ant deadlocking at pixel *(i,j)*.

(b) Ant entering image border at pixel *(1,2)*.

Figure 6: Forced ant movement termination.

Ants entering *image borders* (Figure 6b) is an issue to be addressed. Image borders suffers from incomplete neighborhoods, which naturally affects neighborhood calculations. Therefore, ants encountering image borders are forced to end their movement even when there are unvisited pixels left in the neighborhood. Despite all efforts done in advance by making the image borders less attractive to ants, some will still choose a border pixel. The amount of ants entering image borders are greatly affected by the very same parameters as in deadlocking.

Ant Pheromone Deposition

Ants deposit a certain amount of pheromone during movement. The amount of pheromone deposited on a pixel varies depending on the actual quality of it. More pheromone is deposited on pixels representing edges in the image, compared to the amount of pheromone deposited on edgeless pixels. Subsequent ants then makes use of this variation in pheromone information to help them find the more promising areas of the image. Hence, guiding them towards the edges in the image.

The construction phase contains two separate pheromone updates. One immediate *local pheromone update* (pheromone decay), and one *offline pheromone update* (pheromone evaporation).

The local update is performed immediately after every movement step. In other words, after each ant k has executed a single movement step l within a round n, the pheromone matrix $\tau_{i,j}^{(n)}$ is updated:

$$\tau_{i,j}^{(n)} = (1-\psi)\cdot\tau_{i,j}^{(n-1)} + \psi\cdot\tau_{i,j}^{(0)} \tag{5}$$

- ψ is the pheromone decay coefficient.
- $\tau_{i,j}^{(0)}$ is the initial pheromone matrix.

Pheromone decay is meant to diversify the search performed by subsequent ants during the same round. By decreasing the pheromone concentration on the traversed pixels, subsequent ants are encouraged to choose other pixels, thus exploring other paths and providing a better coverage of the image.

The offline pheromone update is performed at the end of each round. In other words, after all K ants have finished all L movement steps within the same round n, the pheromone matrix is updated again:

$$\tau_{i,j}^{(n-1)} = \begin{cases} (1-\rho)\cdot\tau_{i,j}^{(n-1)} + \rho\cdot\eta_{i,j}^{(k)}, & if\ pixel\,(i,j)\,is\,visited \\ \tau_{i,j}^{(n-1)}, & otherwise \end{cases} \tag{6}$$

- ρ is the evaporation rate.

The pheromone information (each element in the pheromone matrix) is changed if, and only if, a pixel is visited by an ant. For all pixels left unvisited, the pheromone level remains the same.

Note that the pheromone evaporation update does not exactly follow the original ACS approach [4]. Some aspects of ant colony system do not suit the nature of image edge detection. In ACS, only the pixels belonging to a best-so-far tour is updated. Having a best-so-far tour makes sense when each ant produces a complete solution to the problem, which indeed was the case when ACO was introduced to solve the traveling salesman problem. In an ACO-based edge detection approach, however, an individual ant does not produce a complete solution to the problem. Each ant produces only a partial solution, hence, it's the ant collective that make up the complete solution. Therefore, it makes no sense to introduce a best-so-far tour in image edge detection.

2.1.3 Decision Phase

The solution is based on the values in the final pheromone matrix. The literature applies a threshold technique, also known as the Otsu threshold technique [8], to reduce the resulting gray scale image to a binary image with only two possible values for each pixel. This is done to be able to classify each pixel as either an edge or a non-edge. Though, when it comes to analyzing the work carried out by the ant collective in image edge detection, a result showing various degrees in intensity values (gradient) is just as good as a black and white declaration. Hence, in ant image edge detection, the solution is a direct result of the values in the final pheromone matrix.

3 Morphological Image Processing

Morphological image processing [9, 10, 11] is developed from Mathematical Morphology (MM) and mathematical set theory. The technique is often used to detect object boundaries (edges), skeletons and convex hulls in images. Likewise, it is frequently used as a pre- and post- processing technique for thinning and pruning of edges.

Morphological image processing operates by the use of structuring elements and morphological operators. A structure element is a binary image of any size that contains any combination of 1's and 0's. The process involves passing this structure element over the pixels in the image whilst performing a logical operation between them. The outcome of this operation is dependent on the size and content of the structuring element as well as the morphological operation being used. While originally intended for binary images, morphology has been extended to gray scale images as well.

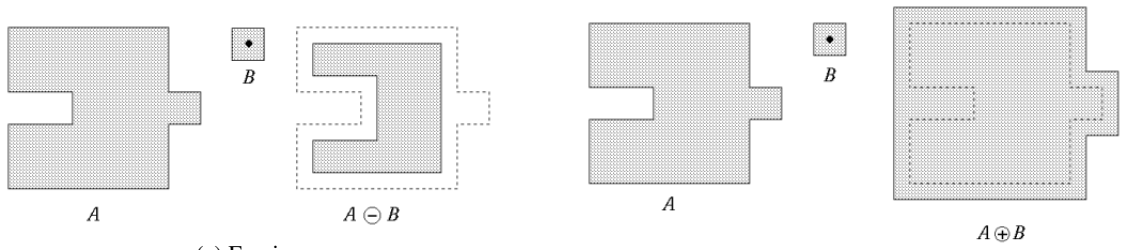
3.1 Morphological Operators

Generally speaking, most morphological operators are based on simple shrinking and expanding operations. The two most common morphological operators is *erosion* and *dilation* (Figure 7). Erosion makes objects smaller (shrinking) whilst dilation makes objects larger (expanding). Composite operations such as *opening* and *closing* (Figure 8) is obtained by combining these two basic operations. In the following:

- A denotes the binary image.
- B denotes the structuring element.
- i, j denotes the current pixel location.
- $B_{i,j}$ denotes the structuring element with its origin located at pixel *i,j*.

Erosion

In binary erosion (Figure 7a), where every image pixel only has two possible values, every pixel located inside an object that has at least one neighbor outside of the object is eliminated. In other words, every



(a) Erosion.

(b) Dilation.

Figure 7: Erosion and dilation [2].

object pixel that is touching a background pixel is itself changed into a background pixel. Therefore, erosion makes objects smaller by one pixel all around. In fact, if repeatedly applied, erosion will shrink any object out of existence. It will also disconnect, break an object into multiple objects, at any point less than three pixels thick. Erosion is useful for removing objects from an image that are too small to be of interest. Binary erosion is defined by:

$$A \ominus B = \left\{ i, j \mid B_{i,j} \subseteq A \right\} \tag{7}$$

Gray scale erosion is analogous to its binary counterpart. In gray scale erosion, where image pixels can have more than two possible values, the minimal value of the current pixel and its neighbor pixels is used. The value of the current pixel is set to this minimum value. The amount of neighbors considered depends on the size of the applied structuring element. Therefore, gray scale erosion has the effect of darkening small and bright areas. Moreover might very small areas, only a few pixels wide, be completely eliminated.

Dilation

In binary dilation (Figure 7b), every pixel located outside an object that has at least one neighbor inside of the object is incorporated into the object. In other words, every background pixel that is touching an object pixel is itself changed into an object pixel. Therefore, dilation makes objects larger by one pixel all around. Applying dilation repeatedly will merge all the objects in an image into one. Likewise, all objects separated by less than three pixels at any point, will be merged at that point. Dilation is useful for filling holes in objects. Binary dilation is defined by:

$$A \oplus B = \left\{ i, j \mid B_{i,j} \cap A \neq \emptyset \right\} \tag{8}$$

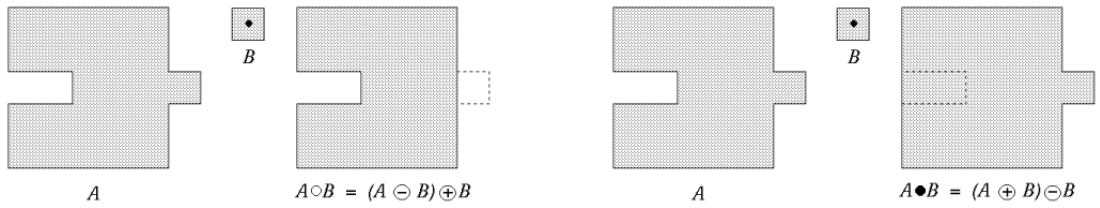
Gray scale dilation is analogous to binary dilation. In gray scale dilation, the value of a pixel is set to the maximal value between itself and its neighbors. Therefore, gray scale dilation has the effect of brightening small and dark areas in a gray scale image. Furthermore, very small and dark holes might get completely eliminated.

Opening

Just as with dilation and erosion are opening and closing dual operations.

Opening (Figure 8a) is simply an erosion followed by a dilation. Hence, it eliminates small and thin objects as well as it breaks larger objects at thin points. Additionally, and because the resulting set is dilated using the same structure element, opening also has the effect of smoothing the (inner) boundaries of larger objects, without significantly changing their area. Opening is defined by:

$$A \circ B = (A \ominus B) \oplus B \tag{9}$$



(a) Opening. (b) Closing.

Figure 8: Opening and closing [2].

Closing

Closing (Figure 8b) is obtained by simply swapping the order of erosion and dilation. Hence, it fills small and thin holes in objects as well as it connects nearby objects. Closing also has the additional effect of smoothing the (outer) boundaries of objects, without significantly changing their area, since the resulting set is being eroded using the same structure element. Closing is defined as follows:

$$A \bullet B = (A \oplus B) \ominus B \tag{10}$$

Both gray scale opening and gray scale closing is defined like in binary morphology, (9) and (10) respectively.

3.2 Morphological Image Processing in Image Edge Detection

When applying morphological image processing to image edge detection [10], various compositions of morphological operators make up different morphological edge detectors. The results obtained by the use of morphological edge detectors is highly dependent on both the configuration of the operators in the edge detector itself, as well as the configuration of the structuring element. More precisely, a successful morphological edge detection is based on both the design and structure of the edge detector, as well as the size, shape and orientation of the structuring element. The following algorithms are all commonly used when applying morphological image processing to image edge detection:

Erosion Residue Edge Detector

The edge of an image can be identified as the difference between the image itself and its eroded set. This is known as the erosion residue edge detector, which is defined as:

$$A - (A \ominus B) \tag{11}$$

Dilation Residue Edge Detector

Similarly, the edge of an image can be identified as the difference between its dilated set and the image itself. This is also known as the dilation residue edge detector:

$$(A \oplus B) - A \tag{12}$$

Morphological Gradient Edge Detector

Morphological gradient edge detector highlights sharp gray level transitions by taking the difference between the dilated set and the eroded set of an image:

$$(A \oplus B) - (A \ominus B) \tag{13}$$

Reduced Noise Morphological Gradient Edge Detector

The reduced noise morphological gradient edge detector is a much more complex algorithm, proposed in [10], and defined as:

$$(M \bullet B) \oplus B - (M \bullet B) \tag{14}$$

$$M = (A \bullet B) \circ B \tag{15}$$

Structure Elements

In order to get clear image edges the selection of an appropriate structuring element is of utmost importance. The structure elements used in practice is generally much smaller than the image itself, and often a 3×3 matrix. Consequently, the structure elements used in this approach is all 3×3 matrices (Table 2).

0	0	0
1	1	1
0	0	0

(a) 0 deg SE.

0	0	1
0	1	0
1	0	0

(b) 45 deg SE.

0	1	0
0	1	0
0	1	0

(c) 90 deg SE.

1	0	0
0	1	0
0	0	1

(d) 135 deg SE.

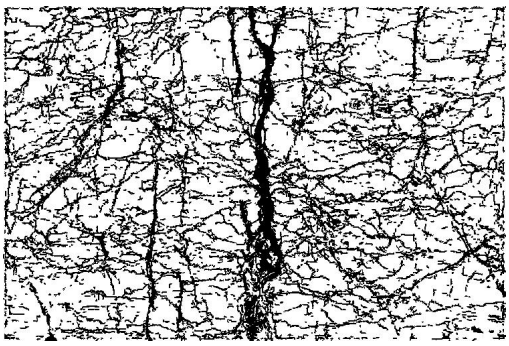
Table 2: Four flat structuring elements (SE).

Selecting an appropriate structure element comes down to studying the features in the image. Considering the fact that edges are fairly thin and relatively straight lines, simple line-based structuring elements seems most appropriate. Hence, flat linear structure elements that is symmetric with respect to the neighborhood center is created. In total, and for orientation purposes, four different structuring elements have been made. Each structuring element, measured in a counterclockwise direction from the horizontal axis, covers 0, 45, 90 and 135 degrees, respectively. This is done as an attempt to cover the main directions in the images.

4 Crack Extraction

So far the focus has been on detecting the edges in the images. That way the cracks outlines can be highlighted (Figure 9a). However, by studying only an edge trace, it is not easy to determine which side of the edge that actually represents a crack. Ultimately the crack itself should be identified and not just its outline. More specifically, the gap between two corresponding edges (containing the actual crack) should be filled (Figure 9b).

(a) Highlighted crack outlines.



(b) Filled cracks.

Figure 9: Crack extraction.

Algorithm 2 The proposed novel crack extraction approach.

1. **Thresholding**

   Reduce the (gray scale) edge information to (binary) crack information.

2. **Noise Filtering**

   Remove circle-like components.

3. **Crack Filling**

```
for iteration n = 1 : N
    for crack pixel m = 1 : M
        if n == 1
            Expand the crack outline by one pixel into the crack.
        else
            Color new neighbor crack pixels.
        end
    end
end
```

Edge detection is a fundamental tool in image processing, particularly in the area of feature detection, which aim at identifying points in an image at which the image brightness changes sharply or, more formally, has discontinuities. Unlike in image edge detection there is no well-known techniques or methods for extracting cracks in an image. Therefore, a novel approach based on the obtained edge information is developed in order to identify the actual cracks rather than just highlighting their edges.

The proposed novel crack extraction approach is described in Algorithm 2.

4.1 Edge Thresholding using Statistics and Percentiles

The first step towards extracting cracks is to reduce the gray scale edge information to binary crack information using thresholding. That way, with only two possible values for each pixel, every pixel is classified as either a crack or a non-crack.

During the thresholding process, individual pixels in the edge image are marked as crack pixels if their value is larger than some threshold value, and as background (non-crack) pixels otherwise. Finally, a binary crack image is created by coloring each pixel either black or white, depending on whether it is marked as a crack pixel or a background pixel. Needless to say, this threshold value plays an important role when it comes to marking the appropriate pixels. Therefore, and in order to get a threshold value that best represents the underlying edge image, statistics and percentiles are applied.

In statistics, a percentile provides an indication of how the data values, sorted from smallest to largest, are spread. Approximately p percent of the data values fall below the pth percentile, and roughly $100-p$ percent of the data values are above the pth percentile. In other words, by arranging all the pixel values in the edge image from smallest to largest, it becomes possible to mark all pixels having a value above the specified percentile as a crack. All remaining pixels is simply converted to background pixels. As a result, every pixel in the edge image is reduced to either a crack or a non-crack pixel. Note however, that the percentiles in this work are specified using percentages.

4.2 Noise Filtering using Connected Components Eccentricity Property

A threshold process will generate a certain amount of noise, hence, unwanted information. Typically this noise represents itself as randomly occurring pixels which has abnormal intensity levels compared to their neighboring pixels. In detail, a (black) crack pixel may appear in the middle of a (white) non-crack area. Obviously, this is not a desired effect. Therefore, a method for removing noise is implemented.

First, all connected components in the binary image is found using an 8-connectivity neighborhood. This means that the binary image is processed, pixel by pixel, grouping all adjacent crack pixels into one component. For this reason, all cracks that are connected to other cracks, in an 8-connectivity neighborhood, will constitute one single composite component. Each connected component is then measured according to its eccentricity property.

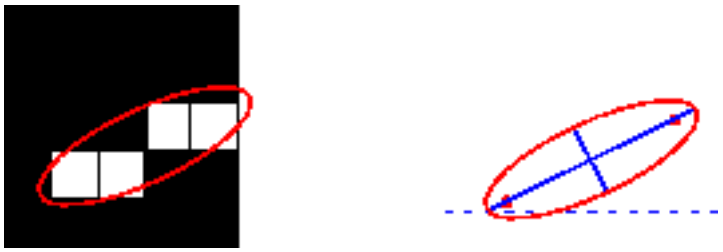


Figure 10: Axes and orientation of the ellipse [3].

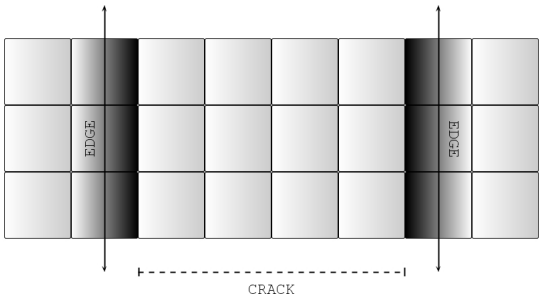
The eccentricity is the ratio of the distance between the foci of an ellipse (comprising the pixels in the component) and its major axis length (Figure 10). The value is between 0 and 1. An ellipse whose eccentricity is 0 is actually a circle, while an ellipse whose eccentricity is 1 is a line segment. By taking advantage of this fact, and by simply comparing a connected components eccentricity value against a specified value between 0 and 1, it becomes possible to remove complete circle-like components. More precisely, all connected components that has an eccentricity ratio less than the pre-defined value is filtered away. This seems appropriate considering the fact that the focus is biased towards linear trends (line segments) in the images.

4.3 Crack Filling by Comparing Gray Scale Values

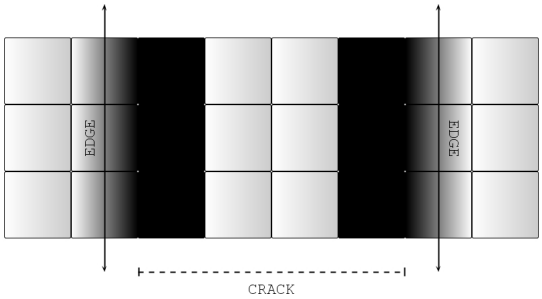
Last step of the crack extraction process consists of simple comparisons between gray scale values. Every crack pixel has a corresponding gray scale value found in the underlying gray scale image. Up to this point all crack pixels are a direct result of edge detection. Hence, all crack pixels are in fact binary edge pixels where only one side of a pixel is considered valid as far as being a part of the actual crack. Therefore, filling the appropriate gap between two corresponding binary edge pixels (Figure 11) is a twofold iterative approach where the first iteration differs from the rest. Furthermore, the marking of new crack pixels is done separately. For that reason, during a new iteration, only pixels marked in the previous iteration is considered. That way the amount of pixels to process becomes smaller for every additional iteration.

The first iteration can be though of as “moving one pixel away from the edge and thus into the crack”. During the first iteration, a crack (binary edge) pixel compares its neighbors gray scale values against each other in a total of four ways. The gray scale value of its left neighbor is compared to the gray scale value of its right neighbor. The neighbor that has the smallest value are marked as a new crack pixel if, and only if, its value is also smaller than the current pixels gray scale value. The same reasoning applies to the other opposite neighbors. As a result, all binary edges are expanded by one pixel into the crack (Figure 11b).

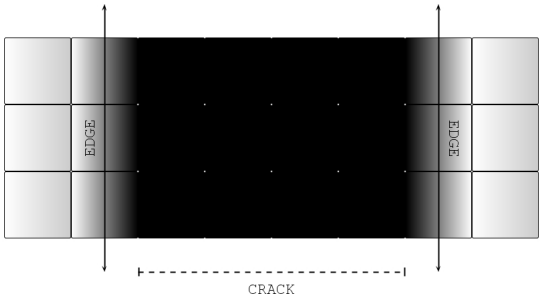
The remaining iterations is somewhat different. Like in edge thresholding, percentiles is here used in order to find a sufficiently small value, based on the values in the gray scale image, to use as the definition of a crack pixel. More specifically, in order to be marked as a crack pixel, the gray scale value of a pixel needs to be lower than this percentile value. That way, it becomes feasible to simply pick every



(a) Before crack filling.



(b) After 1*st* crack filling iteration.



(c) After *nth* crack filling iteration.

Figure 11: Crack filling.

neighbor with a gray scale value smaller than the pre-defined crack value. Considering that every crack pixel is now in fact located inside an actual crack, this proves to be useful. Moreover, such an approach is appropriate considering that small gray scale values often is an indication of a crack pixel.

The complete crack extraction process is summarized in Algorithm 3.

Algorithm 3 The complete crack extraction process.

1. **Preprocessing**

   Read the original crater image, resize it to the appropriate image size, convert to gray scale and apply a Gaussian blur filter.

2. **Ant Edge Detection** (Algorithm 1)

3. **Crack Extraction** (Algorithm 2)

5 Experimental Results

Experiments are carried out in order to evaluate the performance of the proposed approaches using a selection of images from the Ritland meteorite impact crater. The three images *8372*, *8382* and *8397* (Appendix A) are used as the training data set, and for the sole purpose of deciding appropriate parameter values. Then, experiments are carried out on a set of sixteen test images (Appendix B) independent of the training data.

The original image resolution of 3008×2000 pixels is fairly high to process within a reasonable amount of time. Naturally, the bigger an image is the more computational costly and time consuming it becomes to process it. Though using a higher resolution than the typical 128×128 and 256×256 used in the literature is desired. The image size adopted in this report, 752×500, contains a fairly high detail level as well as it maintains the original images aspect ratio. All images processed are converted to gray scale images. In other words, the value of each pixel contains only intensity information varying from black at the weakest to white at the strongest. A Gaussian blur [12] (smoothing) low-pass filter is also used to reduce image noise. The following loss of information is not considered an issue since the project concerns the extraction of dark cracks in gray crater rocks.

All parameters are assigned values that have been found to produce good results. When deciding an appropriate parameter value (5.1 and 5.3), all other parameters are fixed. That way only the parameter being discussed is affecting the result. Parameter values not discussed are adopted from the literature.

ACO Parameters

τ_{init} amount of initialized pheromone

- 0.00000001

N total number of construction steps (rounds)

- 5

L total number of ant movement steps (per round)

- 250

K total number of ants

- $\lfloor \sqrt{I_w \times I_h} \rfloor = \lfloor \sqrt{752 \times 500} \rfloor = 613$

r influence of direction control

- 0.1

α influence of pheromone information

- 1.0

β influence of intensity variation

- 1.0

ρ pheromone evaporation

- 0.1

ψ pheromone decoy

- 0.05

Crack Extraction Parameters

N total number of iterations

- 25

λ edge percentile

- 85%

γ gray scale percentile

- 7.5%

ε eccentricity property

- 0.25

The proposed approaches is implemented using the Matlab programming language and tested remotely on a server cluster with 64-bit Linux operating system, 16 CPU's running at $3334MHz$ and a total of $\sim 74GB$ memory. The computational times varies between the different methods. Morphological edge detection is close to real-time detection and takes roughly a second to complete. Ant edge detection however takes about seven minutes, using the appropriate parameter values, in order to finish. The novel crack extraction approach completes in less than a minute.

The complete Matlab source code, consisting of a total of four m-files, is listed in Appendix C.

5.1 Experimental Ant Edge Detection Results

Experimental results demonstrates the effectiveness of artificial ants in detecting edges in a digital image.

In ant edge detection the white lines follows the paths where the ants have been walking. The stronger and more intense this white color is, the larger the number of ants walking that path successfully depositing pheromone on its trail. Hence, white lines is an indication of edges in the image. Darker regions means little to no pheromone. However, it does not necessarily mean that no ants have never explored that part of the image during the construction process. It simply means that too few ants have been continuously walking there, hence, little to no pheromone is frequently being deposited. Then, all pheromone deposited have evaporated over time.

Comparing Direction Controlled Ants to Undirected Ants

Direction control lets ants favor pixels laying in their current moving direction. In other words, if an ant is walking north it continues to favor north-laying pixels over other pixels. That way ants are discouraged from making sharp u-turns and the probability of moving in circles is reduced. In a general edge detection scenario, where every edge is of equal importance, such a feature would perhaps not serve its purpose. However, when focusing on the more linear edges, letting ants follow more or less straight paths makes the whole difference, as illustrated in Figure 12. Here, all parameter values are identical. The only difference is that ants, in Figure 12b, are in addition direction controlled.

As seen in Figure 12c, undirected ants generate a lot of round and almost circle-like patterns. This is the result of ants moving around in edgeless regions of the image. Lacking pixels with sharp intensity variations to guide the ants forward, may cause them to move in circles until there is no more unvisited pixels left in the adjacent neighborhood. Hence, a deadlock occurs. Since direction controlled ants are discouraged from such circular movement, the probability of an ant having visited all its neighbors naturally becomes very small. Hence, the probability that an ant deadlocks becomes very small. This is demonstrated in Figure 12b, where these circles are gone.

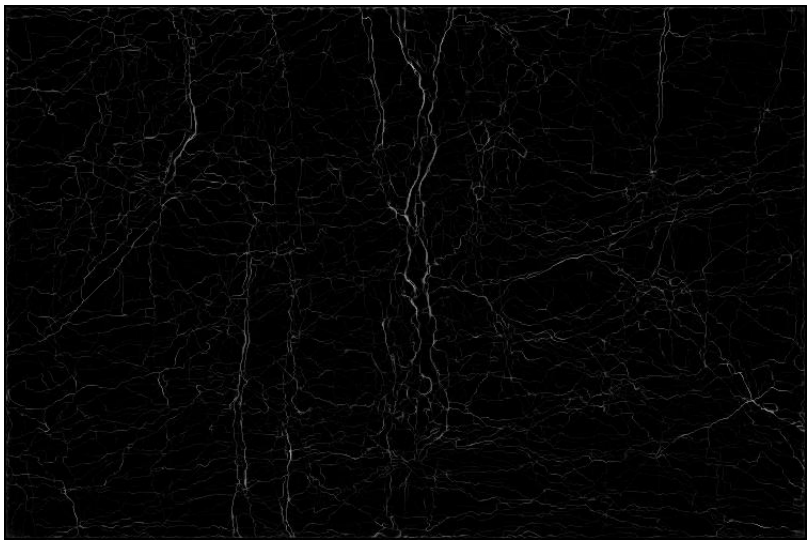
However, the number of ants entering image borders, border ants, are increased since ants are moving in straight lines. Naturally, when walking more or less straight paths, the distance a single ant covers becomes larger. The amount of visited pixels remains the same, but a direction controlled ant will end up at a far more remote location as compared to an undirected ant. The probability that it enters an image border is therefore increased considering the high amount of movement steps. By taking a closer look at the pixels located close by the image borders, in Figure 12b, this is easy to see. It looks like a white frame around the image. This is related to the fact that a lot of ants are indeed depositing pheromone around the image borders. This is not the case with undirected ants, as seen in Figure 12c.

In other words, direction control affects both the number of ants that deadlock as well as the number of border ants. Although having a slightly negative effect with respect to the number of border ants, the reduced occurrences of deadlocks makes it worthwhile.

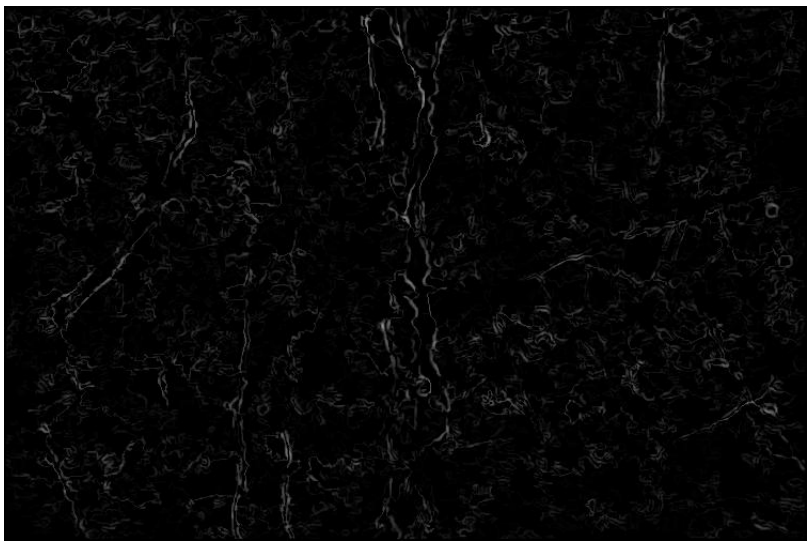
Influence of Direction Control

When it comes to the influence of the direction control, the difference between too little and too much is crucial. Too little direction control means, in practice, the same thing as undirected ants. While the

(a) Original crater image.

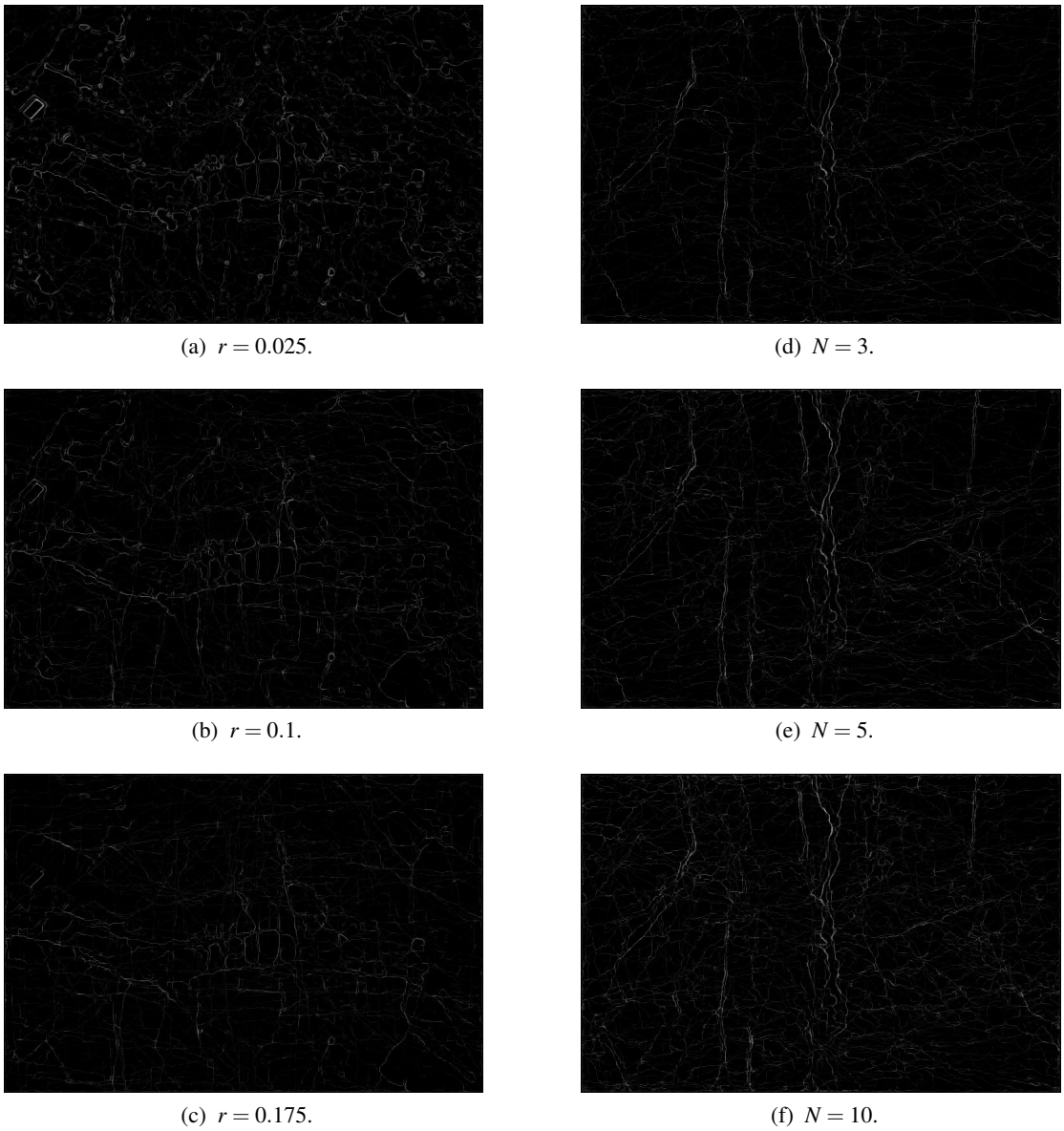


(b) Direction controlled ants.



(c) Undirected ants.

Figure 12: Direction controlled ants vs. undirected ants.



(a) $r = 0.025$. (d) $N = 3$.

(b) $r = 0.1$. (e) $N = 5$.

(c) $r = 0.175$. (f) $N = 10$.

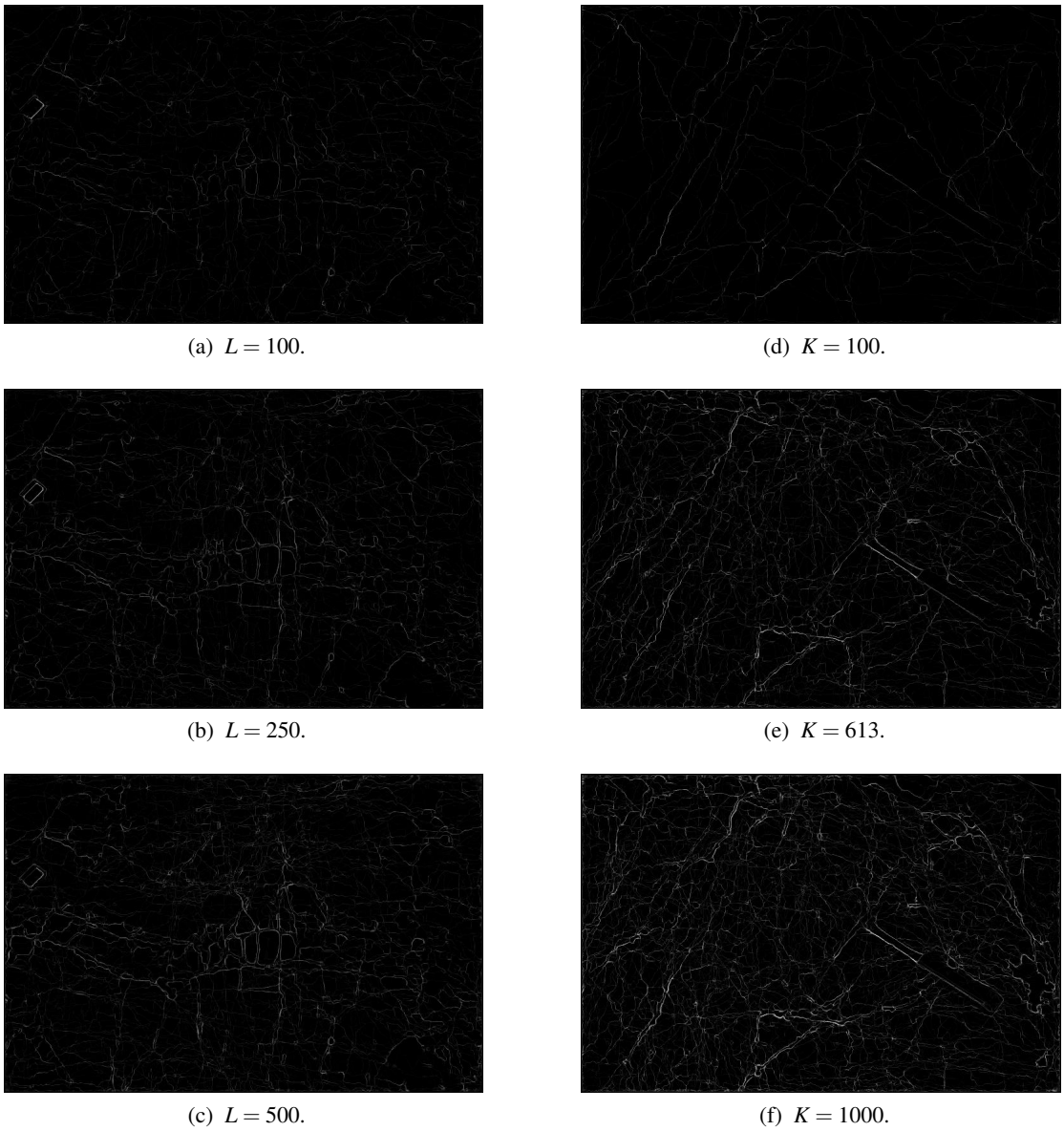
Figure 13: Direction control and total number of rounds.

latter basically means that ants will only move straight forward, completely ignoring the other important transition factors such as pheromone and intensity variation. This is demonstrated in Figure 13a - 13c.

Looking at Figure 13a, it becomes clear that with $r = 0.025$, too little direction control is applied. Results generated by such ants suffers much of the same consequences as when using undirected ants: Small round circles. However, too much direction control seems to be reached already when $r = 0.175$, and leads to perhaps even worse results, as illustrated in Figure 13c. There may not be any unwanted circles anymore, and although the focus is biased towards straight lines it is important to not loose all other edges. After all, not every edge is a completely straight line. Therefore, ants must be able to turn when the pheromone information or intensity variation indicates so. A value of around 0.1 seems to be suitable for r. Figure 13b shows this.

Total Number of Rounds

The difference in the total number of executed construction steps (rounds) is not that crucial. It does not take the ants many iterations to be able to produce a satisfying edge result. The difference between three, five and ten rounds is illustrated in Figure 13d - Figure 13f.



(a) $L = 100$.

(d) $K = 100$.

(b) $L = 250$.

(e) $K = 613$.

(c) $L = 500$.

(f) $K = 1000$.

Figure 14: Total number of movement steps and total number of ants.

Generally it seems like $N = 5$, Figure 13e, is sufficient for detecting the edges in the image. Even three rounds, Figure 13d, produces a fairly good result. However, if $N = 3$, it is possible that some of the edges may become a little too weak. Hence, the risk of missing out on valuable edge information is perhaps a little too high. Though, using as much as ten rounds, Figure 13f, gives only somewhat better highlighted edges as compared to five. Additionally, it does seem to generate a little too much false edges. Therefore, five rounds seems most appropriate.

Total Number of Movement Steps

Too few movement steps generally leads to worse results, in the sense of generating inadequate edge traces. This is closely related to the fact that most edges are rather long, considering the current image size, and therefore consists of a relatively high amount of pixels. For this reason, ants need to walk a decent amount of steps. This is demonstrated in Figure 14a - Figure 14c.

An inadequate edge trace is shown in Figure 14a. It is fairly easy to see that $L = 100$ is not really sufficient for an image size of 752×500. Several edges do not seem to be fully detected by the ants, and some valuable edge information does not show at all. A more suitable amount is shown in Figure 14b,

when $L = 250$. Even as much as $L = 500$ generates adequate results, as illustrated in Figure 14c. One could in fact argue that the latter does seem to contain more valuable edge information. After all, higher values implies that each ant is capable of detecting even more edges each round. On the other hand, it does generate more noise and more false edges as well, considering that ants do need to walk somewhere when moving from one edge to another. Although the difference between the two is noticeable, it is not truly the most decisive factor in the end. Hence, $L = 250$ seems sufficient.

Total Number of Ants

The number of ants is essential. After all, ants are the ones responsible for detecting the edges in the images. Using too few of them simply leads to a poor edge trace because they will not be able to fully cover the whole image. Hence, the total number of ants needs to be big enough to ensure that the edges in the images is being detected. Though overdoing it would suggest that a lot of ants will be walking around and depositing pheromone in edgeless areas of the image. Which in turn results in false edges. Since they all start at random locations there is no way to prevent this from happening. Therefore, the total number of ants also needs to be small enough in order to limit the amount of false edge information. This is demonstrated in Figure 14d - Figure 14f.

As illustrated in Figure 14d, $K = 100$ leads to a poor edge trace. There is a lot of edges not being successfully detected by the ants. Then, Figure 14f illustrates the opposite. $K = 1000$ contains so much edge information that it is almost impossible to distinguish one edge from another. Besides, that many ants are detecting a lot of false edges. The formula used in the studied literature [5, 6], when applying artificial ants to image edge detection, proves to hold. In other words, $K = \lfloor \sqrt{752 \times 500} \rfloor = 613$ seems to be a good number, as shown in Figure 14e.

5.2 Experimental Morphological Edge Detection Results

Experimental results demonstrates the usefulness of mathematical morphology and mathematical set theory in detecting edges in a digital image.

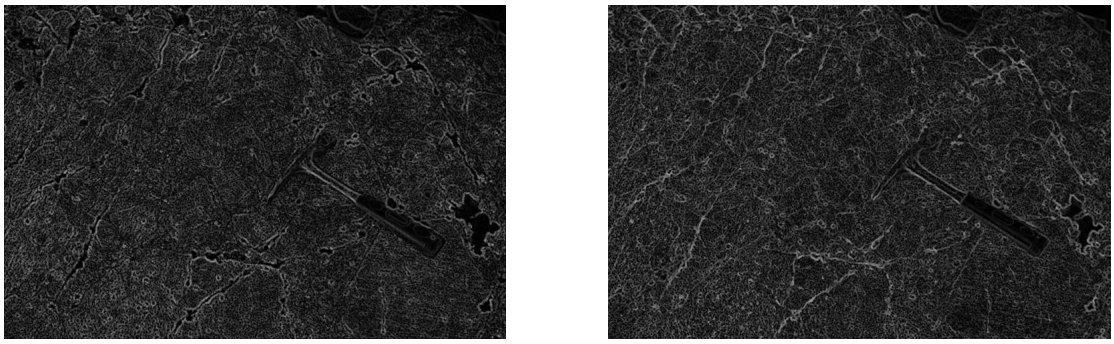
In morphological edge detection, just as with ant edge detection, white lines is an indication of edges in the images. Likewise is a dark pixel an indication of a non edge pixel.

However, as opposed to ant edge detection, there are no parameter values to adjust in morphological edge detection. The results obtained depends on the composition of the morphological edge detector as well as the construction of the structuring element. In the case of erosion residue edge detection, dilation residue edge detection and morphological gradient edge detection, the results are all obtained by performing multiple erosions and dilations of the image, using each of the four structuring elements in succession. That way, each of these detectors generate only a single composite result (per image), utilizing all four structuring elements. For the case of reduced noise morphological gradient edge detection however, the case is a little different. The detector itself is much more complex and contains composite morphological operators. For this reason, and considering the fact that both opening and closing dilates and erodes its resulting set using the same initial structure element, the process of applying several structuring elements in succession becomes cumbersome. Hence, reduced noise morphological edge detection is performed once per structuring element, generating a total of four results (per image).

Erosion and Dilation Residue Edge Detection

Both erosion residue edge detection and dilation residue edge detection gives unsatisfactory results, as illustrated in Figure 15. Using one or the other does not seem to matter that much as they both suffer the same major weakness: Noise. At the same time it is fairly easy to distinguish between the two detectors.

In erosion residue edge detection, Figure 15a, it seems almost like the image edges are shifted (one pixel) outwards. Considering the fact that gray scale erosion is supposed to darken an image, it seems only appropriate that darker objects (cracks) are indeed expanded a little. Hence, transposing their edges



(a) Erosion residue edge detection.

(b) Dilation residue edge detection.

Figure 15: Erosion and dilation residue edge detection.

further out. The opposite effect is shown in Figure 15b, where dilation residue edge detection shifts the edges (one pixel) inwards, closing the gaps and thus brightening the image.

The big issue however, which neither of the two detectors are able to cope with, is noise. In fact, all the noise makes most of the edge traces confusing and hard to follow.

Morphological Gradient Edge Detection

Morphological gradient edge detection is illustrated in Figure 16. A morphological gradient edge detector highlights sharp gray level transitions. As easily seen, that does not produce a fortunate result when it comes to producing clear image edges. As a matter of fact, this is the worst edge trace so far.

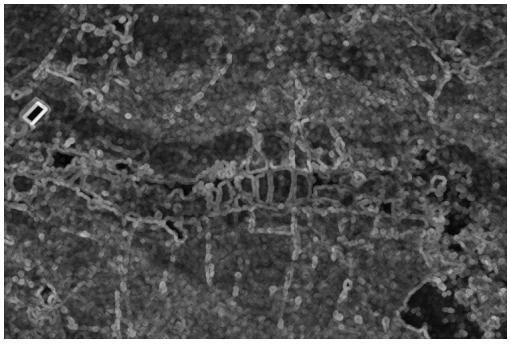


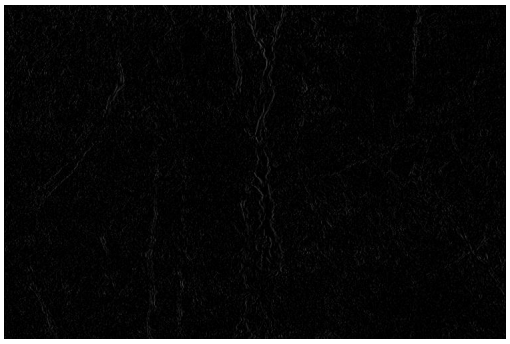
Figure 16: Morphological gradient edge detection.

The total amount of noise is about the same as with the residue detectors, but the visual presentation has become all blurry and obscure. It is possible to follow some of the major edge trends in the image when knowing exactly where to look. However, without any pre-knowledge of the input image and its edges, analyzing a result like this becomes problematic at best. It is quite obvious that a morphological gradient edge detector does not suit the nature of this problem very well.

Reduced Noise Morphological Gradient Edge Detector

Reduced noise morphological gradient edge detector is demonstrated in Figure 17. Results using four 3×3 structuring elements (Table 2) is shown in Figure 17a - 17d.

As compared to the (noisy) morphological gradient edge detector, these results does contain considerable less noise. Furthermore, the results are neither blurry nor unclear. Besides, none of the edges seems to be transposed in any direction as is the case with the residue detectors. There is one big disadvantage though, that makes these results less valuable, and that is the quality of the detected edges. Depending on the orientation of the applied structuring element several edges are not detected at all. Then, the majority of the edges that are detected have a rather fragmented and incomplete edge trace. It looks a little like an incomplete ant edge detection result with additional noise. Even though the amount of noise is less compared to the other morphological edge detection results, it is still considered a decisive amount. After all, it does make it more troublesome to follow the edge trends in the image.



(a) Reduced noise morphological gradient edge detection using a 0 deg structuring element.

(c) Reduced noise morphological gradient edge detection using a 90 deg structuring element.

(b) Reduced noise morphological gradient edge detection using a 45 deg structuring element.

(d) Reduced noise morphological gradient edge detection using a 135 deg structuring element.

Figure 17: Reduced noise morphological gradient edge detector.

Comparing Ant Edge Detection to Morphological Edge Detection

By comparing ant edge detection to morphological edge detection, Figure 18, it becomes quite clear that (directed) artificial ants outperforms mathematical morphology in detecting the edges in the images.

Morphological edge detection, Figure 18c, have the advantage of being close to "real time" edge detection. In other words, it takes the detectors roughly a second to process an image. Ant edge detection takes about seven minutes, hence, morphological edge detection is in fact several hundreds times faster. Additionally, none of the morphological edge detectors generates a false edge frame along the image border, like ants do. Though, all the noise generated by morphological edge detectors is not desired. Besides, artificial ants generates clearer and more complete edge traces, as well as a considerable less amount of noise, as seen in Figure 18b. Since computational times is not considered an issue, this is a much more valuable result.

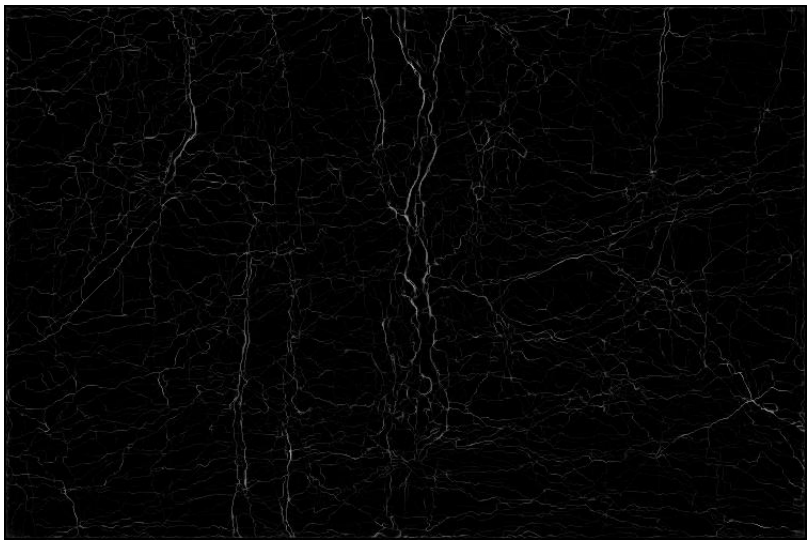
5.3 Experimental Crack Extraction Results

Experimental results demonstrates the practicability of the proposed novel approach in extracting crack information from a digital image.

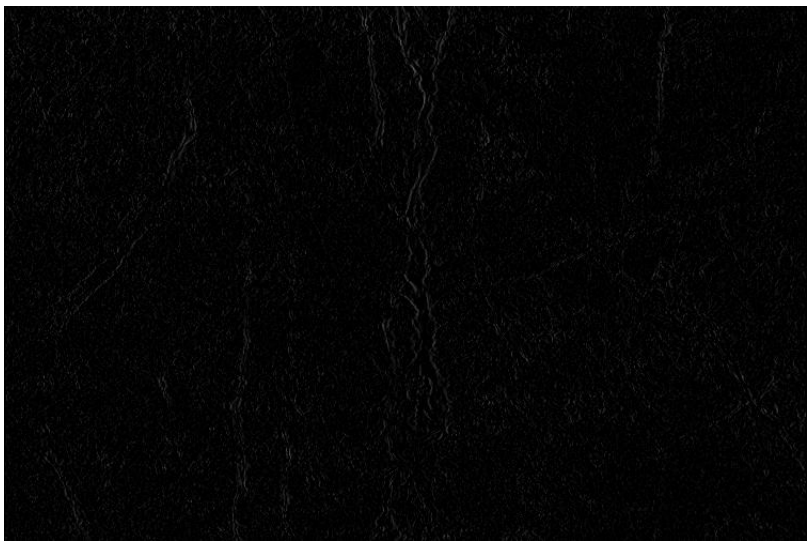
In crack extraction, as opposed to the two edge detection methods where white pixels indicates edges, black pixels is an indication of cracks in the image. In other words, a white pixel now represents a background (non-crack) pixel. Typically, an object (crack) pixel is given a value of "1" (white), while a background (non-crack) pixel is given a value of "0" (black). However, to better distinguish between the results obtained in edge detection and crack extraction, a black pixel indicates a crack pixel.

Note that crack extraction is based on the edge information detected by artificial ants. Direction controlled ants clearly outperformed mathematical morphology in detecting edges in the images. Hence, no morphological edge detection results are used to extract cracks. For this reason, the quality of the extracted crack information is highly dependent on the work done by the ant collective.

(a) Original crater image.

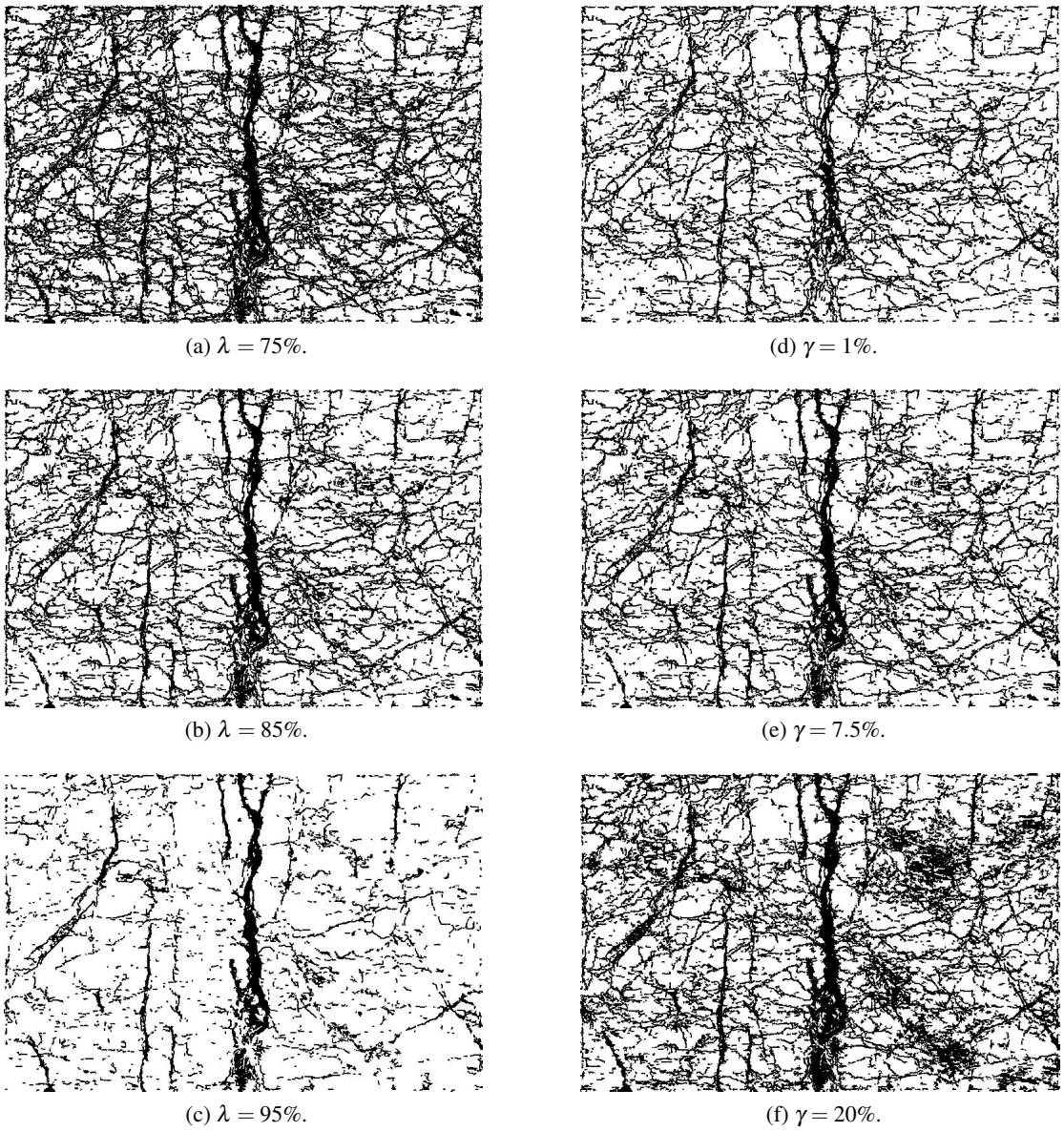


(b) Ant edge detection.



(c) Morphological edge detection.

Figure 18: Ant edge detection vs. morphological edge detection.



(a) $\lambda = 75\%$. (d) $\gamma = 1\%$.

(b) $\lambda = 85\%$. (e) $\gamma = 7.5\%$.

(c) $\lambda = 95\%$. (f) $\gamma = 20\%$.

Figure 19: Ant edge and gray scale thresholding using percentiles.

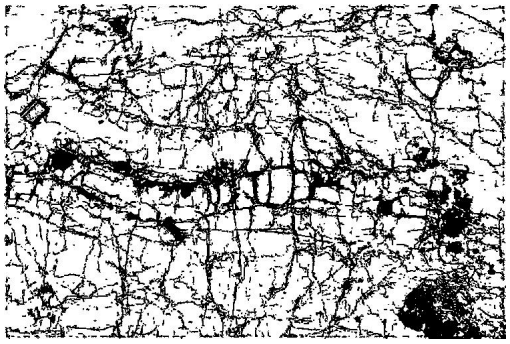
Thresholding using Percentiles

When reducing the gray scale ant edge image to a binary crack image, it is important to keep in mind the project goal. Generally speaking, it is considered more valuable to generate a result showing "a little too much" information, rather than thresholding it all away. More specifically, a result clearly showing cracks and linear trends in the image, at the expense of some noise, is better than a result showing only the biggest and most significant cracks. Therefore, thresholding with appropriate percentiles becomes the most decisive factor when extracting valuable crack information. In fact, the higher the value of the given percentiles, the less cracks will be extracted. Then, thresholding by too low percentiles leads to considerably more noise and unwanted information. Figure 19 demonstrates this. Figure 19a - 19c illustrates the effect of different edge percentiles used when reducing the gray scale edge information to binary crack information. Figure 19d - 19f illustrates the effect of different gray scale percentiles used when finding a sufficiently small value, based on the values in the gray scale image, to use as the definition of a crack pixel.

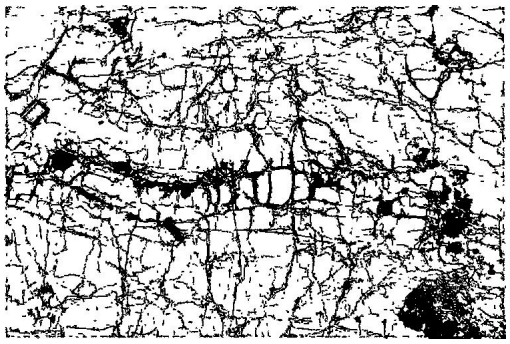
Judging by Figure 19c, a 95% edge percentile seems too strict. That way, only edge pixels having a value among the top five percent of the total edge data is considered a valid crack pixel. This leads to the

bigger and more significant cracks becoming extracted pretty well. However, there is clearly a lot of the minor crack trends in the image that have been thresholded away simply because their initial edge value is not high enough to survive the thresholding process. More specifically, the ants have not been depositing enough pheromone on these edge pixels during edge detection. This is unfortunate considering the fact that such crack trends is considered valuable information. Though, by decreasing the edge percentile to 75%, as illustrated in Figure 19a, a lot more, and in fact too much, information is being extracted. This is the result of being to lenient towards the ants by using a threshold value so low that even pixels not really representing edges in the images, but rather is being traversed as the ants are searching for more promising pixels, survives the thresholding process. More specifically, it requires just a small amount of pheromone in order to be marked as a crack pixel. This is clearly not optimal. Using a 85% edge percentile seems more appropriate, as shown in Figure 19b. There is still a sufficient amount of noise and false cracks present due to ants walking these paths during edge detection. However, several minor cracks are extracted and it is relatively easy to follow trends in the image.

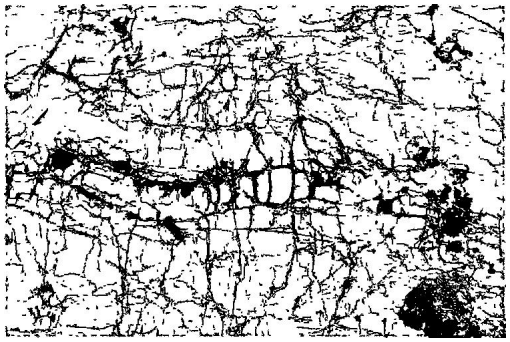
The same reasoning applies to gray scale percentiles, but with the opposite effect. As shown in Figure 19d, requiring a crack pixel to have a gray scale value among the lowest 1% of the total gray scale data is too strict. This is easily seen as several parts of the big center crack is not being filled. In fact, lots of true crack pixels are not being marked as crack pixels. Then, $\gamma = 20\%$ results in more cracks than appropriate, as shown in Figure 19f. As illustrated in Figure 19e, $\gamma = 7.5\%$ seems to be a good value.



(a) No noise filter.



(b) $\varepsilon = 0.25$.



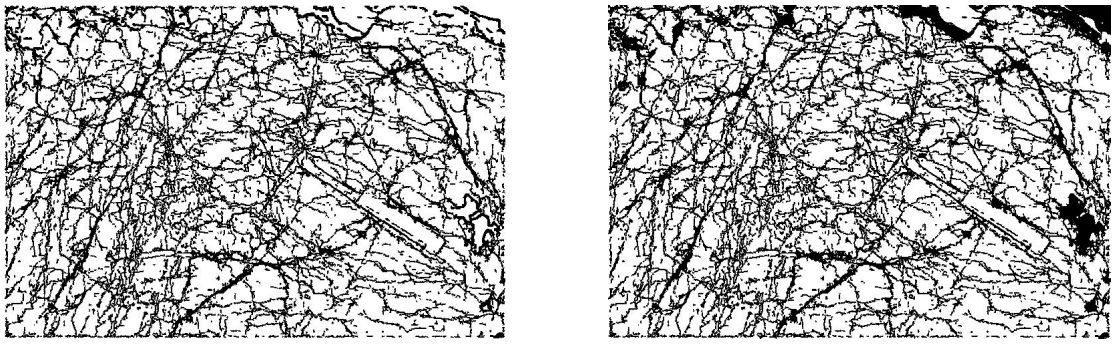
(c) $\varepsilon = 0.75$.

Figure 20: Noise filtering by eccentricity.

Noise Filtering by Eccentricity

The noise filter is responsible for removing randomly occurring pixels, as a result of thresholding the ant edge image to a binary crack image. Regions of noise often have a very small eccentricity ratio (circular components), which can be removed by the filter. Also, the edge frame generated by border ants should be removed. Figure 20 demonstrates the difference between no noise filter (Figure 20a), a suitable noise filter (Figure 20b), and too much noise filtering (Figure 20c).

By not using a noise filter it becomes quite easy to see the amount of noise generated during the thresholding process. The result suffers from lots of smaller pixel groups (one, two, three connected pixels) occurring seemingly randomly over the entire image. These are rarely a part of an actual crack. Hence, they are considered as noise and should consequently be removed. Though, overdoing the noise filter leads to filtering away more than just noise. As seen by comparing the two filters, entire cracks are actually missing in the latter case. This is obviously bad and indicates that removing all connected components having an eccentricity ratio below 0.75 is not appropriate. A noise filter capable of removing small pixel groups, and at the same time preserving the crack trends in the image, seems to suit a value of 0.25.



(a) $N = 2$. (b) $N = 25$.

Figure 21: Total number of crack filling iterations.

Total Number of Crack Filling Iterations

The number of iterations affects the extent of the crack filling process, as demonstrated in Figure 21. After the first iteration all crack outlines are extended by one pixel into the crack itself. Hence, every crack now has a more or less two pixel wide outline. For this reason, cracks only a few pixels wide becomes completely filled after only a few iterations (Figure 21a). Then, with every new iteration, the gaps continues to fill by one pixel all around their perimeter. That way, even the widest cracks becomes completely filled after a sufficient amount of iterations. To make sure every crack is eventually filled, even the widest cracks, as much as 25 iterations is used (Figure 21b). This might be considered a little extreme. Though, considering the fact that each iteration completes within seconds, as well as every additional iteration taking less time than the previous, it is not considered a real issue.

5.4 Experimental Test Results

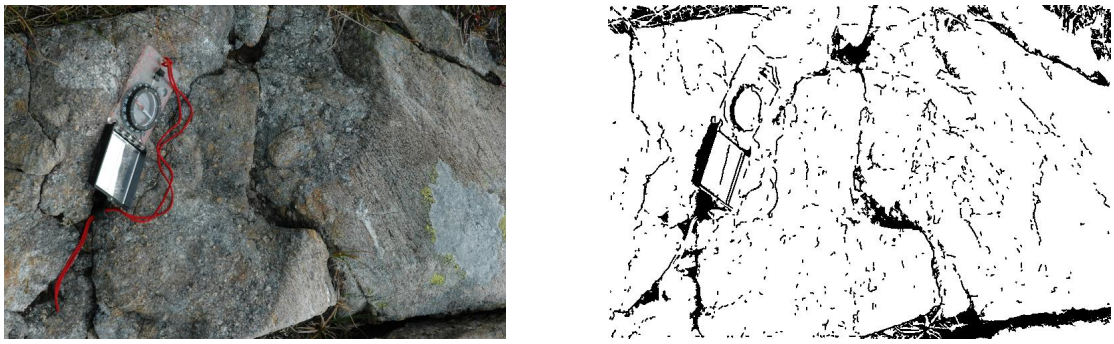
Experimental test results (Appendix B) demonstrates the usefulness of the proposed approach in extracting cracks from images taken from the Ritland meteorite impact crater. Experiments are carried out on a total of sixteen test images, independent of the previous training images, using the appropriate parameter values. The reason for not using a larger selection of test images is the similarity in the original images. The sixteen images selected are those that best represents the diversity among the available images.

The goal of this project is to generate (binary) images showing the cracks in the crater rocks. Although some of the test results obtained are better than others, all results are showing a satisfactory amount of extracted crack information.

The results are influenced by feedback from a geologist. Therefore, as much cracks as possible, both big and small, is attempted extracted in order to better determine the various crack trends in the images. Hence, noise and false cracks is also to be expected. Besides, crack extraction is based on ant edge detection. Hence, the quality of the extracted cracks is dependent on the work carried out by a direction controlled ant collective. In fact, a majority of the false cracks are a (direct) result of ant edge detection, where ants deposit pheromone along false crack outlines as they search for true edges in the images.

It is fairly easy to see that in images where the crack outlines are originally vague (Figure 38 on page 48), the ants have some trouble with detecting edges properly, hence, generating a satisfying edge trace to use in the crack filling process. The reason for this is that vague crack outlines suffers from small intensity variations, meaning that edge pixels have a gray scale intensity value almost as small as its surrounding non-edge pixels. This naturally makes the edges less attractive to ants. In images with initially clear outlines however, it is easy to determine the various crack trends generated in the result images (Figure 28 on page 38).

Ants naturally have trouble with distinguishing between edges belonging to a crack and edges belonging to a moss patch, as they are both identified by pixels where the intensity variation is high. Therefore, ants are only able to treat a moss patch as a crack, hence, detect its edges. Besides, both cracks and



(a) Original image.

(b) Significant cracks extracted.

Figure 22: Extracting only the most significant cracks.

moss patches are identified by dark pixels, which affects the crack filling process. In other words, moss patches will be marked and colored as cracks in the resulting images (Figure 33 on page 43).

In order to generate a result showing only the most significant cracks, increasing the applied edge percentile is useful. This leads to the amount of false crack information becoming considerably less. Though, doing so naturally results in the loss of valuable minor crack trends as well. This is demonstrated in Figure 22, where (only) the edge percentile is increased from 85% to 97.5%.

6 Conclusion

A new direction controlled ant colony optimization method, building on improvements introduced in the ant colony system algorithm, have been implemented and tested in image edge detection. Experiments show the superiority of the new method, as compared to mathematical morphology, in detecting edges in images of rocks taken from a meteorite impact crater. Furthermore, a novel approach for extracting cracks from the images, by making use of the edge information obtained by the new direction controlled ants, have been developed and tested. Experiments show the success of the algorithm in extracting valuable crack information from the impact crater images. The obtained end results is meant to assist geologists when studying crack trends in images of crater rocks. The results are highly dependent on the actual nature of the images and is therefore likely to vary in other applications.

7 Future Work

As a continuation of work on the subject, it is possible to further examine how the quality of the extracted information is affected by different parameter value combinations. It could be interesting to assign ants different sensitivity levels against pheromone, making some ants more sensitive whilst others less. Besides, fine tuning the new direction control feature by looking at the ants latest steps, in stead of their starting position, may be profitable. There is also reason to believe that if the smaller and least significant cracks could be darkened, by improved pre-processing, it would be beneficial. Then, using a more sophisticated method for choosing the threshold values, as well as applying a more appropriate noise filter, will have a positive effect on the end results. Furthermore, the proposed approach would be enriched by a rose diagram feature that shows crack orientation. That way, the cracks' direction can be automatically measured and quantified. It is also likely profitable to gain even better understanding of the inner nature of meteorite impact craters. Hence, getting more insight and knowledge as to what a geologist actually looks for when analyzing a meteorite impact crater. Besides, providing geologists the possibility to manually remove and add cracks (an interactive program) from the images being processed is thought to be a most valuable addition.

Appendix A

Training Images

(a) Training image: 8372.

(b) Training image: 8382.

(c) Training image: 8397.

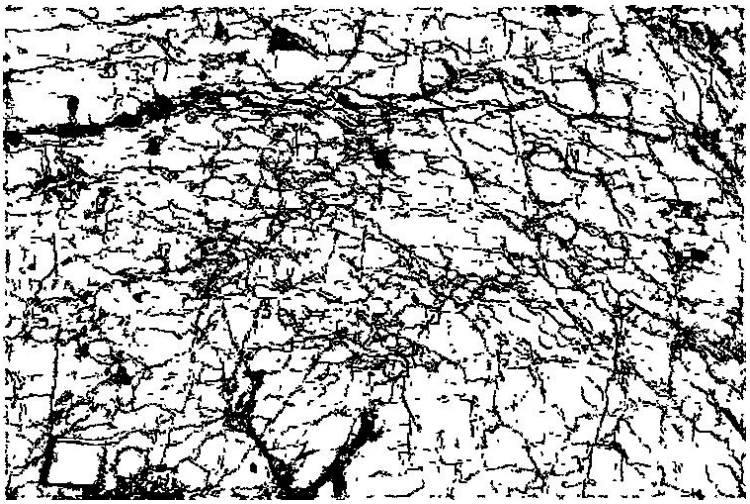
Figure 23: Training images.

Appendix B

Test Image: 8358

(a) 8358 original.

(b) 8358 ant edge detection.

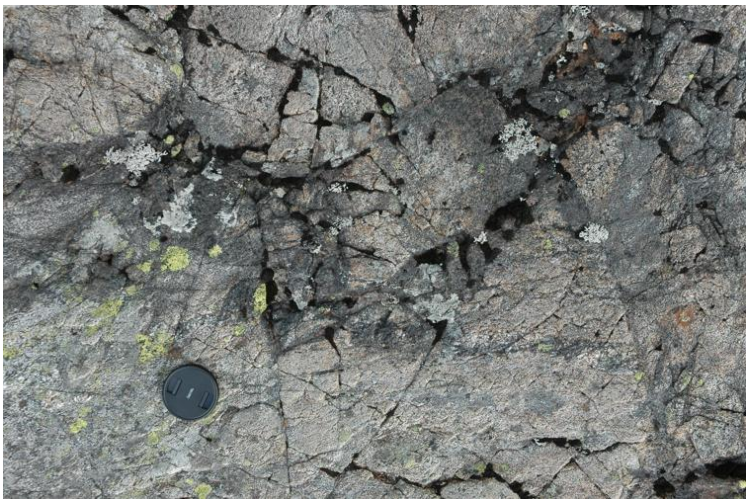


(c) 8358 crack extraction.

Figure 24: Test image: 8358.

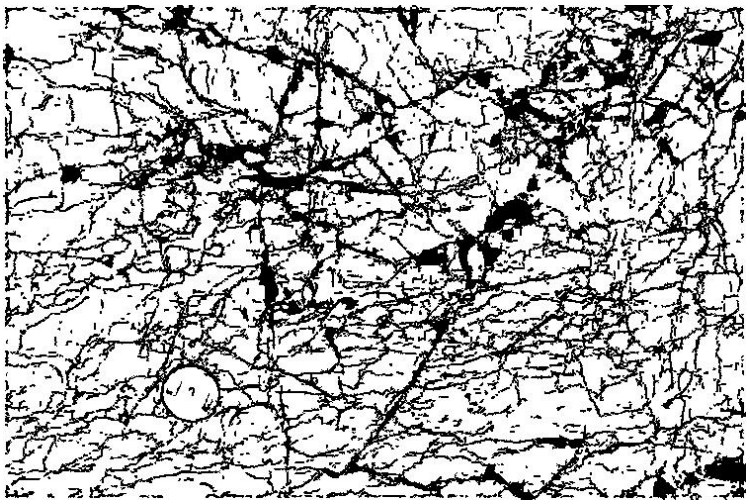
Appendix B

Test Image: 8367



(a) 8367 original.

(b) 8367 ant edge detection.



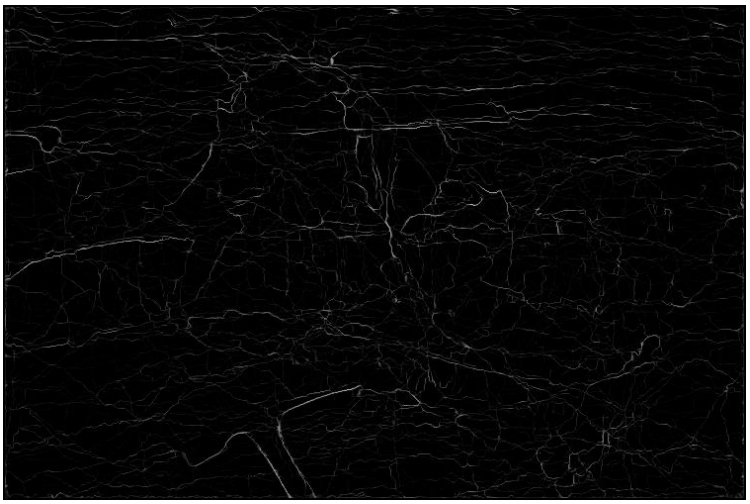
(c) 8367 crack extraction.

Figure 25: Test image: 8367.

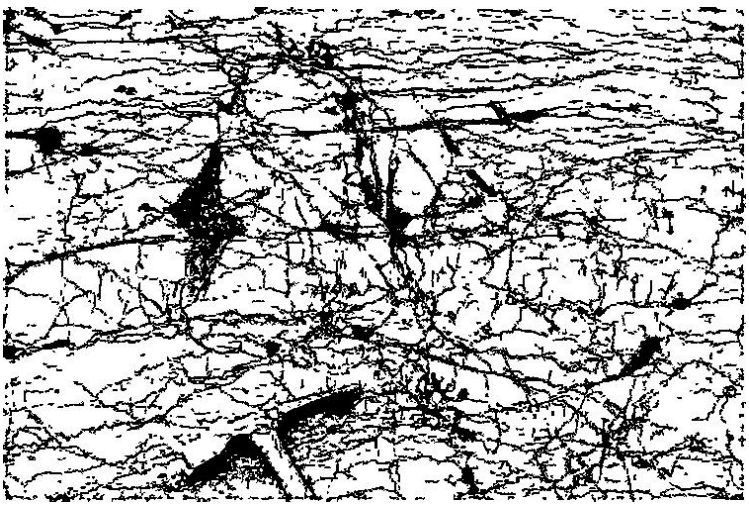
Appendix B

Test Image: 8371

(a) 8371 original.



(b) 8371 ant edge detection.



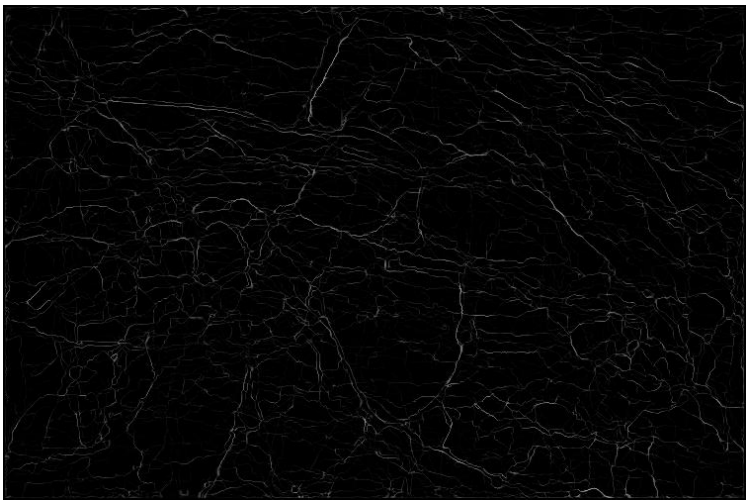
(c) 8371 crack extraction.

Figure 26: Test image: 8371.

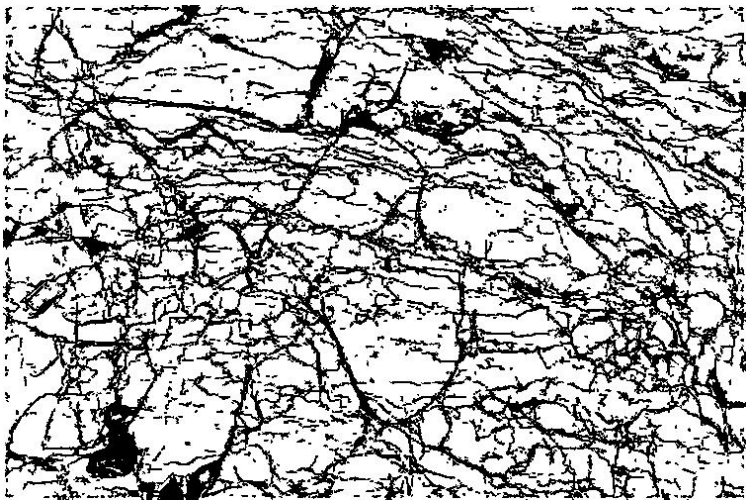
Appendix B

Test Image: 8377

(a) 8377 original.



(b) 8377 ant edge detection.



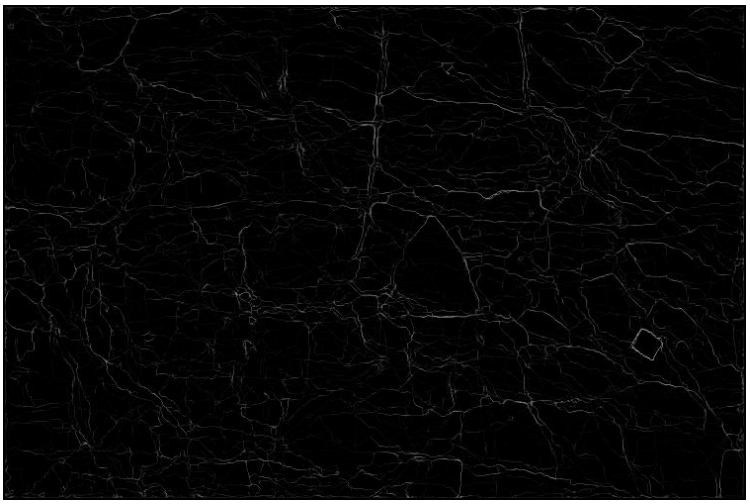
(c) 8377 crack extraction.

Figure 27: Test image: 8377.

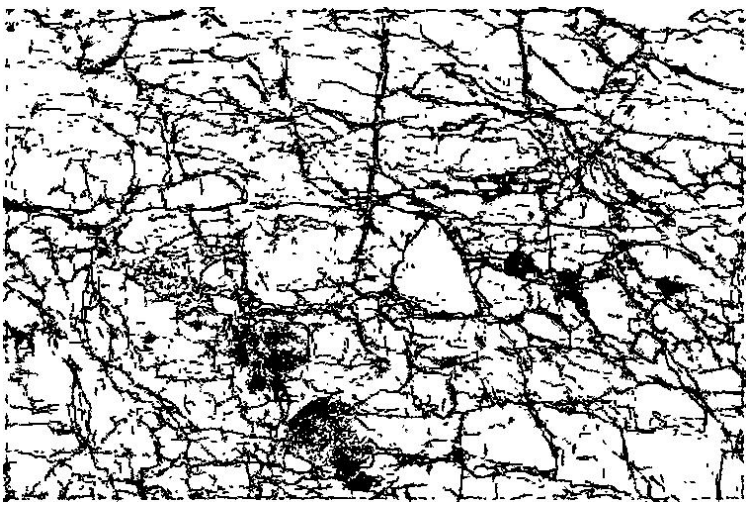
Appendix B

Test Image: 8378

(a) 8378 original.



(b) 8378 ant edge detection.



(c) 8378 crack extraction.

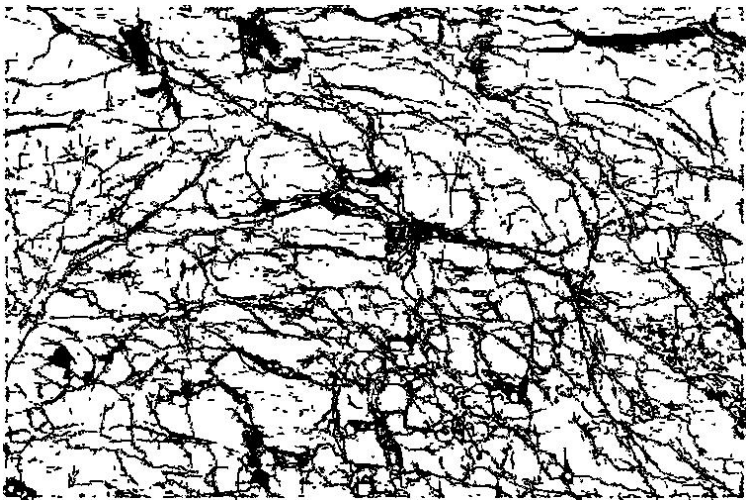
Figure 28: Test image: 8378.

Appendix B

Test Image: 8379

(a) 8379 original.

(b) 8379 ant edge detection.



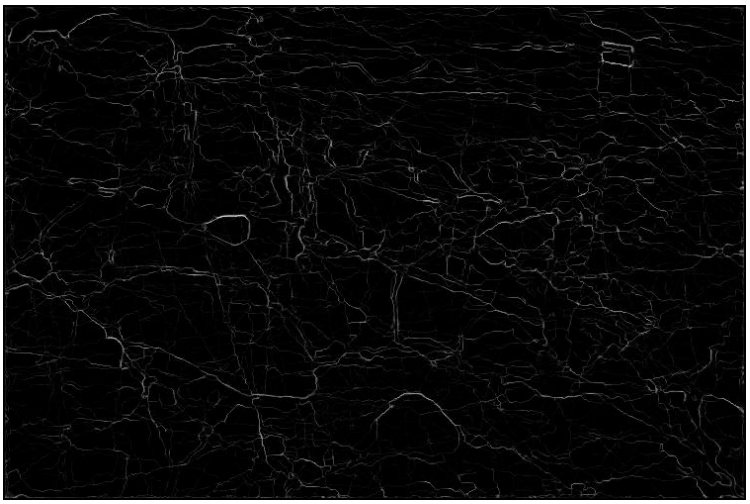
(c) 8379 crack extraction.

Figure 29: Test image: 8379.

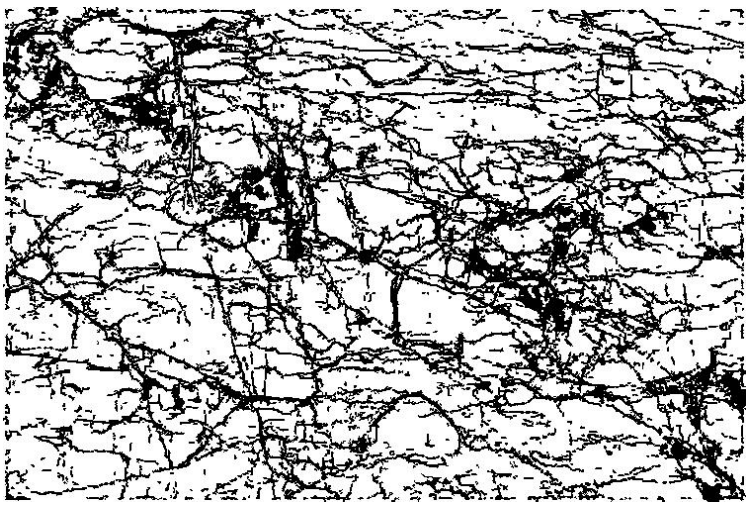
Appendix B

Test Image: 8380

(a) 8380 original.



(b) 8380 ant edge detection.

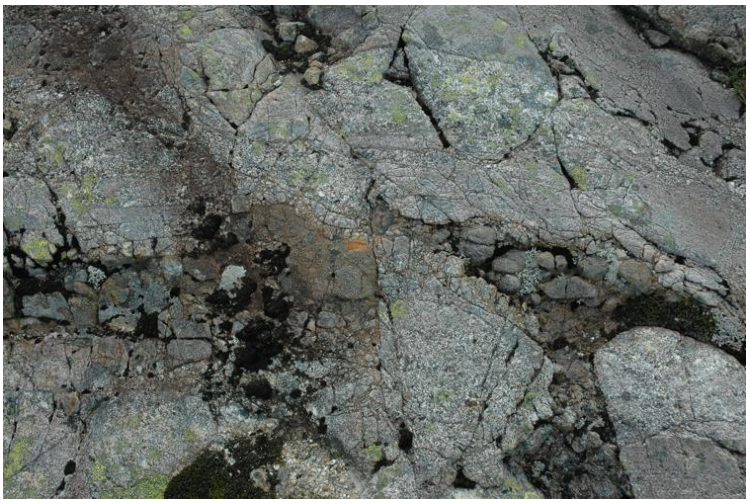


(c) 8380 crack extraction.

Figure 30: Test image: 8380.

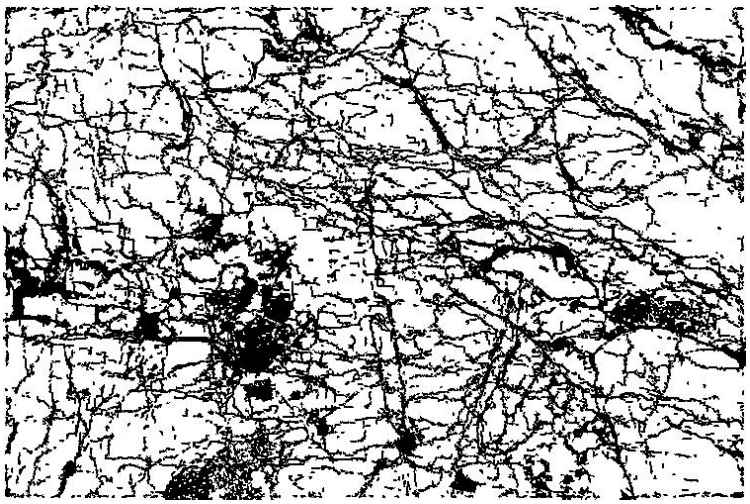
Appendix B

Test Image: 8383



(a) 8383 original.

(b) 8383 ant edge detection.



(c) 8383 crack extraction.

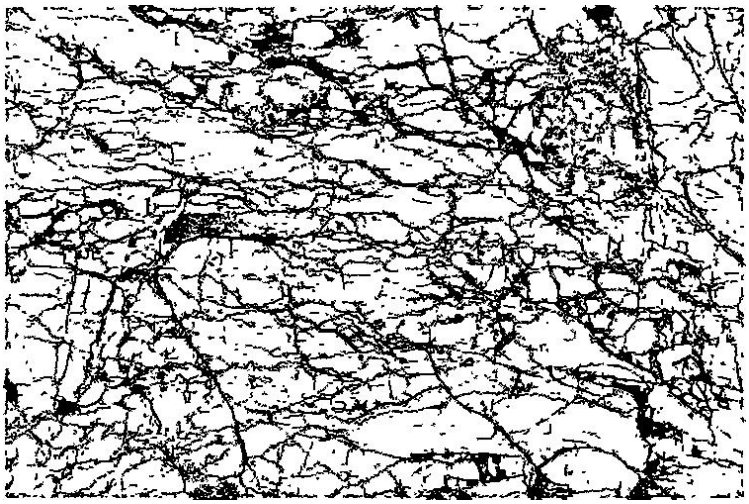
Figure 31: Test image: 8383.

Appendix B

Test Image: 8384

(a) 8384 original.

(b) 8384 ant edge detection.

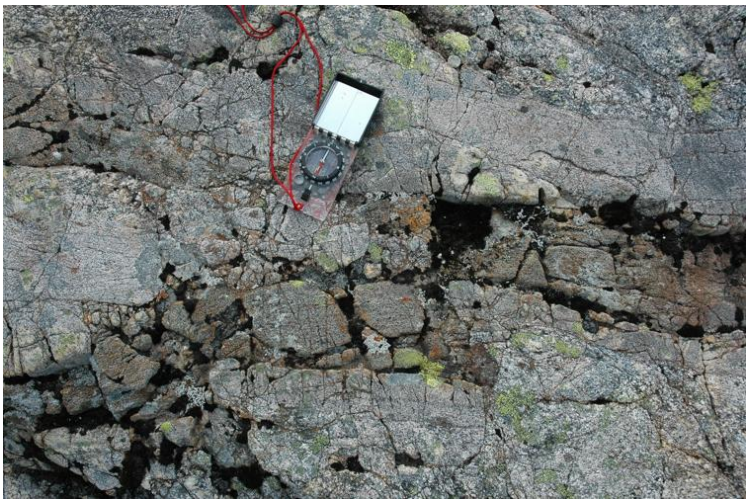


(c) 8384 crack extraction.

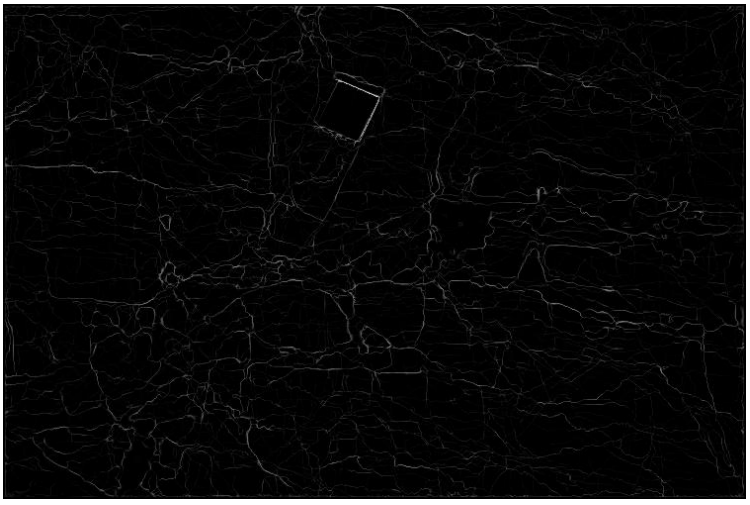
Figure 32: Test image: 8384.

Appendix B

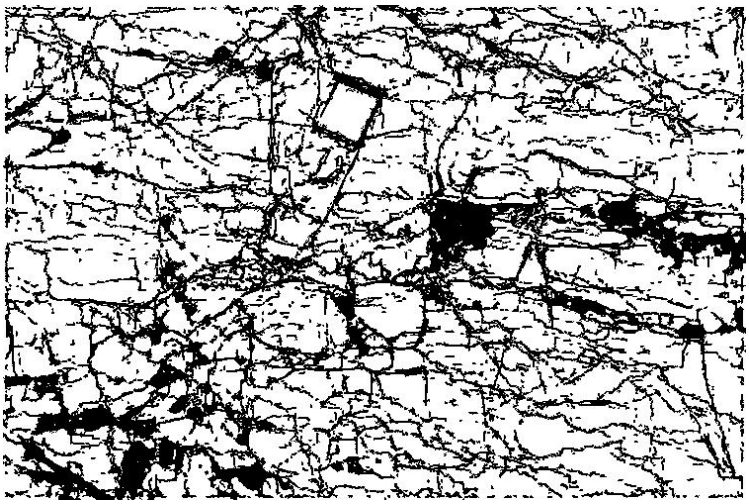
Test Image: 8385



(a) 8385 original.



(b) 8385 ant edge detection.



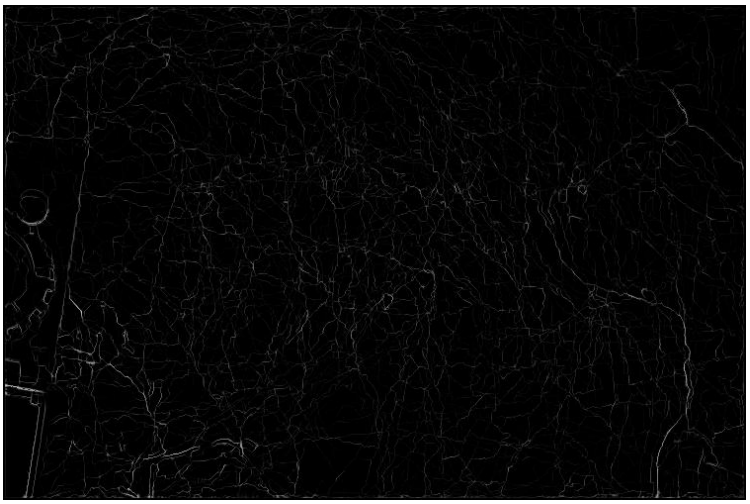
(c) 8385 crack extraction.

Figure 33: Test image: 8385.

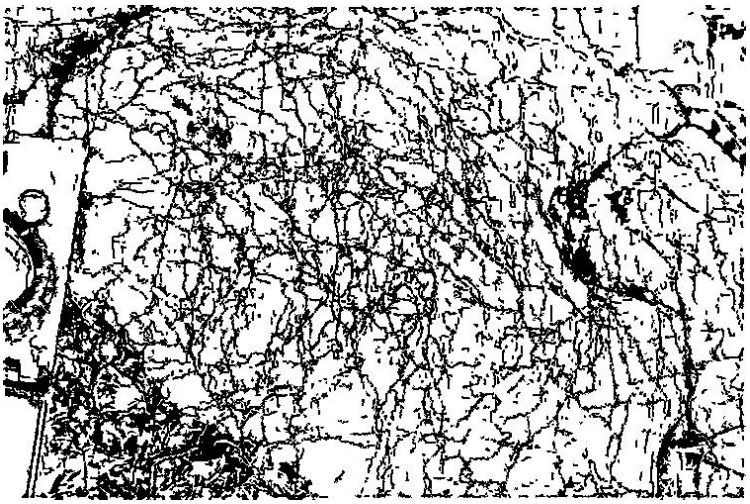
Appendix B

Test Image: 8387

(a) 8387 original.



(b) 8387 ant edge detection.



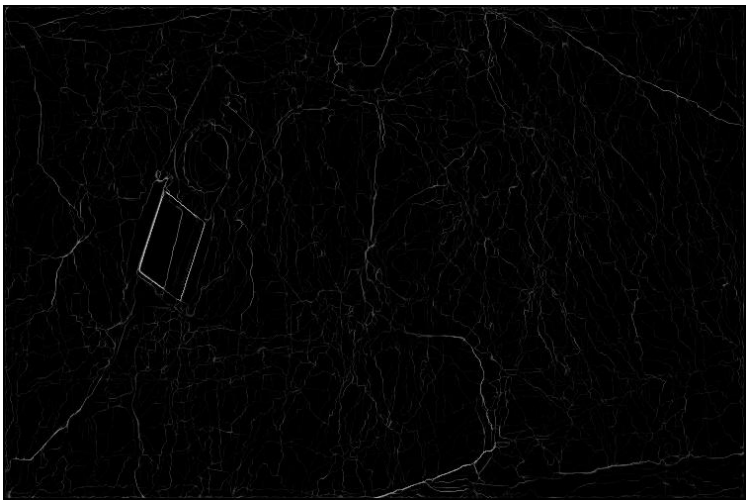
(c) 8387 crack extraction.

Figure 34: Test image: 8387.

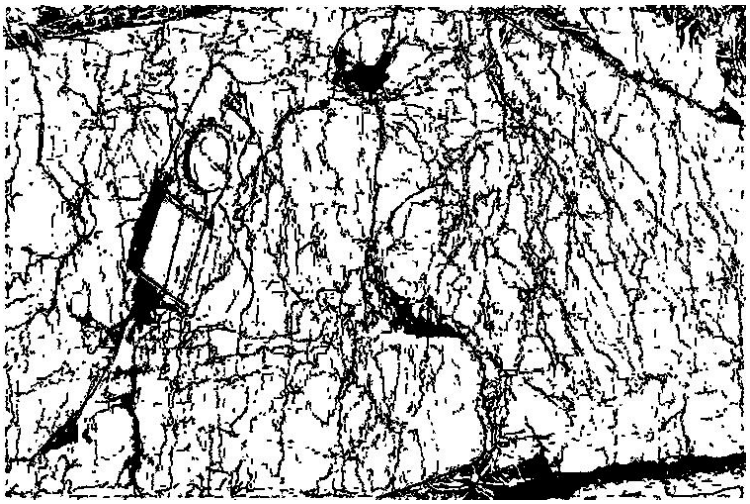
Appendix B

Test Image: 8388

(a) 8388 original.



(b) 8388 ant edge detection.



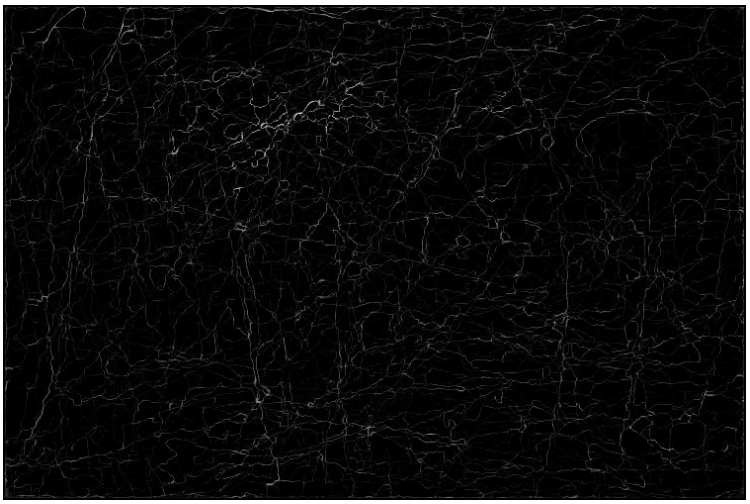
(c) 8388 crack extraction.

Figure 35: Test image: 8388.

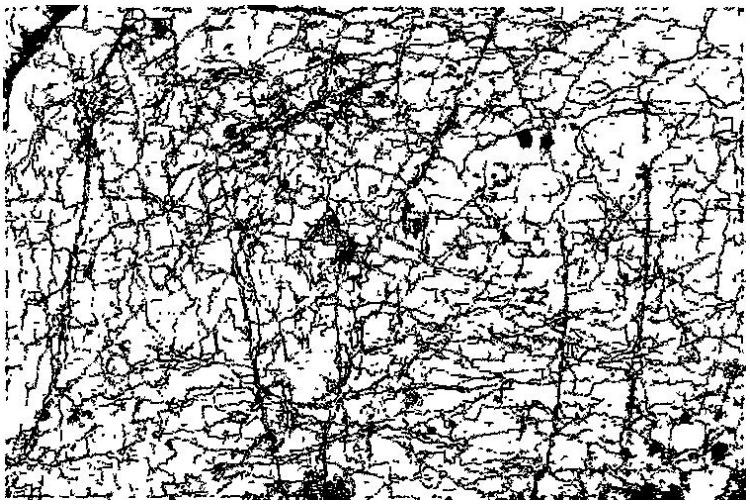
Appendix B

Test Image: 8393

(a) 8393 original.



(b) 8393 ant edge detection.



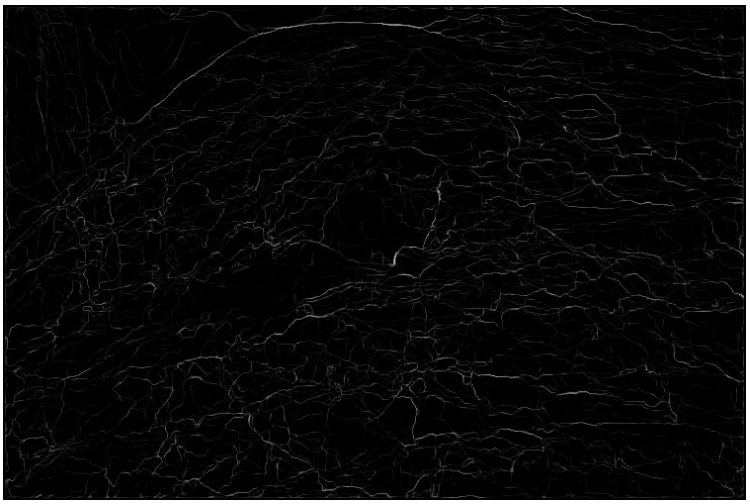
(c) 8393 crack extraction.

Figure 36: Test image: 8393.

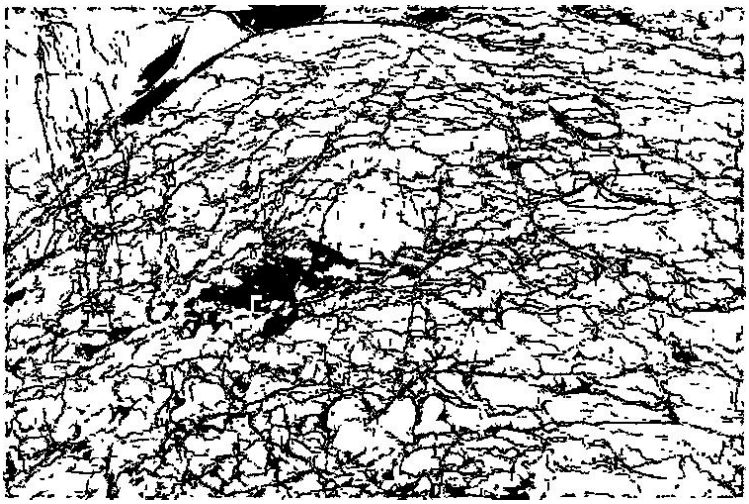
Appendix B

Test Image: 8394

(a) 8394 original.



(b) 8394 ant edge detection.



(c) 8394 crack extraction.

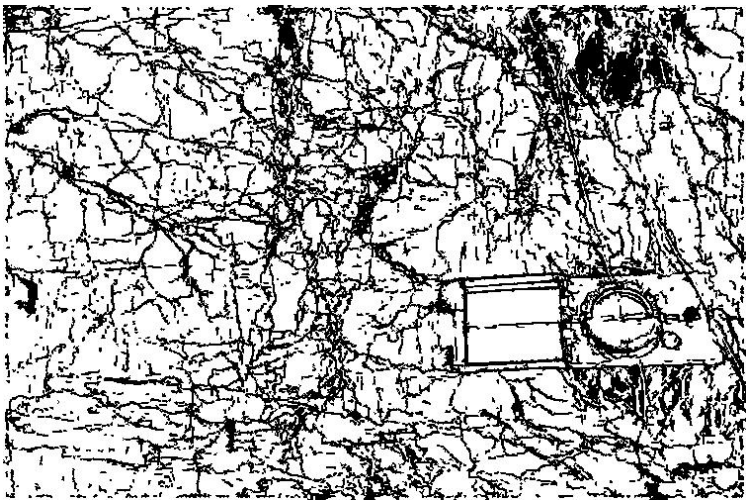
Figure 37: Test image: 8394.

Appendix B

Test Image: 8398

(a) 8398 original.

(b) 8398 ant edge detection.



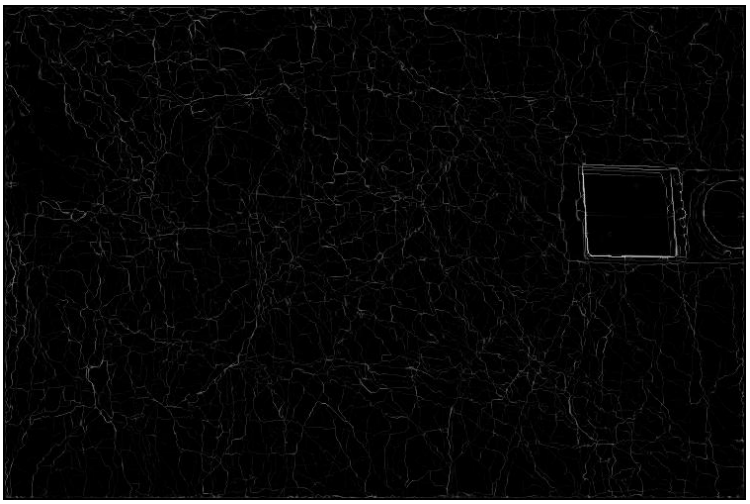
(c) 8398 crack extraction.

Figure 38: Test image: 8398.

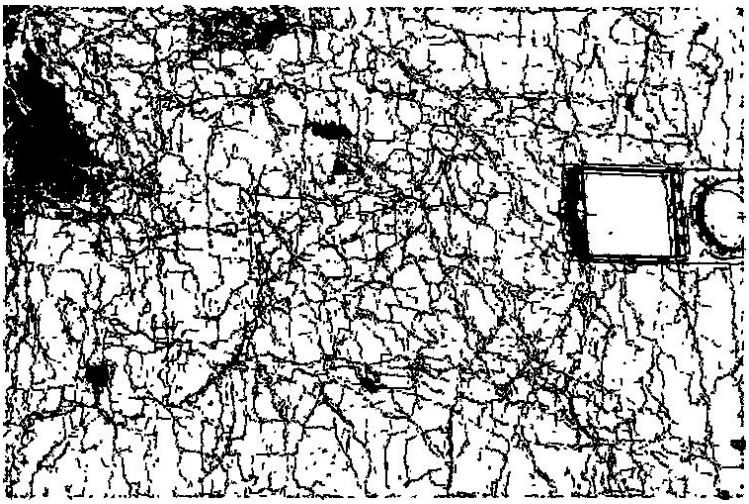
Appendix B

Test Image: 8399

(a) 8399 original.



(b) 8399 ant edge detection.



(c) 8399 crack extraction.

Figure 39: Test image: 8399.

Appendix C

Matlab Source Code

Image Reading, Image Processing and Image Writing (IRIPIW.m)

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                               %
%      Image Reading, Image Processing and Image Writing (IRIPIW.m)             %
%                                                                               %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear;
close all;

% Image Reading Variables
directory = 'workspace/';           % Current Working Directory
files = dir([directory '*.jpg']);   % All (jpg) Image Files in workspace
I = cell(numel(files), 1);          % Cell Array of Images

% Image Processing Variables
gauss = fspecial('gaussian',[5 5]); % Gaussian Blur Filter

% Image Reading, Image Processing and Image Writing
for idx = 1:numel(I); % Images

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %                    Image Reading and Pre-Processing                         %
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    % Read the Original Image: 3008x2000 Pixels
    I{idx} = double(imread(files(idx).name))./255;

    % Resize the Image: 752x500 Pixels (Aspect Ratio Maintained (NaN))
    I{idx} = imresize(I{idx}, [500 NaN]);
    imwrite(I{idx}, [strrep(files(idx).name, '.jpg', '') '_ORIG.jpg'], 'jpg');

    % Convert the Image to Gray Scale
    I{idx} = rgb2gray(I{idx});

    % Apply Gaussian Blur (Reduce Image Noise)
    I{idx} = imfilter(I{idx},gauss);
    imwrite(I{idx}, [strrep(files(idx).name, '.jpg', '') '_GRAY.jpg'], 'jpg');

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %                              Image Processing                               %
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    % Ant Edge Detection
    N = [5];        % Number of Rounds
    L = [250];      % Number of Movement Steps
    K = [613];      % Number of Ants
    R = [0.1];      % Influence of Direction Control
    for n = 1:numel(N);
        for l = 1:numel(L);
            for k = 1:numel(K);
                for r = 1:numel(R);
                    % Direction Controlled Ant Colony Optimization (DCACO)
                    [T] = DCACO(I{idx}, N(n), L(l), K(k), R(r));
                    imwrite(T, [strrep(files(idx).name, '.jpg', '') '_EDGE(' num2str(N(n)) '↩
                        -' num2str(L(l)) '-' num2str(K(k)) '-' num2str(R(r)) ').jpg'], 'jpg'↩
                        );
                end % for r
            end % for k
        end % for l
    end % for n

    % Morphological Edge Detection
    [ERED DRED MGED RNMGED1 RNMGED2 RNMGED3 RNMGED4] = MIP(I{idx});
    imwrite(ERED,   [strrep(files(idx).name, '.jpg', '') '_MIP(ERED).jpg'], 'jpg');
    imwrite(DRED,   [strrep(files(idx).name, '.jpg', '') '_MIP(DRED).jpg'], 'jpg');
```

```
    imwrite(MGED,    [strrep(files(idx).name, '.jpg', '') '_MIP(MGED).jpg'], 'jpg');
    imwrite(RNMGED1,[strrep(files(idx).name, '.jpg', '') '_MIP(RNMGED1).jpg'], 'jpg');
    imwrite(RNMGED2,[strrep(files(idx).name, '.jpg', '') '_MIP(RNMGED2).jpg'], 'jpg');
    imwrite(RNMGED3,[strrep(files(idx).name, '.jpg', '') '_MIP(RNMGED3).jpg'], 'jpg');
    imwrite(RNMGED4,[strrep(files(idx).name, '.jpg', '') '_MIP(RNMGED4).jpg'], 'jpg');

    % Crack Extraction
    N  = [25];      % Number of Iterations
    EP = [85];      % Edge Percentile (> % Edge)
    GP = [7.5];     % Gray Percentile (< % Gray)
    E  = [0.25];    % Eccentricity
    for n = 1:numel(N);
        for ep = 1:numel(EP);
            for gp = 1:numel(GP);
                for e = 1:numel(E);
                    % Crack Closing (CC)
                    [C] = CC(I{idx}, T, N(n), EP(ep), GP(gp), E(e));
                    imwrite(C, [strrep(files(idx).name, '.jpg', '') '_CRACK(' num2str(N(n)) ↩
                        '-' num2str(EP(ep)) '-' num2str(GP(gp)) '-' num2str(E(e)) ').jpg'], ↩
                        'jpg');
                end % for r
            end % for k
        end % for l
    end % for n

end % end for idx

% end IRIPIW.m
```

Direction Controlled Ant Colony Optimization (DCACO.m)

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                               %
%          Direction Controlled Ant Colony Optimization (DCACO.m)               %
%                                                                               %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [T] = DCACO(I, N, L, K, r)
% T = Output Final Pheromone Matrix
% I = Input Image
% N = Input Total Number of Rounds
% L = Input Total Number of Movement Steps
% K = Input Total Number of Ants
% r = Input Influence of Direction Control

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                       DCACO (1. Initialization Phase)                         %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% MISC
[rows cols] = size(I);              % Matrix Dimensions

% Pheromone: T
T_init = 0.00000001;                % Initial Pheromone
T(1:rows,1:cols) = T_init;          % Pheromone Matrix
T0 = T;                             % Initial Pheromone Matrix

% Local Intensity Variation Matrix (The Clique): Vc
for i = 1:rows;
    for j = 1:cols;
        % Local Intensity Variation at The Two Outermost Pixels
        if ( (i<3) || (j<3) || (i>(rows-2)) || (j>(cols-2)) )
            Vc(i,j) = T_init;
        % Local Intensity Variation at The Clique
        else
            Vc(i,j) = abs(I(i-2,j  )-I(i+2,j  )) + abs(I(i-1,j  )-I(i+1,j  )) ...
                    + abs(I(i  ,j-2)-I(i  ,j+2)) + abs(I(i  ,j-1)-I(i  ,j+1)) ...
                    + abs(I(i-2,j-2)-I(i+2,j+2)) + abs(I(i+2,j-2)-I(i-2,j+2)) ...
                    + abs(I(i-1,j-1)-I(i+1,j+1)) + abs(I(i+1,j-1)-I(i-1,j+1)) ...
```

```
                    + abs(I(i-2,j-1)-I(i+2,j+1)) + abs(I(i+2,j-1)-I(i-2,j+1)) ...
                    + abs(I(i-1,j-2)-I(i+1,j+2)) + abs(I(i+1,j-2)-I(i-1,j+2));
            % Clique Check: Clique Sum Equals Zero (0)
            if (Vc(i,j) == 0)
                Vc(i,j) = T_init;
            end % if
        end % if else
    end % for
end % for

% Maximum Intensity Variation: Vmax
Vmax = max(Vc(:));

% Normalized Intensity Variation Matrix (The Heuristic Information): H
H(1:rows,1:cols) = Vc(:,:)./Vmax;

% Polar Matrix (Direction Control): pol
pol = [(5*pi)/4   (3*pi)/2    (7*pi)/4;
          pi          0           0   ;
       (3*pi)/4     pi/2        pi/4 ];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                          DCACO (2. Construction Phase)                       %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

fprintf('Ants at work...\n');

% MISC
rand('state', sum(clock));          % Random Seed Uses Time (Clock)

% ACO Spesific Parameters
%N = 5;                             % Number of Rounds
%L = 250;                           % Number of Movement Steps
%K = floor(sqrt(rows*cols));        % Number of Ants
a = 1.0;                            % Influence of Pheromone
b = 1.0;                            % Influence of Heuristics
evop = 0.1;                         % Evaporation (pr. Round)
w = 0.05;                           % Decay (pr. Ant)

% Direction Control Spesific Parameters
%r = 0.1;                           % Influence of Direction Control

% Ant Tracking
antDeadlock = zeros([K N]);         % Ant Deadlocks
antOnBorder = zeros([K N]);         % Ant on Image Borders

% Ant Movement
tCon=tic;
for n = 1:N; % Rounds
    fprintf('\n');
    display(['ROUND: ' num2str(n)]);
    % Ants Random (Round) Start Positions (No Ants Start on Image Borders)
    for k = 1:K;
        antTour{k,n}(1,1:2) = [round(2+(rows-3).*rand(1,1)), round(2+(cols-3).*rand(1,1))];
    end % for k
    % Ants (Round) Deadlock and Border Counter
    deadlockCounter = 0;
    borderCounter = 0;
    for l = 1:L; % Movement Steps
        for k = 1:K; % Ants
            % Ant Check: Deadlocked or Image Bordered (No Movement!)
            if ((antDeadlock(k,n)==1) || (antOnBorder(k,n)==1))
                continue;
            end % if "Ant Check"
            % Current Ant Position: (i,j)
            i = antTour{k,n}(1,1);
            j = antTour{k,n}(1,2);
            % Neighborhood Pheromone Information: nT
            nT(1:3,1:3) = T(i-1:i+1, j-1:j+1);
            nT(2,2) = 0; % Nullify (Exclude) Current Pixel
            %display(nT);
            % Neighborhood Intensity Variation Information: nH
            nH(1:3,1:3) = H(i-1:i+1, j-1:j+1);
```

```
            nH(2,2) = 0; % Nullify (Exclude) Current Pixel
            %display(nH);
            % (Traditional ACO) Transition Probabilities: p
            p(1:3,1:3) = ((nT.^a).*(nH.^b))./(sum(dot((nT.^a),(nH.^b))));
            %display(p);
            % No Direction Control (First Movement Step!)
            if (l == 1)
                % Ant Current Position Equals Ant Start Position
            % Direction Control
            else
                % Directions: North-East, East, South-East
                if (j > antTour{k,n}(1,2))
                    g = atan((i-antTour{k,n}(1,1))/(j-antTour{k,n}(1,2)));
                % Directions: North-West, West, South-West
                elseif (j < antTour{k,n}(1,2))
                    % Directions: West, South-West
                    if (i >= antTour{k,n}(1,1))
                        g = atan((i-antTour{k,n}(1,1))/(j-antTour{k,n}(1,2))) + pi;
                    % Direction: North-West
                    else
                        g = atan((i-antTour{k,n}(1,1))/(j-antTour{k,n}(1,2))) - pi;
                    end % if else
                % Directions: North, South
                else
                    % Direction: North
                    if (i < antTour{k,n}(1,1))
                        g = -(pi/2);
                    % Direction: South
                    else
                        g = pi/2;
                    end % if else
                end % if elseif else
                %display(g);
                % Neighborhood Direction Control: nQ
                nQ = r.*(cos(pol-g)+1);
                %nQ = (cos(pol-g)+1).^r;
                nQ(2,2) = 0; % Nullify (Exclude) Current Pixel
                %display(nQ);
                % (DCACO) Transition Probabilities: p
                p = (p+nQ)./(sum(p(:)+nQ(:)));
                %display(p);
            end % if else "Direction Control"
            % Local Destination Index: (lx,ly) (May Be More Than One!)
            [lx ly] = find(p==max(p(:)));
            % Destination: (x,y)
            x = i-2+lx(1,1);
            y = j-2+ly(1,1);
            % Destination Check: Visited Pixels
            while ((any(ismember(antTour{k,n}(:,1:2),[x y],'rows'))==1) && (max(p(:))~=0))
                p(lx,ly) = 0; % Nullify (Exclude) Visited Pixel
                %display(p);
                [lx ly] = find(p==max(p(:)));
                % New Destination: (x,y)
                x = i-2+lx(1,1);
                y = j-2+ly(1,1);
            end % while "Destination Check"
            % Ant Deadlock Check: Prevent Further Ant (Round) Movement
            if (max(p(:)) == 0)
                display(['Ant ' num2str(k) ' DEADLOCKED at pixel [' num2str(i) ',' num2str(j↩
                    ) '] after ' num2str(l-1) ' steps in round ' num2str(n)]);
                antDeadlock(k,n) = 1;
                deadlockCounter = deadlockCounter + 1;
                continue;
            end % if "Ant Deadlock Check"
            % Ant Border Check: Prevent Further Ant (Round) Movement
            if ( (x==1) || (x==rows) || (y==1) || (y==cols) )
                display(['Ant ' num2str(k) ' IMBORDERED at pixel [' num2str(x) ',' num2str(y↩
                    ) '] after ' num2str(l) ' steps in round ' num2str(n)]);
                antOnBorder(k,n) = 1;
                borderCounter = borderCounter + 1;
            end % if "Ant Border Check"
            % Ant (Round) Movement: To Destination (x,y)
            antTour{k,n}(l+1,1) = x;
```

```
            antTour{k,n}(l+1,2) = y;
            % Immediate Local Pheromone Update: Pheromone Decay
            T(x,y) = (1-w).*T(x,y) + w.*T0(x,y);
        end % for K Ants
    end % for L Movement Steps
    for k = 1:K; % Ants
        for l = 2:size(antTour{k,n},1);
            % Offline Pheromone Update: Pheromone Evaporation
            T(antTour{k,n}(l,1), antTour{k,n}(l,2)) = (1-evop).*T(antTour{k,n}(l,1), antTour↩
                {k,n}(l,2)) ...
                                                    + evop.*H(antTour{k,n}(l,1), antTour↩
                                                        {k,n}(l,2));
        end % for l
    end % for k
    fprintf('\n');
    display(['DEADLOCKS: ' num2str(deadlockCounter) ' (' num2str(((deadlockCounter/K)*100)) ↩
        '%)']);
    display(['IMBORDERS: ' num2str(borderCounter) ' (' num2str(((borderCounter/K)*100)) '%)'↩
        ]);
    toc(tCon);
end % for N Rounds

fprintf('\n...Ants done!\n');

end
% end DCACO.m
```

Morphological Image Processing (MIP.m)

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                           %
%                   Morphological Image Processing (MIP.m)                  %
%                                                                           %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [ERED DRED MGED RNMGED1 RNMGED2 RNMGED3 RNMGED4] = MIP(I)

% Structuring Elements
%   0 0 0   0 0 1   0 1 0   1 0 0
%   1 1 1   0 1 0   0 1 0   0 1 0
%   0 0 0   1 0 0   0 1 0   0 0 1
SE = [ strel('line', 3, 0)
       strel('line', 3, 45)
       strel('line', 3, 90)
       strel('line', 3, 135) ];

% Erosion Residue Edge Detector (ERED)
ERED = I - imerode(I,SE);

% Dilation Residue Edge Detector (DRED)
DRED = imdilate(I,SE) - I;

% Morphological Gradient Edge Detector (MGED)
MGED = imdilate(I,SE) - imerode(I,SE);

% Reduced Noise Morphological Gradient Edge Detection #1 (RNMGED1)
M = imopen(imclose(I,SE(1)),SE(1));
RNMGED1 = imdilate(imclose(M,SE(1)),SE(1)) - imclose(M,SE(1));
% Reduced Noise Morphological Gradient Edge Detection #2 (RNMGED2)
M = imopen(imclose(I,SE(2)),SE(2));
RNMGED2 = imdilate(imclose(M,SE(2)),SE(2)) - imclose(M,SE(2));
% Reduced Noise Morphological Gradient Edge Detection #3 (RNMGED3)
M = imopen(imclose(I,SE(3)),SE(3));
RNMGED3 = imdilate(imclose(M,SE(3)),SE(3)) - imclose(M,SE(3));
% Reduced Noise Morphological Gradient Edge Detection #4 (RNMGED4)
M = imopen(imclose(I,SE(4)),SE(4));
RNMGED4 = imdilate(imclose(M,SE(4)),SE(4)) - imclose(M,SE(4));

end % MIP.m
```

Crack Closing (CC.m)

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                               %
%                              Crack Closing (CC.m)                             %
%                                                                               %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [B] = CC(G, A, N, EP, GP, E)
% B  = Output Crack Information
% G  = Input Gray Scale Image
% A  = Input Ant Edge Image
% N  = Input Number of Iterations
% EP = Input Edge Percentile (%)
% GP = Input Gray Percentile (%)
% E  = Input Shape Measurement: Eccentricity

% MISC
[rows cols] = size(G);                                                  % Image ↩
    Dimensions

% Statistics (percentile)
crack = prctile(sort(G(:)), GP);                                        % Crack (< Gray) ↩
    Threshold
edge =  prctile(sort(A(:)), EP);                                        % Edge (> Ant) ↩
    Threshold

% Binary Image
B = im2bw(A,edge);                                                      % Original (↩
    Binary) Ant Image

% Figure(s)
set(0,'DefaultFigurePosition',get(0,'ScreenSize'));
figure(1); imshow(G,'InitialMagnification','fit','Border','tight');
pause(2);
figure(2); imshow(A,'InitialMagnification','fit','Border','tight');
pause(2);
figure(3); imshow(~B,'InitialMagnification','fit','Border','tight');
pause(2);

% Remove Ant Frame
B(3,:) = 0;                                                             % Clear Row: 3
B(rows-2,:) = 0;                                                        % Clear Row: Row↩
    -2
B(:,3) = 0;                                                             % Clear Col: 3
B(:,cols-2) = 0;                                                        % Clear Col: Col↩
    -2

% Noise Filtering
cc = bwconncomp(B,8);                                                   % Connected ↩
    Components (8-Connectivity)
stat  = regionprops(cc, 'Eccentricity');
for i = 1:cc.NumObjects;
    if( stat(i).('Eccentricity') < E )
        B(cc.PixelIdxList{i}) = 0;
    end % if
end % for

figure(4); imshow(~B,'InitialMagnification','fit','Border','tight');
pause(2);

b = B;                                                                  % Processed (↩
    Binary) Ant Image
% Crack Highlighting and Closing
for n = 1:N; % Iterations
    % Binary Edge (1) Pixels
    [i j] = find(b==1);
    b(:) = 0;
    for x = 1:numel(i);
        % Not Image Border Pixel
        if( i(x)~=1 && j(x)~=1 && i(x)~=rows && j(x)~=cols )
```

```
            % Crack Highlighting
            if (n==1)
                % Horisontal Crack Highlighting (HCH): [ - ]
                if((G(i(x),j(x)-1) < G(i(x),j(x)+1)) && (G(i(x),j(x)-1) < G(i(x),j(x))))
                    B(i(x),j(x)-1) = 1;
                    b(i(x),j(x)-1) = 1;
                elseif( (G(i(x),j(x)+1) < G(i(x),j(x)-1)) && (G(i(x),j(x)+1) < G(i(x),j(x)))↩
                    )
                    B(i(x),j(x)+1) = 1;
                    b(i(x),j(x)+1) = 1;
                else
                    % Crack Horisontally Highlighted
                end % if HCH
                % Vertical Crack Highlighting (VCH): [ | ]
                if((G(i(x)-1,j(x)) < G(i(x)+1,j(x))) && (G(i(x)-1,j(x)) < G(i(x),j(x))))
                    B(i(x)-1,j(x)) = 1;
                    b(i(x)-1,j(x)) = 1;
                elseif( (G(i(x)+1,j(x)) < G(i(x)-1,j(x))) && (G(i(x)+1,j(x)) < G(i(x),j(x)))↩
                    )
                    B(i(x)+1,j(x)) = 1;
                    b(i(x)+1,j(x)) = 1;
                else
                    % Crack Vertically Highlighted
                end % if VCH
                % Diagonal Crack Highlighting #1 (DCH1): [ \ ]
                if((G(i(x)-1,j(x)-1) < G(i(x)+1,j(x)+1)) && (G(i(x)-1,j(x)-1) < G(i(x),j(x))↩
                    ))
                    B(i(x)-1,j(x)-1) = 1;
                    b(i(x)-1,j(x)-1) = 1;
                elseif( (G(i(x)+1,j(x)+1) < G(i(x)-1,j(x)-1)) && (G(i(x)+1,j(x)+1) < G(i(x),↩
                    j(x))) )
                    B(i(x)+1,j(x)+1) = 1;
                    b(i(x)+1,j(x)+1) = 1;
                else
                    % Crack Diagonally \ Highlighted
                end % if DCH1
                % Diagonal Crack Highlighting #2 (DCH2); [ / ]
                if((G(i(x)+1,j(x)-1) < G(i(x)-1,j(x)+1)) && (G(i(x)+1,j(x)-1) < G(i(x),j(x))↩
                    ))
                    B(i(x)+1,j(x)-1) = 1;
                    b(i(x)+1,j(x)-1) = 1;
                elseif( (G(i(x)-1,j(x)+1) < G(i(x)+1,j(x)-1)) && (G(i(x)-1,j(x)+1) < G(i(x),↩
                    j(x))) )
                    B(i(x)-1,j(x)+1) = 1;
                    b(i(x)-1,j(x)+1) = 1;
                else
                    % Crack Diagonally / Highlighted
                end % if DCH2
            % Crack Closing
            else % (n > 1)
                [li lj] = find( G(i(x)-1:i(x)+1,j(x)-1:j(x)+1) < crack );
                for lx = 1:numel(li);
                    B(i(x)-2+li(lx),j(x)-2+lj(lx)) = 1;
                    b(i(x)-2+li(lx),j(x)-2+lj(lx)) = 1;
                end % for lx
            end % if "Crack Highlighting" else "Crack Closing"
        end % if "Not Image Border Pixel"
    end % for x

    imshow(~B,'InitialMagnification','fit','Border','tight');
    pause(0.25);

end % for n

B = ~B;

end % CC.m
```

References

[1] V. F. Thorsen, "Meteorite Impact Crater Crack Extraction and Image Edge Detection using Ant Colony Optimization," December 2010.

[2] P. Peterlin, "Morphological Operations: An Overview." Website. `http://www.inf.u-szeged.hu/ssip/1996/morpho/morphology.html`.

[3] T. M. Inc., "Measure properties of image regions." Website. `http://www.mathworks.com/help/toolbox/images/ref/regionprops.html`.

[4] M. Dorigo, M. Birattari, and T. Stützle, "Ant Colony Optimization – Artificial Ants as a Computational Intelligence Technique," *IEEE Computational Intelligence Magazine*, vol. 1, pp. 28–39, November 2006.

[5] J. Tian, W. Yu, and S. Xie, "An ant colony optimization algorithm for image edge detection," in *2008 IEEE World Congress on Evolutionary Computation*, pp. 751–756, IEEE, June 2008.

[6] A. V. Baterina and C. Oppus, "Image Edge Detection Using Ant Colony Optimization," *International Journal Of Circuits, Systems And Signal Processing*, vol. 4, 2010.

[7] D. Ziou and S. Tabbone, "Edge Detection Techniques - An Overview," *International Journal of Pattern Recognition and Image Analysis*, vol. 8, pp. 537–559, 1998.

[8] N. Otsu, "A Threshold Selection Method from Gray-Level Histograms," *Systems, Man and Cybernetics, IEEE Transactions on*, vol. 9, no. 1, pp. 62–66, 1979.

[9] K. R. Castleman, *Digital Image Processing*. Prentice-Hall, 1996.

[10] Z. Yu-qian, G. Wei-hua, C. Zhen-cheng, T. Jing-tian, and L. Ling-yun, "Medical Images Edge Detection Based on Mathematical Morphology," September 2005.

[11] T. Niemueller, "Automatic Detection and Segmentation of Cracks in Underground Pipeline Images," 2006.

[12] M. Nixon and A. S. Aguado, *Feature Extraction & Image Processing, Second Edition*. Academic Press, 2 ed., January 2008.

Attachments

Content found on the attached CD/DVD:

- Report in .pdf format.
- Matlab source code (m-files).
- Ritland meteorite impact crater images.
- Experimental results.