Path opening library

Ben Appleton and Hugues Talbot

February 27, 2013

1 Introduction

Path openings and closing [10, 4, 1, 3, 2] are a way to filter an image using mathematical morphology principles [7, 6], [8],[5] using families of paths as structuring elements.

In most filtering techniques, a noise and/or an image model are assumed, e.g. Gaussian noise and piecewise smooth images. With mathematical morphology, we assume images contains objects and/or noise which conform to some geometrical and/or topological properties.

Structuring elements, which are basically sliding windows on which we compute a maximum or a minimum instead of a mean or a median, are key to this idea. Their shape and size are important. For instance, say that we want to extract (technically, to segment) thin objects that are locally linear. We could use families of straight segments, as in [9] to filter out anything that would not be locally so, and use other filters such as a top-hat to eliminate thick objects.

However, very thin objects would also be filtered out using a family of segments. Instead, it would be useful to be able to use structuring elements that are locally linear but not necessarily perfectly straight. This is where path openings come in: they allow us to use families of paths that fit within a more or less constrained cone.

Such families have a population that grows exponentially with the path length. If we were to implement path openings by combining "traditional" structuring-element-based approach, it would be unworkable.

Instead, path openings employ a decomposition technique described in [4] and an ordered algorithm described in [10] to achieve very reasonable computing times. We can also improve this approach by leaving a few pixels out of the paths at random, which makes them very robust to noise.

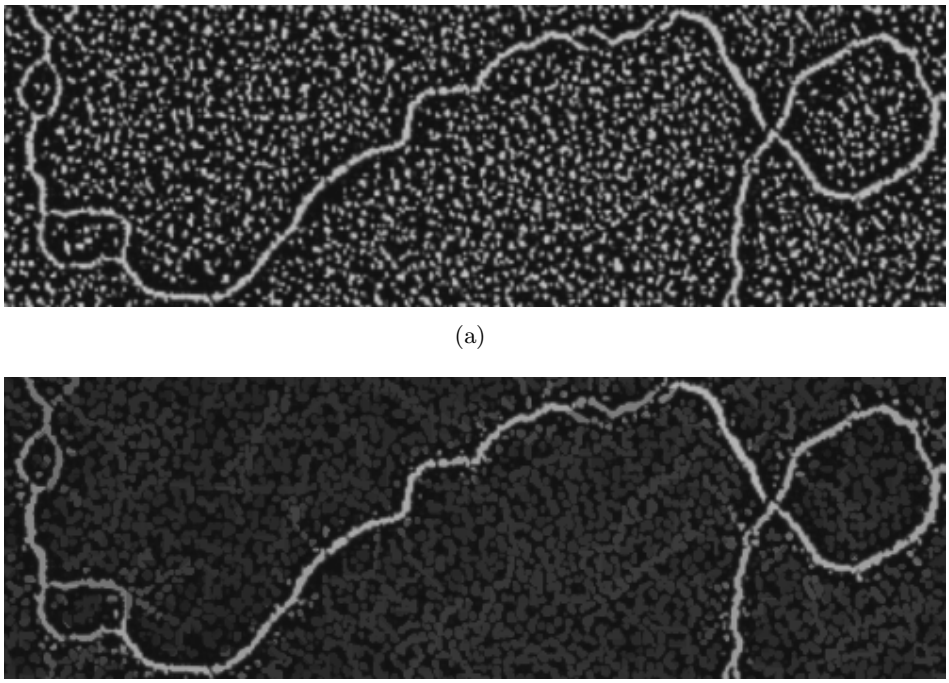
In essence, a path opening will regain any bright feature over a darker background in which a path of a given length can fit. A path closing does the same for dark features over a bright background.

2 Code compilation

Code compilation is fairly straightforward.

Requirements The library ImageMagick is required, with its associated development files. For instance, under Ubuntu 10.4, the following packages are required:

- `libmagickcore2`



(a)

(b)

Figure 1: Example path opening with length 70 pixels and tolerance 3 : (a) original, (b) filtered result

- `libmagickcore-dev`

Note that ImageMagick is only required for image I/O, so if `/usr/include/ImageMagick/magick/api.h` is installed, and the script `MagickCore-config` is in your path, most likely you will be fine. The rest of the distribution is self-contained.

Under Windows, try using the `http://www.cygwin.com/`Cygwin environment. This is currently untested. Please feel free to report success.

Edit the makefile Most likely, you will need to edit the lines

```
PREFIX=/opt/local
```

in the makefile. This is valid for Mac OSX with the MacPorts project installed. For Ubuntu, change this to `PREFIX=/usr` and this should be ok.

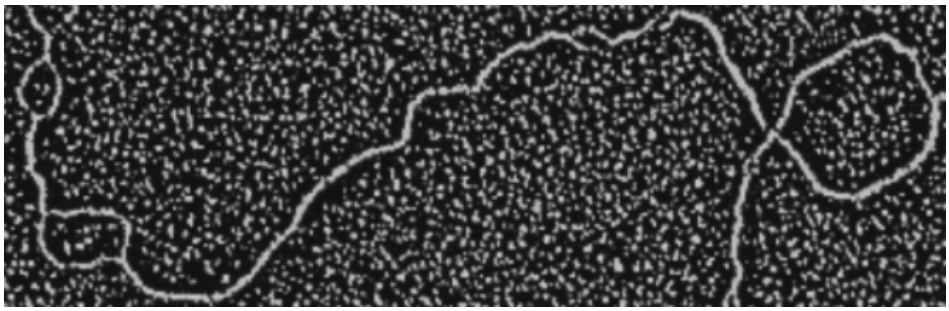
Compilation Depending on your setting, you might also need to set up the `PKG_CONFIG_PATH` environment variable, if the program. The symptom is the following:

```
Package MagickCore was not found in the pkg-config search path.
Perhaps you should add the directory containing 'MagickCore.pc'
```

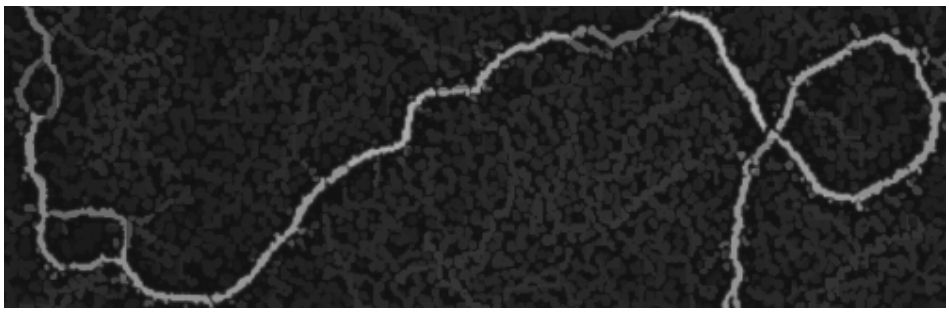
For example, you might have to type

```
export PKG_CONFIG_PATH=/opt/local/lib/pkgconfig
```

Then just type `make` in a terminal.



(a) Initial image



(b) Path opening with L=70,K=1

Figure 2: Path opening result

3 Usage

The makefile will produce an executable called `test_pathopen`

```
$ ./test_pathopen
Usage : ./test_pathopen <input image> L K <output image>
Where : <input image> is an 8-bit grey-level image in any format readable by ImageMagick
        L is the length of the path
        K is the number of admissible missing pixels
        <output image> is an 8-bit image. The extention determines the format
```

There is an example image in the `images` directory.
just run the following example:

```
./test_pathopen images/DNAGI.tif 70 1 result.tif
```

Any output format is suitable, just specify it by its extension (`result.png`, etc), however make sure your output format is compatible with 16-bit output, so TIFF and PGM are good choices.

With the above parameters, the results is shown in Fig. 2.

4 Library

Here be the documentation about the ABI.

References

[1] Ben Appleton and Hugues Talbot. Efficient path openings and closings. In C. Ronse, L. Najman, and E. Decencière, editors, *Mathematical Morphology: 40 Years On*, volume 30 of *Computational Imaging and Vision*, pages 33–42, Dordrecht, 2005. Springer-Verlag.

[2] M. Buckley and H. Talbot. Flexible linear openings and closings. In L. Vincent and D. Bloomberg, editors, *Mathematical Morphology and its application to image analysis*, pages 109–118, Palo Alto, June 2000. ISMM, Kluwer.

[3] H. Heijmans, M. Buckley, and H. Talbot. Path-based morphological openings. In *Proceedings of ICIP'04*, pages 3085–3088, Singapore, October 2004. IEEE.

[4] H. Heijmans, M. Buckley, and H. Talbot. Path openings and closings. *Journal of Mathematical Imaging and Vision*, 22:107–119, 2005.

[5] L. Najman and H. Talbot, editors. *Morphologie mathématique 1: approches déterministes*. Lavoisier, Paris, 2008. ISBN 978-2746218413.

[6] J. Serra. *Image Analysis and Mathematical Morphology*. Academic Press, 1982.

[7] J. Serra. *Image Analysis and Mathematical Morphology - Volume II : Theoretical Advances*. Academic Press, London, 1988.

[8] P. Soille. *Morphological Image Analysis, principles and applications*. Springer-Verlag, 2nd edition, 2003. ISBN 3-540-42988-3.

[9] P. Soille and H. Talbot. Directional morphological filtering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(11):1313–1329, 2001.

[10] H. Talbot and B. Appleton. Efficient complete and incomplete paths openings and closings. *Image and Vision Computing*, 25(4):416–425, April 2007.