Published in the proceedings of ISMM 2015.
The final publication is available at `link.springer.com`.

Fast Computation of Greyscale Path Openings

Jasper J. van de Gronde[*], Herman R. Schubert[*], and Jos B. T. M. Roerdink

Johann Bernoulli Institute for Mathematics and Computer Science,
University of Groningen, P.O. Box 407, 9700 AK Groningen, The Netherlands
{j.j.van.de.gronde,j.b.t.m.roerdink}@rug.nl, h.robert.schubert@gmail.com [**]

Abstract. Path openings are morphological operators that are used to preserve long, thin, and curved structures in images. They have the ability to adapt to local image structures, which allows them to detect lines that are not perfectly straight. They are applicable in extracting cracks, roads, and more. Although path openings are very efficient to implement for binary images, the greyscale case is more problematic. This study provides an analysis of the main existing greyscale algorithm, and shows that although its time complexity can be quadratic in the number of pixels, this is optimal in terms of the output (if the full opening transform is created). Also, it is shown that under many circumstances the worst-case running time is much less than quadratic. Finally, a new algorithm is provided, which has the same time complexity, but is simpler, faster in practice and more amenable to parallelization.

Keywords: path openings, algebraic morphological operators, attributes, stack opening, time complexity.

1 Introduction

It is often useful to be able to extract long, but not necessarily thick, structures, for example: guide-wires in X-ray fluoroscopy [2], roads in remote sensing images [12], and cracks for non-destructive testing [7]. A possible way to do this is to apply openings using fixed line structural elements [8]. However, these openings can be inadequate if the features of interest are not perfectly straight, and they can be fairly expensive if needed for multiple directions. Path openings [4] solve these issues by looking for paths in a small number of purpose-made directed acyclic graphs (DAGs) (see Fig. 1).

Path openings were originally introduced by Heijmans et al. [4], along with an algorithm for the binary case that is roughly linear in the number of pixels. Unfortunately, the proposed algorithm does not transfer immediately to the greyscale case. To compute the opening in the greyscale case, we would have to perform the binary opening for every unique grey level, which becomes highly inefficient for images with a large number of grey levels. Talbot and Appleton [11] improved on this by only looking at the differences between adjacent

[*] These authors contributed equally.

[**] This research is partially funded by the Netherlands Organisation for Scientific Research (NWO), project no. 612.001.001.

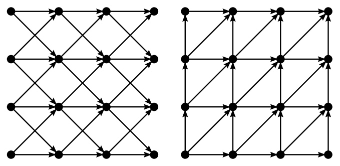


Fig. 1. Illustration of the DAGs used for path openings (our implementations only use graphs for the horizontal and vertical directions).

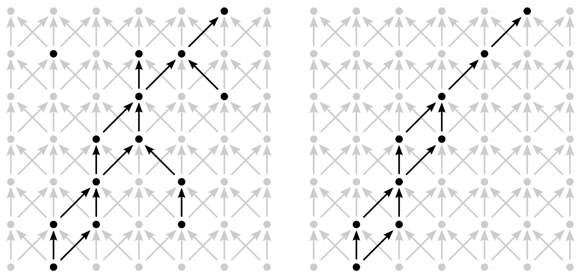


Fig. 2. A set $X \subseteq \Omega$ (black points on the left) and its path opening with $k = 7$ (black points on the right). Points only contained in paths of length 6 or less have been discarded (also see Fig. 3). The underlying adjacency graph is light grey, with black arrows highlighting the edges that are part of paths.

threshold levels, which can greatly reduce the amount of work needed. Unfortunately, despite several (additional) optimizations, the Talbot algorithm can take minutes when processing large images on modern desktop machines, raising the question whether it is possible to do better. Morard et al. [10] propose using 1D "path" openings on a carefully selected subset of possible paths to speed up path openings, but this only gives an approximation of the full path opening.

Despite some educated guesses [1, 3, 11] (of the *expected* time complexity), the time complexity of the algorithm developed by Talbot and Appleton (Talbot's algorithm for short) was never rigorously analysed. Here we show that the opening transform (the most general output of the algorithm) has a *space* complexity in $O(\min(d, |L|)\,|\Omega|)$, and that the *time* complexity of Talbot's algorithm is restricted purely by the size of the opening transform. Here d is the depth of the graph (typically the width/height of the image), L the set of grey levels in the image, and Ω the image domain. This implies the time complexity could be quadratic in the worst case, but only in the unlikely event that both the depth d *and* the number of grey levels $|L|$ can be considered of the same order as the number of pixels $|\Omega|$. We also introduce a new algorithm with the same time complexity as Talbot's algorithm, but which is simpler, faster in practice, and more amenable to parallelization. We demonstrate a considerable (additional) speedup when using a parallel implementation. Finally, it should be noted that our results are not limited to 2D images: in all our derivations and algorithms we simply assume the input is a weighted (sparse) directed acyclic graph.

2 Definitions

Path openings are constructed on directed acyclic graphs (DAGs). DAGs can be defined using a binary relation '$\mapsto$'. When $x \mapsto y$ (x adjacent to y) it means that there is an edge from x to y. We can also define the set of successors and predecessors for each pixel from the adjacency relationship.

Definition 1. *Let $(\Omega, \mapsto)$ form a DAG. We define the set of successors of a set $X \in \mathcal{P}(\Omega)$ as*

$$\delta(X) = \{y \in \Omega \mid x \mapsto y \text{ for some } x \in X\}. \tag{1}$$

The set of predecessors can similarly be defined as

$$\hat{\delta}(X) = \{y \in \Omega \mid y \mapsto x \text{ for some } x \in X\}. \tag{2}$$

Definition 2. *Let $\mathbf{a} = (a_1, a_2, ..., a_k)$ be a k-tuple of pixels, then $\mathbf{a}$ is called a path of length k iff $a_i \mapsto a_{i+1}$ for all $i \in [1, k-1]$.*

The set of elements in a path $\mathbf{a}$ is denoted by $\sigma(\mathbf{a})$. We can now define the concept of a path opening.

Definition 3. *Let Π_k be the set of all paths of length k, and let X be the foreground image. Then the path opening is defined by*

$$\alpha_k(X) = \bigcup\{\sigma(\mathbf{a}) \mid \mathbf{a} \in \Pi_k \text{ and } \sigma(\mathbf{a}) \subseteq X\}. \tag{3}$$

The path opening α_k gives the union of all sets of elements in k-tuple paths contained in X (see Fig. 2). It can be established that it is indeed an opening, i.e., it is increasing, anti-extensive and idempotent [4]. It is important to note that the final result is typically the union of the path openings for a set of different DAGs. However, this is immaterial to our discussion, so we will not stress this point further (a typical set of DAGs used is illustrated in Fig. 1).

The path opening can also be defined in an alternative manner. To this end, define the *opening transform* $\lambda_X : \Omega \to \mathbb{N}$ as the mapping that gives the maximum length of all paths restricted to X that visit a given pixel in Ω (so $\lambda_X(x) = 0$ for any $x \notin X$). Here $\mathbb{N}$ denotes the set of all non-negative integers. Using λ_X we then create the following definition of a path opening:

$$\alpha_k(X) = \{x \in X \mid \lambda_X(x) \geq k\}. \tag{4}$$

This simply preserves those pixels satisfying the path length criterion. Since only path *lengths* are important in the path opening, we can efficiently compute a path opening without keeping track of the paths themselves, as shown below.

In the greyscale case, we conceptually apply the binary algorithm to every upper level set. This can be expressed using the greyscale opening transform $\lambda_f : \Omega \times \mathbb{R} \to \mathbb{N}$, which returns the maximum path length for a certain position and threshold in a greyscale image $f : \Omega \to \mathbb{R}$.

3 Sizing up the opening transform

The time complexity of an algorithm is always bounded from below by the space complexity of its output. After all, it has to have the time to construct this output. The algorithms for path openings that we discuss here (and that have been discussed, to our knowledge, in the literature) all output the full opening transform, or are all capable of outputting the opening transform (without this affecting their time complexity). In fact, we will see that their time complexities can be given solely in terms of the size of the opening transform. Theorem 1 and Corollary 1 give bounds for the size of the opening transform, and by extension for the time complexities of the presented algorithms.

Suppose we have a greyscale image $f : \Omega \to \mathbb{R}$ and the associated opening transform $\lambda_f : \Omega \times \mathbb{R} \to \mathbb{N}$. How much data is necessary to represent this opening transform? If we just look at a certain position x, then the mapping $\lambda_f(x) : \mathbb{R} \to \mathbb{N}$ given by $l \mapsto \lambda_f(x, l)$ can be seen to be weakly decreasing. That is, as the threshold level goes up, the maximum path length must go down (because the upper level sets become smaller). This means $\lambda_f(x)$ can be *represented* using any set $\Lambda_f(x) \in \mathcal{P}(\mathbb{R} \times \mathbb{N})$ of pairs of grey levels and their associated maximum path lengths, such that

$$\lambda_f(x, l) = \sup\{\lambda \mid (l', \lambda) \in \Lambda_f(x) \text{ and } l' \geq l\}. \tag{5}$$

When the set over which a supremum is computed is empty, the result is taken to be zero (there is no path at this position and threshold level). It should be clear that if $\Lambda_f(x)$ and $\Lambda'_f(x)$ are two sets satisfying Eq. (5) (so both give rise to the correct $\lambda_f(x)$), then $\Lambda_f(x) \cap \Lambda'_f(x)$ must also satisfy Eq. (5). In fact, it can be shown that this is true for the intersection of any set of sets that satisfy Eq. (5). We can thus speak of the smallest set of pairs of grey levels and maximum path lengths that satisfies Eq. (5), and in the remainder we will assume that $\Lambda_f(x)$ is in fact this smallest set of pairs. This means that it cannot contain two pairs with the same grey level or maximum path length. With a finite number of pixels $|\Omega|$, we can now bound the space needed to represent the opening transform.

Theorem 1. *If L is the set of grey levels in $f : \Omega \to \mathbb{R}$ and d is the maximum path length in the DAG given by '$\mapsto$' on Ω, then the total number of pairs in $\Lambda_f : \Omega \to \mathcal{P}(\mathbb{R} \times \mathbb{N})$ is bounded from below by $|\Omega|$ and from above by the class $O(\min(d, |L|)\, |\Omega|)$. Both bounds can be reached.*

Proof. The lower bound follows from the fact that for each position x the path length at threshold $f(x)$ is greater than zero, implying that $\Lambda_f(x)$ always contains at least one pair. For the upper bound we simply prove that the number of pairs in $\Lambda_f(x)$ is less than or equal to $\min(d, |L|)$ for all $x \in \Omega$, the statement then follows by multiplying this bound by $|\Omega|$. That $|\Lambda_f(x)|$ is less than or equal to the number of grey levels follows from the fact that we cannot have two pairs in $\Lambda_f(x)$ with the same grey level. Similarly, we cannot have more than d pairs in $\Lambda_f(x)$, since we cannot have more than d distinct positive integer path lengths less than or equal to d (zero path lengths would not occur explicitly in $\Lambda_f(x)$).

To see that the lower bound can be reached, just consider a constant image. There is then exactly one pair in $\Lambda_f(x)$ for all $x \in \Omega$. To prove that the upper bound can be reached, consider an image that consists of a sequence of rows with strictly increasing grey levels (but constant within each row), with edges (only) between adjacent rows (see Fig. 1). In this case the pixels on the first row have one pair in $\Lambda_f(x)$, the pixels on the second row two pairs, and so on, for a total numbers of pairs in $\Theta(d\,|\Omega|)$. If the grey levels stay constant after $|L| \leq d$ rows, the total number of pairs is in $\Theta(\min(d, |L|)\,|\Omega|)$. □

The above result shows that even on images with high bit depths, the size of the path opening transform will typically not be quadratic in the number of pixels, as d tends be $O(\sqrt{|\Omega|})$. On the other hand, for low bit depths the size will be linear in the number of pixels (albeit with a potentially high constant). Still, the resulting space complexity *can* be worse than linear. One optimization that has been applied in the literature is to consider all path lengths above a certain threshold to be equivalent. If we know the path length threshold we will be interested in, or at least some upper bound t, then we can define $\lambda^t_f(x, l) = \min(\lambda_f(x, l), t)$ and the associated Λ^t. Crucially, Λ^t would contain at most one pair with a path length greater than or equal to t (as it is known that the path length will be even greater for lower grey levels).

Corollary 1. *If L is the set of grey levels in f, d is the maximum path length in the DAG given by '$\mapsto$' on Ω, and $t > 0$ is the maximum path length threshold, then the total number of pairs in $\Lambda^t_f : \Omega \to \mathcal{P}(\mathbb{R} \times \mathbb{N})$ is in $O(\min(t, d, |L|)\,|\Omega|)$.*

Proof. This follows from Theorem 1, except that we can now also constrain the number of positive path lengths by t: at most $t - 1$ positive path lengths less than t and at most one path length greater than or equal to t. □

If we were to allow non-integer path lengths we get a bound in $O(|L|\,|\Omega|)$, but a lot of work on path openings does use integer path lengths. In either case, the reader might wonder whether it is possible to give better bounds. As shown above, it is possible to provide examples for which the path opening actually does require the amount of space suggested by the bounds derived above. On the other hand, we would expect a typical image to require much less storage, so there is definitely some room for making the above bounds more precise. Also, although it seems doubtful that much could be improved in terms of representing the opening transform at a specific position, it is definitely not beyond the realm of possibility that there exists a more efficient representation of the opening transform as a whole. We have not found one, however.

4 Algorithms

In this section we present three algorithms: the traditional binary algorithm, Talbot's algorithm, and our new stack-based algorithm.

4.1 Binary images

An algorithm which computes the binary path opening efficiently is given by Heijmans et al. [4]. The idea is to do two sequential scans on the binary image, computing the largest path length up to each pixel in opposite directions, and then combining these results to compute λ. In the first scan we traverse all the

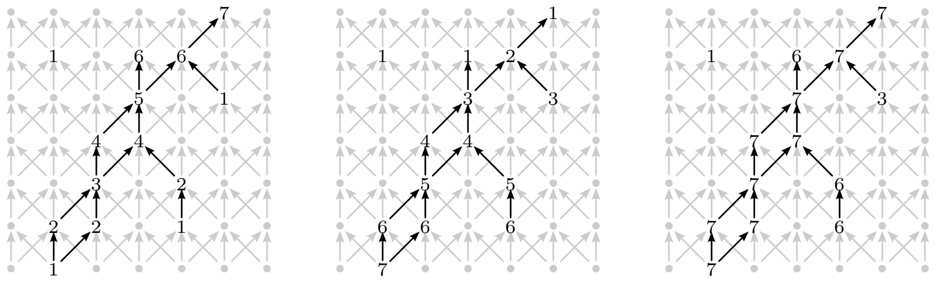


Fig. 3. Computation of λ in the example shown in Fig. 2. From left to right: the forward scan pass λ^{+}, the backward scan pass λ^{-}, the calculated length per pixel λ.

rows (or columns, or ...) of the image from top to bottom. Let $\lambda_{+} : \Omega \to \mathbb{R}$ be the map which gives the maximum path length for each pixel $x \in X$ based only on its *predecessors*. To compute such a map we use the following relations [4] (only for $x \in X$, for other pixels the λ's are set to zero):

$$\lambda_{+}(x) = \max_{y \in X \mid y \mapsto x} \lambda_{+}(y) + 1, \qquad \lambda_{-}(x) = \max_{y \in X \mid x \mapsto y} \lambda_{-}(y) + 1. \tag{6}$$

It was shown by Heijmans et al. [4] that by combining $\lambda_{+}(x)$ and $\lambda_{-}(x)$, we can recover the maximum path length using the following relation (for $x \in X$):

$$\lambda(x) = \lambda_{+}(x) + \lambda_{-}(x) - 1. \tag{7}$$

This notion is intuitive, as $\lambda_{+}(x)$ holds the maximum path length of all paths ending in x, and $\lambda_{-}(x)$ holds the maximum path length of all paths starting in x, so by combining them we recover the maximum path length through x. We subtract one from the sum of the two partial path lengths to avoid counting pixel x twice. Figure 3 shows an example of the computation of λ.

The above can be easily turned into an actual algorithm by topologically sorting [5] the DAG and then applying Eq. (6) in (reverse) topological order. This way the λ_{+} and λ_{-} only need to be set once for each pixel.

4.2 Talbot's algorithm

If we apply the algorithm described in the previous section to all upper level sets of an image $f : \Omega \to \mathbb{R}$, then we can find the path opening (transform) by combining all those results. However, this could be quite expensive. Luckily, as described by Talbot and Appleton [11], the algorithm described above can be modified to only update λ_{+} and λ_{-} based on the changes resulting from going to one grey level to the next. This works by first sorting all grey levels and initializing all $\lambda_{\pm}$ to the largest possible path length, and then applying Algorithm 1 for each grey level (and an analogous algorithm for λ_{-}), in increasing order of grey level. For each grey level, after λ_{+} and λ_{-} are updated, the algorithm goes through all the affected pixels and records any changes in per-pixel lists to

Algorithm 1: Update of λ_+.

```
Input  : $\lambda_+$ for the current grey level.
Output: $\lambda_+$ for the next grey level.
Initialize the priority queue $Q$ with all pixels at the current grey level.
  /* Priorities compatible with the topological order of the DAG. */
while $Q$ not empty do
    remove smallest pixel $x$ from $Q$
    $\lambda \leftarrow 0$
    if $f(x)$ above current grey level then
        $\lambda \leftarrow \max_{y \in X | y \mapsto x} \lambda_+(y) + 1$
    if $\lambda < \lambda_+(x)$ then
        $\lambda_+(x) \leftarrow \lambda$
        push successors of $y$ onto $Q$
```

build up the opening transform. Although presented somewhat differently from the original, we will call the resulting algorithm “Talbot’s algorithm”. Instead of computing the entire opening transform, it can also directly compute the opening (but this does not affect the time complexity).

Theorem 2. *Assume each position has $O(1)$ predecessors/successors (the DAG induced by '$\mapsto$' is sparse), and that the priority queue in Algorithm 1 allows insertion and removal in $O(\log(|Q|))$. Talbot’s algorithm then has a time complexity of $O(|\Lambda_f| \log(|\Lambda_f|))$, where $|\Lambda_f|$ is taken to mean the total number of pairs needed to represent Λ_f and $f : \Omega \to \mathbb{R}$ is the input image.*

Proof. Disregarding the time needed to sort the grey levels, it can be seen that asymptotically the running time is determined by the work done in Algorithm 1 (the amount of work done by the rest of the algorithm is dominated by the amount of work done in the update steps).

The crucial observation is now that $\Lambda_f(x)$ has a pair with the current grey level if and only if x is processed when updating λ_+ and/or λ_- *and* while doing so, the condition on Line 9 is true (or the analogous condition for λ_-). Each time this happens $O(1)$ elements are pushed onto Q (due to the sparse graph assumption), so if we look at all applications of Algorithm 1, the total number of queue pushes and (thus) executions of the while loop must be in $\Theta(|\Omega|+|\Lambda_f|) = \Theta(|\Lambda_f|)$. Assuming a priority queue with $O(\log(|Q|))$ insertion and removal, the total amount of work done is then in $O(|\Lambda_f| \log(|\Lambda_f|))$. Finally, we conclude that sorting all grey levels requires at most $O(|\Omega| \log(|\Omega|))$ time, so it does not alter the time complexity. □

The assumption that the DAG is sparse is fairly benign, as all existing use cases (to our knowledge) satisfy this assumption. That the priority queue allows $O(\log(|Q|))$ insertion and removal is also fairly standard. In some cases (like the typical DAGs used on images), it is even possible to get constant-time insertion and removal, by grouping pixels into “rows” based on their depth in the DAG.

We conjecture that in general it may be possible to get (amortized) constant-time insertion and removal by using specialized data structures. In combination with linear time sorting of all (integer) grey levels, this would put the time complexity of Talbot's algorithm in $\Theta(|\Lambda_f|)$.

It should be noted that some tweaks to Talbot's algorithm can further reduce the time complexity by essentially restricting the opening transform as in Corollary 1, as well as ignoring pixels whose path length dropped *below* a certain threshold. It is currently not entirely clear how this affects the time complexity of the algorithm. Also, although Luengo Hendriks [6] presents a modification of Talbot's algorithm that is dimensionality independent, it does not necessarily process the pixels in optimal order, leading us to start from Talbot's algorithm instead (note that our presentation of Talbot's algorithm is also dimensionality independent). Similarly, if we look at Talbot's code, it loops over *all* rows/columns in each update step, while our code only visits those rows/columns where it has to do something. We do not expect these implementation differences to make a huge difference in performance, but it should be understood that due to these differences actual implementations might have slightly different time complexities from the one presented here.

4.3 Stack-based path openings

Talbot's algorithm is essentially optimal in terms of its time complexity, but a truly optimal implementation (without the logarithmic factor) can be quite complex, and the algorithm has a fairly random memory access pattern, which is undesirable in most modern computer architectures. In this section we present an algorithm that suffers from none of these problems, based on the 1D algorithm presented by Morard et al. [9].

The basic idea is to use the traditional binary algorithm, but instead of having the scalar $\lambda_+(x)$ and $\lambda_-(x)$, we use sets $\Lambda_+(x)$ and $\Lambda_-(x)$, represented by non-redundant ordered lists of pairs of grey levels and path lengths. The 1D algorithm only needs to push and pop elements, so can use a traditional stack. Our algorithm does the same, but also needs to merge lists. We will still refer to the lists as stacks though. Algorithm 2 details the algorithm needed to compute Λ_+. Λ_- is computed in much the same way in a second pass over the data, and since all the lists are already sorted, merging Λ_+ and Λ_- to get the final opening transform can be done easily and efficiently. Note that the merge procedures should leave their output sorted and without any redundant pairs. This can be accomplished using a technique similar to the one used in merge sort.

Theorem 3. *Assume each position has $O(1)$ predecessors/successors (the DAG induced by '$\hookrightarrow$' is sparse), the stack-based path opening then has a time complexity in $\Theta(|\Lambda_f|)$, where $f : \Omega \to \mathbb{R}$ is the input image.*

Proof. First of all topological sorting can be done in $O(|\Omega|)$ [5], so will not be a bottleneck. Similarly, merging the two partial path opening transforms into the final answer just requires iterating over the partial transforms (once), so also

Algorithm 2: Computation of Λ_+.

```
Input  : The input image $f$.
Output: The partial opening transform $\Lambda_+$.
1 for $x$ in $\Omega$ in topological order do
2     $\Lambda_+^{temp} \leftarrow \text{merge}(\{\Lambda_+(y) \mid y \mapsto x\})$
3     $\lambda_+^{temp} \leftarrow \max(\{0\} \cup \{\lambda \mid (l', \lambda) \in \Lambda_+^{temp} \text{ and } l' \geq f(x)\}) + 1$
4     $\Lambda_+(x) \leftarrow \{(l, \lambda + 1) \mid (l, \lambda) \in \Lambda_+^{temp} \text{ and } l < f(x)\} \cup \{(f(x), \lambda_+^{temp})\}$
```

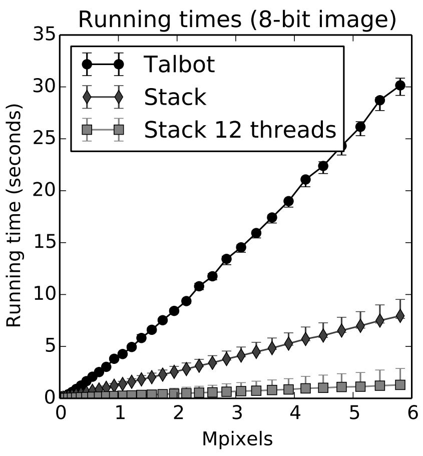
does not add anything to the time complexity. It remains to examine the time spent in Algorithm 2.

Algorithm 2 visits each pixel exactly once. For each pixel it first merges the partial transforms (stacks) from its predecessors (successors if computing Λ_-), then it computes $\lambda_+^{temp} = \lambda_+(x, f(x))$, and finally it makes sure $f(x)$ is the highest grey level in the new stack, while updating path lengths for grey levels below $f(x)$. Since the number of stacks being merged in Line 2 is in $O(1)$, we can assume the merge procedure to take time linear in its input. Since each stack is involved in $O(1)$ merges, the *total* time taken up by all merges (in both passes) is in $O(|\Lambda_+| + |\Lambda_-|)$. Lines 3 and 4 can be implemented together with an overall time complexity in $\Theta(|\Lambda_\pm(x)|)$. Summarizing, the total amount of work done by Algorithm 2 (in both passes, one for Λ_+ and one for Λ_-) is in $\Theta(|\Lambda_+| + |\Lambda_-|)$.

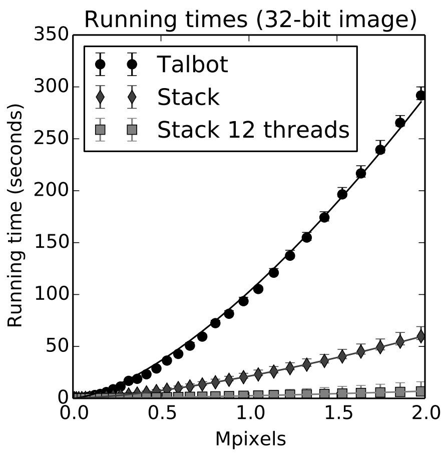
We now note that $|\Lambda_f| \leq |\Lambda_+| + |\Lambda_-| \leq 2\,|\Lambda_f|$. The lower bound follows from the fact that (by definition) every pair in Λ_f must correspond to a pair in Λ_+ or Λ_-. The upper bound can be shown similarly, by considering that every pair in Λ_+ and Λ_- must correspond to a pair in Λ_f. In particular, because of the monotonicity of the (partial) path opening transforms we cannot have a pair in Λ_+ "cancel out" a pair at the same grey level in Λ_-. We can now conclude that the time complexity of the stack-based path opening is indeed in $\Theta(|\Lambda_f|)$. $\square$

The optimization referred to in Corollary 1 can be applied to the stack-based algorithm very easily, preserving the output sensitivity of the algorithm. The other optimization applied by Talbot and Appleton [11] seems to be harder to apply to the stack-based algorithm though. The problem is that this optimization discards any points whose *total* path length drops below the desired threshold, and in the stack-based algorithm we only have access to the total path length (for any grey level) after all computations have been done. For now it is not clear how this effects the time complexity of the algorithm. In terms of the space complexity, Talbot's algorithm is the clear winner though, as the stack-based algorithm always has to build at least part of the opening transform.

The stack-based algorithm will process an image row by row (instead of "row" one can also read column, or diagonal), and within each row the results only depend on the previous row. This allows us to compute the values within a single row in parallel. This kind of parallelization is somewhat limited by needing a synchronization point after each row, but as the next section demonstrates, it still allows for a very decent speedup using a small number of cores. Applying this



(a)



(b)

Fig. 4. Running times of the Talbot algorithm, and the stack-based opening (both generating the full opening transform). (a) On an 8-bit image, and (b) on an artificial 32-bit image representing a gradient. Code available at `http://bit.ly/1BTC2Je`.

technique to Talbot's algorithm would be possible, but would be complicated by not knowing beforehand what pixels need to be processed in each row. Also, this would involve (even) more synchronization, as Talbot's algorithm uses multiple (simpler) passes rather than one pass (per direction). It would of course also be possible to compute Λ_+ and Λ_- independently, as well as to process each of the directions independently, but these measures would apply equally well to either algorithm and are not evaluated here.

5 Results

To assess the performance of the path openings, we created an application in C++ which implements all of the previously discussed algorithms. In particular, we implemented the Talbot opening as discussed in Section 4.2, and the newly introduced stack opening as discussed in Section 4.3. Although we use our own implementation of Talbot's algorithm, it was verified that our implementation has similar or better performance. Although all the algorithms are applicable to arbitrary DAGs, we decided to implement the algorithms using the graphs illustrated in Fig. 1. Note that in the interest of simplicity we only used the horizontal and vertical ones, while Talbot originally included the diagonal ones as well. For the purposes of seeing which implementation was faster the diagonal ones were disabled in Talbot's implementation. This decision does not affect the above analysis in any way (since it holds for arbitrary sparse DAGs), and we also do not expect any major influence on the results in this section either (our parallelization strategy could be slightly less efficient on these other DAGs).

The algorithms were tested for their performance on an Intel® Xeon® E5-2630 with 32 logical cores and 512 GB of memory. The tests measure the performance of the complete opening transform. The differently sized images were generated by downscaling the test images using bilinear interpolation. Both 8-bit images and 32-bit images were assessed, where the 32-bit image is created from a gradient image where all the pixel values are strictly increasing from top to bottom and left to right. This allows us to see the worst-case behaviour of the algorithms and confirm our bound. The running times of the stack-based algorithm and Talbot's algorithm are shown in Fig. 4.

On sparse graphs, both the stack opening and the Talbot opening have a time complexity of $O(\min(d, |L|)\,|\Omega|)$, where $|L| \leq 256$, so we expect (roughly) linear behaviour. This is confirmed by Fig. 4a. Figure 4b, however, shows superlinear behaviour. This is expected, as the number of grey values is no longer the limiting factor. Since d is is equal to the width or the height of the image, the time complexity should be in $O(n\sqrt{n})$ (with n the number of pixels $|\Omega|$). The function $f(n) = c\,n\sqrt{n}$ was fitted to the 32-bit results using least squares, where c is the fitted parameter. We indeed see roughly $n\sqrt{n}$ behaviour in the results.

Both in the single-threaded case and in the multi-threaded case, the stack-based path opening outperforms Talbot's algorithm, by a factor of roughly 4 (or 25 using twelve threads). This is likely because of the data locality of the algorithm, as the algorithm walks more or less sequentially through the image, rather than having to reprocess certain sections repeatedly.

6 Conclusion

We have shown that the space complexity of the path opening transform is in $O(\min(d, |L|)\,|\Omega|)$, with d the depth of the graph being processed, L the set of grey levels in the image, and Ω the image domain. We also showed that although there might be room for refinement, it is possible to construct graphs that reach this bound. Next we analysed the time complexity of a paraphrased version of the algorithm proposed by Talbot and Appleton [11], and found that it is (depending on the exact implementation) optimally output-sensitive, assuming the full opening *transform* is output. Finally, we presented a new algorithm that is easier to implement (at least in optimal fashion), is still optimally output-sensitive, allows for easier parallelization, and is significantly faster in practice. We presented results demonstrating our new algorithm outperforming an implementation of Talbot's algorithm by a factor of roughly 4 (and 25 when using parallelization on 12 cores). We also experimentally demonstrated that for high bit-depth images the performance of the algorithms can indeed scale superlinearly.

In future work it would be interesting to take a further look at various optimizations that one can apply if only part of the opening transform (or indeed just the actual opening) is needed. It would also be interesting to see whether the opening transform can be stored more efficiently than we propose here, and if so, whether this can actually lead to faster algorithms. Additionally, it would be interesting to try adapting the stack-based algorithm to robust [3] and incomplete

[4] path openings, or other schemes for making path openings more robust to noise. Different schemes for parallelization could also be explored (for example by dividing the grey levels, rather than pixels, among different processors).

References

[1] Appleton, B., Talbot, H.: Efficient Path Openings and Closings. In: Ronse, C., Najman, L., Decencière, E. (eds.) Mathematical Morphology: 40 Years On, Computational Imaging and Vision, vol. 30, pp. 33–42. Springer Netherlands (2005)

[2] Bismuth, V., Vaillant, R., Talbot, H., Najman, L.: Curvilinear Structure Enhancement with the Polygonal Path Image - Application to Guide-Wire Segmentation in X-Ray Fluoroscopy. In: Ayache, N., Delingette, H., Golland, P., Mori, K. (eds.) Med. Image Comput. Comput. Assist. Interv., LNCS, vol. 7511, pp. 9–16. Springer Berlin Heidelberg (2012)

[3] Cokelaer, F., Talbot, H., Chanussot, J.: Efficient Robust d-Dimensional Path Operators. IEEE J. Sel. Top. Signal. Process. 6(7), 830–839 (2012)

[4] Heijmans, H., Buckley, M., Talbot, H.: Path Openings and Closings. J. Math. Imaging Vis. 22(2), 107–119 (2005)

[5] Kahn, A.B.: Topological Sorting of Large Networks. Commun. ACM 5(11), 558–562 (1962)

[6] Luengo Hendriks, C.L.: Constrained and dimensionality-independent path openings. IEEE Trans. Image Process. 19(6), 1587–1595 (2010)

[7] Morard, V., Decencière, E., Dokládal, P.: Geodesic Attributes Thinnings and Thickenings. In: Soille, P., Pesaresi, M., Ouzounis, G.K. (eds.) Mathematical Morphology and Its Applications to Image and Signal Processing, LNCS, vol. 6671, pp. 200–211. Springer Berlin Heidelberg (2011)

[8] Morard, V., Dokládal, P., Decencière, E.: Linear openings in arbitrary orientation in O(1) per pixel. In: IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). pp. 1457–1460. IEEE (2011)

[9] Morard, V., Dokládal, P., Decencière, E.: One-Dimensional Openings, Granulometries and Component Trees in O(1) Per Pixel. IEEE J. Sel. Top. Signal. Process. 6(7), 840–848 (2012)

[10] Morard, V., Dokládal, P., Decencière, E.: Parsimonious Path Openings and Closings. IEEE Trans. Image Process. 23(4), 1543–1555 (2014)

[11] Talbot, H., Appleton, B.: Efficient complete and incomplete path openings and closings. Image Vis. Comput. 25(4), 416–425 (2007)

[12] Valero, S., Chanussot, J., Benediktsson, J.A., Talbot, H., Waske, B.: Advanced directional mathematical morphology for the detection of the road network in very high resolution remote sensing images. Pattern Recognit. Lett. 31(10), 1120–1127 (2010)