Quantification of microcrack characteristics and implications for stiffness and strength of granite

L. Griffiths[1], M.J. Heap[1], P. Baud[1], J. Schmittbuhl[1]

[1]Institut de Physique de Globe de Strasbourg, Université de Strasbourg/EOST, CNRS UMR 7516, 5 rue René Descartes, 67084 Strasbourg cedex, France.

Corresponding author: Luke Griffiths (luke.griffiths@unistra.fr)

Supplementary Materials: Image Processing and Analysis

To automatically calculate the 2D microcrack length and number of microcracks per unit area (number density), micrographs first need to be processed to obtain binary images of the microcracks. We chose to use optical microscopy over Scanning Electron Microscopy (SEM) as it is cheaper and quicker to produce a large number of images. However, unlike SEM—which is sensitive to mass density—grains or crystals may appear a similar dark colour to microcracks in optical micrographs. Moreover, as microcracks must be distinguished from pores, any automated procedure to map microcracks cannot rely on grayscale alone. The method we present here takes advantage of the difference in aspect ratio between microcracks and pores/grains/crystals. The processing can be broken down into four main steps: (1) involves filtering the microscope image to accentuate thin objects, (2) is the segmentation step, separating the microcracks from the image background to create a binary image, (3) is the iterative thinning or "skeletonisation" of the microcracks to form objects that are a single pixel in thickness, and (4) involves splitting highly curved microcracks into multiple, straighter microcracks. The script for the image processing is written in Python and uses various functions for multi-dimensional image processing from: Scipy[1] ndimage package; the scikit-image[2] package; and the Mahotas[3] image processing package. Throughout, we work with 8-bit grayscale images; 2D integer arrays containing values ranging from 0 (black) to 255 (white). The script is available on demand from the corresponding author.

1. Filtering

Microcracks are low aspect ratio features[4]. To obtain an image containing only microcracks, we must first separate them from the higher aspect ratio pores and grains/crystals. The original grayscale micrograph (Figure A.1a) is subject to a median filter that diminishes the thin microcracks whilst preserving larger objects such as pores and grains (Figure A.1b). The median filter involves a moving window (here 15 x 15 pixels) that replaces the value at the centre of the window with the median of the values surrounding it. The window size was chosen to be several times greater than the microcrack aperture. The advantage of the median filter is its efficiency in preserving edges, unlike smoothing filters such as Gaussian blur or mean filters. Calculating the difference between these images results in a grayscale image containing only thin or small objects (Figure A.1c).

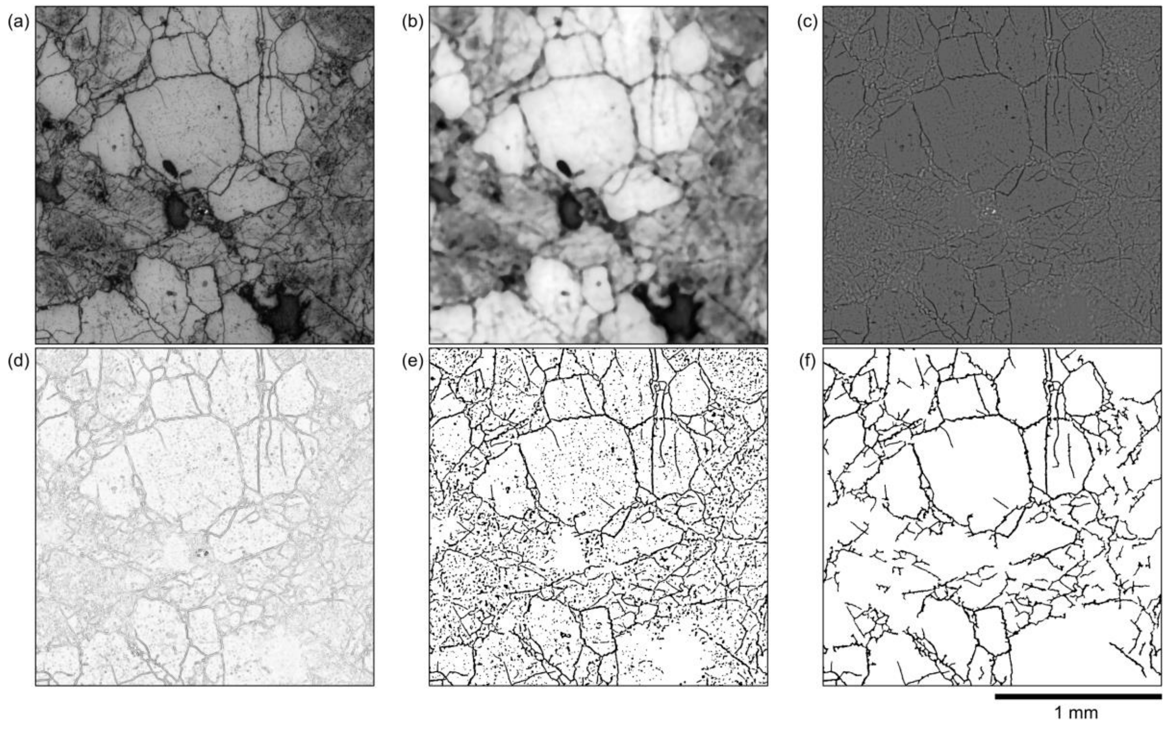

Figure A.1. *Filtering and segmentation: (a) the original cropped 2 mm × 2 mm optical micrograph of a fine-grained granite sample heated to 600 °C, (b) the median filtered original, (c) the difference of (b) and (a), (d) the Sobel filtered gradient of (c), (d) the result of the watershed segmentation of (c) using (d), and (f) the binary segmented image with noise removed.*

2. Segmentation

To create a binary image of the microcracks, the employed segmentation method is the watershed algorithm (part of the scikit-image[2] image processing Python package), which treats the grayscale as topography, "flooding" the image to separate areas of similar elevation. The watershed algorithm requires two input images, a seed image and a gradient image. The seed image contains markers which label areas of Figure A.1c as microcracks, background or unknown, depending on their grayscale values. Here, dark areas (low grayscale value) are labelled as microcracks and light areas (high grayscale value) are the background. We calculate the seed image from the median grayscale value of Figure A.1c, which provides the grayscale value of the background. All areas of grayscale above the median minus 10 are labelled as background. All areas of grayscale less than 20 below the median are labelled as microcracks. Pixels with intermediate grayscales do not have a label. The gradient image of Fig. A.1c is generated using Sobel filter and is shown in Fig. A.1d. The Sobel filter calculates the discreet central difference gradient in grayscale at each pixel in both the horizontal and vertical directions and then sums their squares. The resulting grayscale image is dark where the gradient is steep and light in flatter areas (Fig. A.1d).Tthe watershed algorithm then extends the labelled regions of the seed image, prioritising extension in the direction of low gradients (given by the gradient image). This process continues until the labelled areas meet and the entire image is segmented (Fig. A.1e).

Next, noise is removed by locating and removing all small objects. The “label” and “find_objects” functions of the Scipy Ndimage Python package provide an ensemble of rectangular arrays containing each object. The dimensions of these arrays are the height and width of the object in pixels. We then calculate the length of their diagonal and remove all of those that are smaller than a given value (here 80 µm or 40 pixels).

Following segmentation, some white pixels may be present inside the black cracks. The removal of the white pixels is achieved by “dilating” and “eroding” the image. Dilation is a morphological technique which involves visiting each pixel and setting it to black if there is a black pixel adjacent. Once all holes are filled, a consequent erosion of the image returns the black areas to their original thickness without reopening holes. Erosion works similar to dilation, setting each pixel to white if there is an adjacent white pixel.

3. Thinning and pruning

Next, we iteratively thin the cracks to a thickness of a single pixel, whilst conserving their length and connectivity, using the “thin” function of the Mahotas Python package[3]. The aim is to make it easier to calculate the microcrack length, and to split branched microcrack networks into individual microcracks. Like erosion and dilation, thinning is another morphological technique that involves scanning the image for certain structural elements using a hit-or-miss transform before removing them. These structural elements are 3×3 pixel arrays that represent all the possible configurations for black pixels along the microcrack boundaries that could be set to white without compromising the continuity of the microcrack. Once these pixels have been located, they are set to white and the process begins again. This process is continued until further thinning has no effect and the image has been entirely skeletonized (Fig. A.2a).

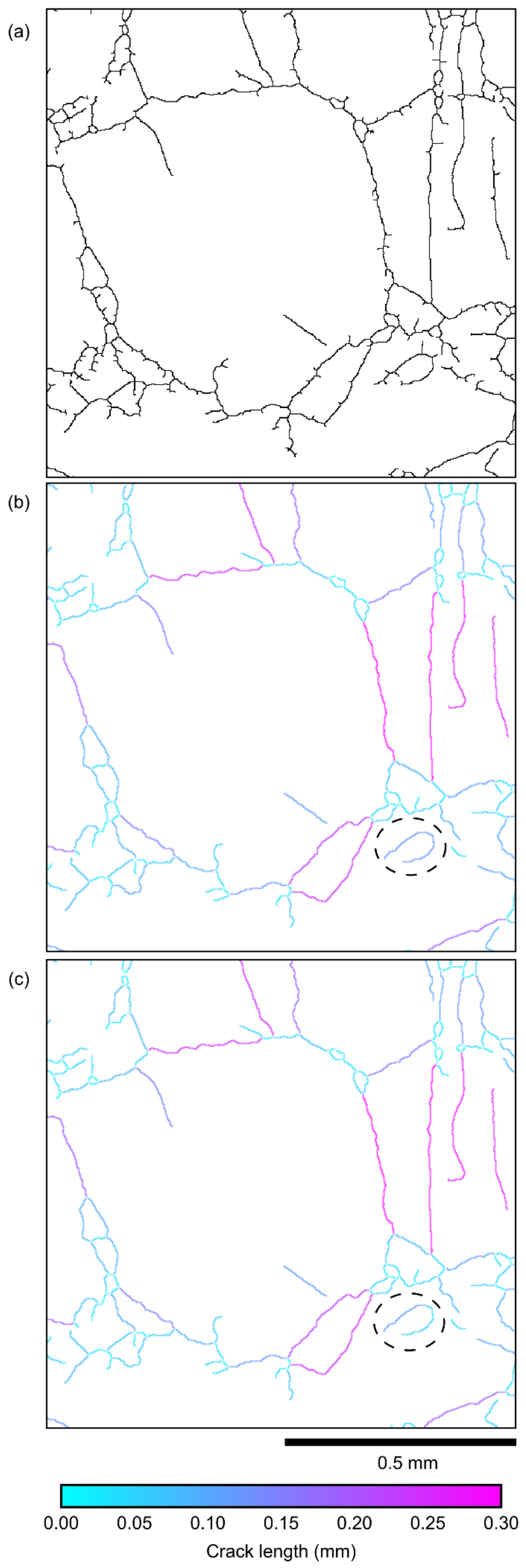
(a)
(b)
(c)
0.5 mm
0.00
0.05
0.10
0.15
0.20
0.25
0.30
Crack length (mm)

Figure A.2. *(a) The thinned version of the segmented image (centre of Fig. A.2f cropped to 1 x 1 mm), which exhibits short residual branches from the thinning algorithm. The microcracks are then pruned (b), and their original lengths are restored in (c) following the procedure presented in Fig. A.3. The microcracks in (b) and (c) are coloured according to their approximated lengths, we see that a highly curved microcrack in (b) has been split into two shorter microcracks in Fig. (c) as demonstrated by Fig. A.4.*

Following the thinning procedure there are often short residual branches (Fig. A.2a) that must be removed to prevent them from later contributing to the microcrack population. The method employed to prune these unwanted branches involves two steps. First, a hit-or-miss transform iteratively locates and removes all end points of the skeleton image. The number of iterations is set to 15, the same as the median filter window size. The resulting image is now assumed to contain only microcracks, although some may still retain more than two end points at this stage. To resolve this, the second step of pruning involves another hit-or-miss algorithm to locate and remove all intersection points (or branch points) in the image, separating branches into individual microcracks. As previous, the end points are then iteratively removed until all microcracks have only two ends (Fig. A.2b).

Whilst pruning removes the parasitic branches, it also shortens the main branches (Figs. A.2a, A.2b). To restore the length of the main branches, one could iteratively dilate the pruned image and intersect it with the unpruned original. The disadvantage of this approach is that residual branches near microcrack ends are also recovered. To avoid this, we propose a modification of this method, demonstrated in Fig. A.3. First, we locate all microcracks in the pruned image and dilate their end points before intersecting them with the un-pruned microcrack (as described above). Next, all branch points are located and removed, isolating the pixels of the growing parasitic branches that are removed once again by locating and removing all small objects (those composed of up to 2 pixels). The branch points are then redrawn and the procedure is repeated until there are no further changes to the skeleton image. This method allows for an almost complete recovery of the microcrack length lost during the first stage of pruning, without recovering any residual branches. The resulting image (Fig. A.2c) contains microcracks that are each a single pixel in width and have exactly two end points. This geometry makes it easier to calculate their lengths and split up any highly curved microcracks into multiple straighter microcracks.

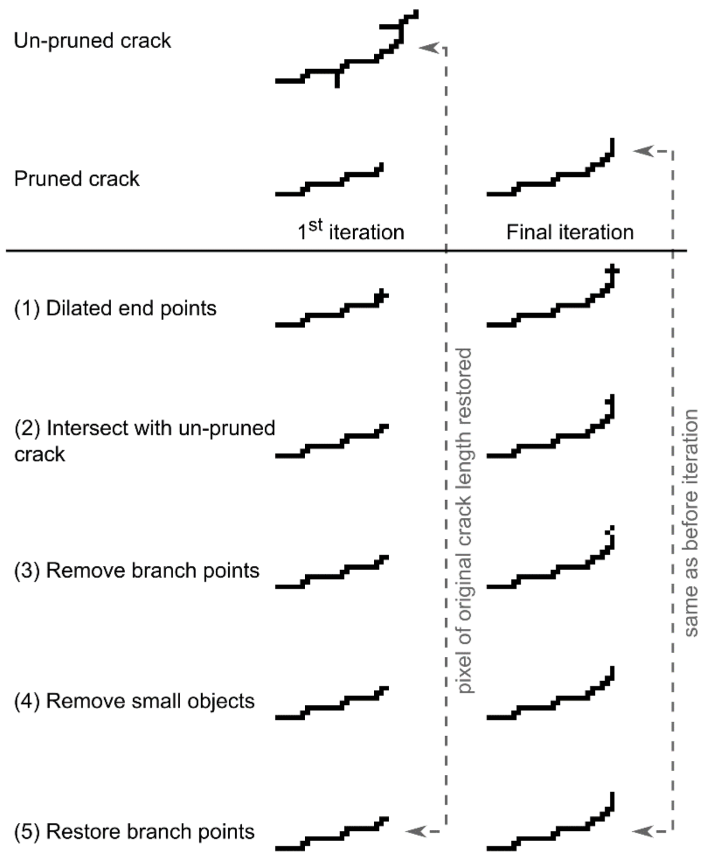

Figure A.3. *An example showing the iterative procedure to restore the original microcrack length following the pruning of parasitic branches. The microcrack is iteratively dilated at its end points, whilst any pixels of the residual branches which may reappear are removed at each iteration. This algorithm ends once further iterations no longer affect the image, resulting in the restoration of the original microcrack length.*

4. Microcrack length and curvature

The length of a microcrack can be approximated by the distance between its two end points (Fig. A.4). This assumption is however only strictly true when the microcrack follows a straight line and is increasingly unsatisfactory as microcrack curvature increases. Before calculating the microcrack lengths, we scan the image for microcracks that have an overall curvature greater than a certain value and split them into two shorter microcracks, repeating the process until all microcracks fulfil this criterion. The curvature is quantified by the ratio of the maximum distance between the microcrack and the straight microcrack approximation, to the length of the straight microcrack approximation (Fig. A.4). If this ratio is greater than a user-defined value, the farthest pixel from the approximated microcrack is removed thus separating the original curved microcrack in two. The microcracks in Fig. A.2c have been split where the curvature was high. The maximum ratio used here corresponds to a microcrack made up of two straight lines of the same length, at an angle of 70° to each other.

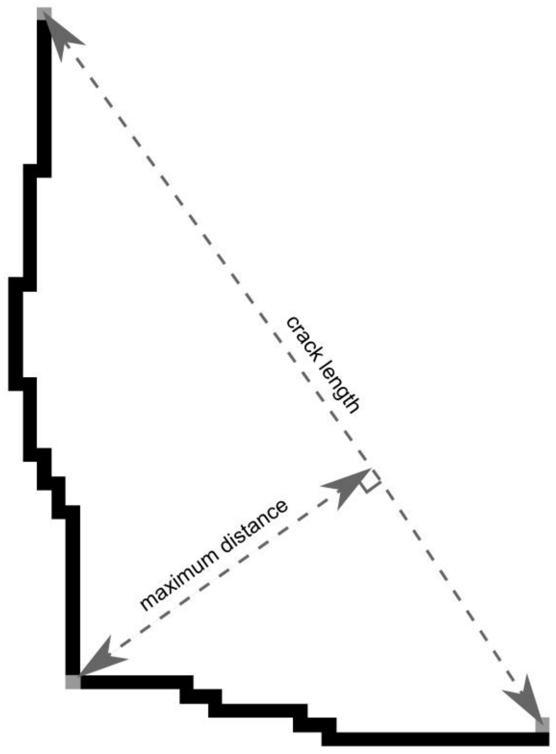

Figure A.4. *The microcrack length is approximated by the distance between its two end points. If the farthest pixel of the microcrack is too far from this approximation (defined by the user), it is removed, splitting the microcrack into two shorter and straighter microcracks.*

References

1. Jones E, Oliphant T, Peterson P. ${$SciPy$}$: open source scientific tools for ${$Python$}$. 2014. http://www.citeulike.org/group/19049/article/13344001. Accessed October 2, 2017.

2. Van der Walt S, Schönberger JL, Nunez-Iglesias J, et al. scikit-image: image processing in Python. *PeerJ*. 2014;2:e453.

3. Coelho LP. Mahotas: Open source software for scriptable computer vision. *Journal of Open Research Software*. 2013;1(1). doi:10.5334/jors.ac.

4. Simmons G, Richter D. Microcracks in rocks. *The physics and chemistry of minerals and rocks*. 1976:105–137.