Efficient Computation of Greyscale Path Openings

Schubert, Herman; van de Gronde, Jasper J.; Roerdink, Johannes

Published in:
Mathematical Morphology - Theory and Applications

DOI:
10.1515/mathm-2016-0010

Research Article **Open Access**

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

Efficient Computation of Greyscale Path Openings

DOI 10.1515/mathm-2016-0010
Received June 30, 2015; accepted February 9, 2016

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 similar structures. 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 [3], roads in remote sensing images [18], and cracks for non-destructive testing [13]. A possible way to do this is to apply openings using fixed line structuring elements [14]. 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 [8] solve these issues by looking for paths in a small number of purpose-made directed acyclic graphs (DAGs) (see Fig. 1), or (more recently) a single directed graph (not necessarily acyclic) [5].

Path openings were originally introduced by Heijmans et al. [8], 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 [17] improved on this by only looking at the differences between adjacent 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. [16] propose using 1D "path" openings on a carefully selected subset of possible paths – whose selection was recently improved upon by Asplund [2] – to speed up path openings, but this only gives an approximation of the full path opening.

Despite some educated guesses [1, 4, 17] (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

***Corresponding Author: Herman Schubert, Jasper J. van de Gronde:** University of Groningen, E-mail: h.robert.schubert@gmail.com, j.j.van.de.gronde@rug.nl
Jos B. T. M. Roerdink: University of Groningen, E-mail: j.b.t.m.roerdink@rug.nl

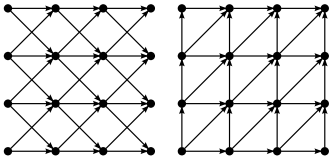

Figure 1: Illustration of the commonly used DAGs for path openings.

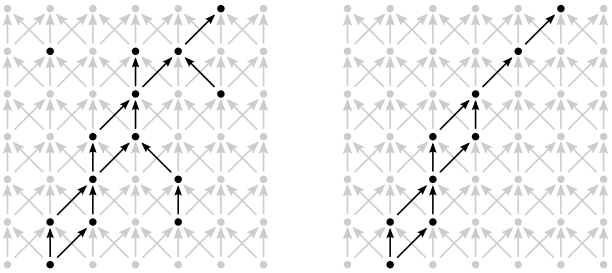

Figure 2: A set $X \subseteq \Omega$ (black points on the left) and its path opening with $k = 7$ (black points on the right). Points only contained in paths of length 6 or less have been discarded (also see Fig. 3). The underlying adjacency graph is light grey, with black arrows highlighting the edges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

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.

This is an extended version of our earlier paper [7]. This paper has the following improvements over the preliminary version: i) a more extensive empirical evaluation, including data on the runtime of the dimensionally independent path opening [11], ii) an extended discussion regarding the implementation of the stack opening, overcoming possible memory issues and iii) a continued discussion on the prospects for (even) faster algorithms. We have also described the presented algorithms in more detail.

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 Ω be the set of foreground pixels and 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 $\mathbf{a} = (a_1, a_2, \ldots, a_k)$ 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 [8]. 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 number 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 $O(|\Omega|^{1/D})$, with D the dimension of the image domain. 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 $\lambda_f^t(x, l) = \min(\lambda_f(x, l), t)$ 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 $\Lambda_f^t : \Omega \to \mathcal{P}(\mathbb{R} \times \mathbb{N})$ 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.

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

Heijmans et al. [8] give an algorithm which efficiently computes binary path openings. 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 rows (or columns, or diagonals) 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*, and analogously let $\lambda_- : \Omega \to \mathbb{R}$ be the map which gives the maximum path length based only on its *successors*. To compute these maps we use the following relations [8] (only for $x \in X$, for other pixels the λ's are set to zero):

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

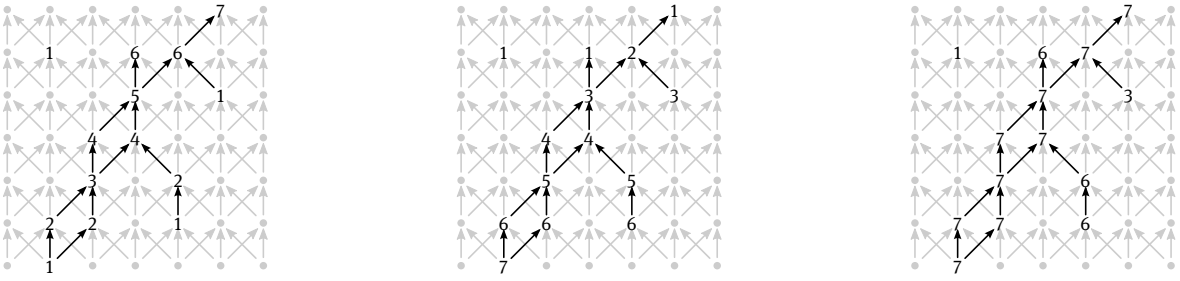

Figure 3: Computation of λ in the example shown in Fig. 2. From left to right: the forward scan pass λ^+, the backward scan pass λ^-, the calculated length per pixel λ.

Algorithm 1: Talbot's algorithm, adapted to compute the opening transform.

Input : The input image f.
Output: The opening transform Λ_f.

1 Initialize λ_+ and λ_- to the largest possible path length.
2 Initialize Λ_f to have a stack at each position x, containing a pair $(\infty, \lambda_+(x) + \lambda_-(x) - 1)$.
3 Sort grey levels L, and partition the pixels into flat zones.
4 **for** $l \in L$ *in ascending order* **do**
5 $B \leftarrow \emptyset$
6 Use Algorithm 2 to update λ_+ and λ_-, and maintain the bag B of affected pixels.
`// The result reflects the removal of all pixels at the current grey level.`
7 **for** $b \in B$ **do**
`//` λ_+ `and/or` λ_- `have been decreased as a consequence of removing pixels at the current grey level.`
8 pop (l', λ) from $\Lambda_f(b)$
9 push (l, λ) onto $\Lambda_f(b)$
10 $\lambda \leftarrow \lambda_+(b) + \lambda_-(b) - 1$
11 **if** $\lambda > 0$ **then**
12 push (∞, λ) onto $\Lambda_f(b)$

It was shown by Heijmans et al. [8] 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 [9] 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 [17], the algorithm described above can be modified to only update λ_+ and λ_- based on the changes resulting from going from one grey level to the next, as in Algorithm 1. Although presented somewhat differently from the original, we will call the resulting algorithm

Algorithm 2: Update of λ_+.

Input : λ_+ for the current grey level, and a bag B of pixels.
Output: λ_+ for the next grey level, and the bag B, augmented with all affected pixels.

1 Initialize the priority queue Q with all pixels at the current grey level.
`// Priorities compatible with the topological order of the DAG.`
2 **while** *Q not empty* **do**
3 remove smallest pixel x from Q
4 $\lambda \leftarrow 0$
5 **if** *$f(x)$ above current grey level* **then**
6 $\lambda \leftarrow \max_{y \in X | y \mapsto x} \lambda_+(y) + 1$
7 **if** $\lambda < \lambda_+(x)$ **then**
8 insert x in B
9 $\lambda_+(x) \leftarrow \lambda$
10 push successors of y onto Q

"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 2 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 2 (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 7 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 2, 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 [11] presented 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 some work. We do not expect these implementation differences to make a huge difference in performance, but it should be understood that due to these differ-

Algorithm 3: Computation of Λ_+.

Input : The input image f.
Output: The partial opening transform Λ_+.

for *x in Ω in topological order* **do**
    $\Lambda_+^{temp} \leftarrow \text{merge}(\{\Lambda_+(y) \mid y \mapsto x\})$ `// See Algorithm 4.`
    $\lambda_+^{temp} \leftarrow \max(\{0\} \cup \{\lambda \mid (l', \lambda) \in \Lambda_+^{temp} \text{ and } l' \geq f(x)\}) + 1$
    $\Lambda_+(x) \leftarrow \{(l, \lambda + 1) \mid (l, \lambda) \in \Lambda_+^{temp} \text{ and } l < f(x)\} \cup \{(f(x), \lambda_+^{temp})\}$

Algorithm 4: Merge two stacks from the same direction.

input : Partial transforms Λ^A and Λ^B.
output: The merged partial transform Λ^C.

$\Lambda^C \leftarrow$ Empty opening transform
$\lambda \leftarrow \lambda_{front} \leftarrow 0$
`// Iterate over` $\Lambda^A \cup \Lambda^B$ `in decreasing order of grey level.`
`// If two pairs have the same grey level, order by decreasing path length as well.`
for $(l, \kappa) \in \Lambda^A \cup \Lambda^B$ **do**
    $\lambda \leftarrow \max(\lambda_{front}, \kappa)$
    `// Only add to the current transform if the length is not redundant.`
    if $\lambda > \lambda_{front}$ **then**
        push (l, λ) onto the front of Λ^C
        $\lambda_{front} \leftarrow \lambda$
return Λ^C

ences actual implementations might have slightly different time complexities from the algorithm 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. [15].

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 (ascending by grey level) 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 3 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 (see Algorithm 4 for example).

Theorem 3. *Assume each position has $O(1)$ predecessors/successors (the DAG induced by '$\mapsto$' 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 $O(|\Omega|)$ [9], 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 does not add anything to the time complexity. It remains to examine the time spent in Algorithm 3.

Algorithm 3 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 3 (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 derived 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 $\Theta(|\Lambda_f|)$. □

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 [17] 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 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. The latter scheme applies equally well to either algorithm and is not evaluated here. Processing the two directions independently on the other hand is easier with the stack-based algorithm, unless Talbot's algorithm is modified to only output Λ_+ (or Λ_-).

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. We performed two experiments: 1) examining the performance of the full opening transforms on 8-bit and 32-bit images as a function of the image size, and 2) comparing the performance of regular path openings using different implementations. For the first experiment we have used our own implementation of both algorithms, as the implementations of the authors do not output full opening transforms. For the second experiment we also use the implementations of the original authors, including the dimensionally-independent version of

a b

Figure 4: A 200 mega-pixel map of London. This image which was used for our benchmarks for Fig. 6a. (a) The full map of London, and (b) zooming in on part of the map.

a b

Figure 5: Two sample images of an aerial photo database from [19]. These images as well as others from the same database are used for our benchmarks for Fig. 7.

Talbot's algorithm introduced by Luengo Hendriks [11], as well as our own implementations. In the first experiment we use the map of London listed in Fig. 4, but for the second experiment we use a database of aerial images as shown in Fig. 5.

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 graphs for the first experiment (where we compute opening transforms), while Talbot originally included the diagonal ones as well. This decision does not affect the above analysis in any way (since it holds for arbitrary sparse DAGs). In the second experiment we both show results for the horizontal and vertical graphs as well as the diagonal ones. Where applicable, the diagonal graphs were disabled in both Talbot's and Luengo Hendriks' implementation.

The algorithms were tested for their performance on an Intel® Xeon® E5-2630 with 16 logical cores and 64 GB of memory (note that this is a different setup than in our previous paper [7]).

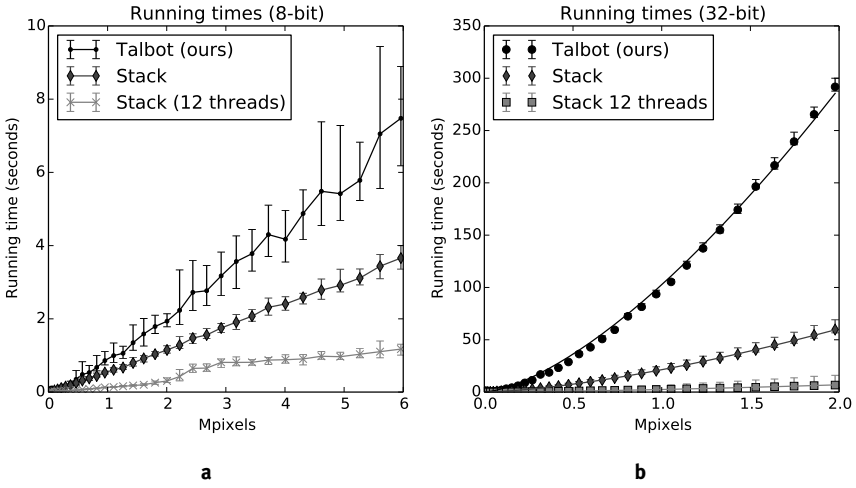

Figure 6: Running times of the Talbot algorithm, and the stack-based opening (both generating the full opening transform). (a) On a set of 8-bit images, and (b) on an artificial 32-bit image representing a gradient. Code available at http://bit.ly/1BTC2Je.

5.1 Opening Transforms

The tests in this section measure the performance of the complete opening transform. Both 8-bit images and 32-bit images were assessed. In the 8-bit case we use random crops of the map of London (see Fig. 4). In the 32-bit case we created 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 different sized images were generated by randomly cropping an image using different scales. This differs from our approach in our previous paper [7], where we used bilinear scaling. We believe that this approach is better, as thinner curvilinear structures are preserved at lower scales, which tend to get removed when bilinear scaling is used.

Both benchmarks use average curves of ten trials (which consist of different cropped images), and the error bars correspond to the minima and maxima at each scale. The running times of the stack-based algorithm and Talbot's algorithm are shown in Fig. 6.

On the graphs listed here, 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. 6a. Figure 6b, however, shows superlinear behaviour. This is expected, as the number of grey values is no longer the limiting factor. Since d 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 (our implementation of) Talbot's algorithm, by a factor of roughly 2 (or 12 using twelve threads). This is likely because of the data locality of the algorithm, as the algorithm processes the image more or less sequentially, rather than having to reprocess certain sections repeatedly.

5.2 Openings

For comparing regular openings, we used Talbot's original implementation [17] as well as our own, Luengo Hendriks' dimensionally independent version [11], and our stack-based implementation. For Figs. 7a and 7b the diagonal paths were disabled to compare the implementations. For the result of Fig. 7c the diagonal graphs as well as the horizontal and vertical graphs are included, and the implementations of the original authors were left unchanged. Note that our implementation of Talbot's algorithm does not support the diagonal graphs and is therefore not included in this result. The openings were tested on a set of 30 images

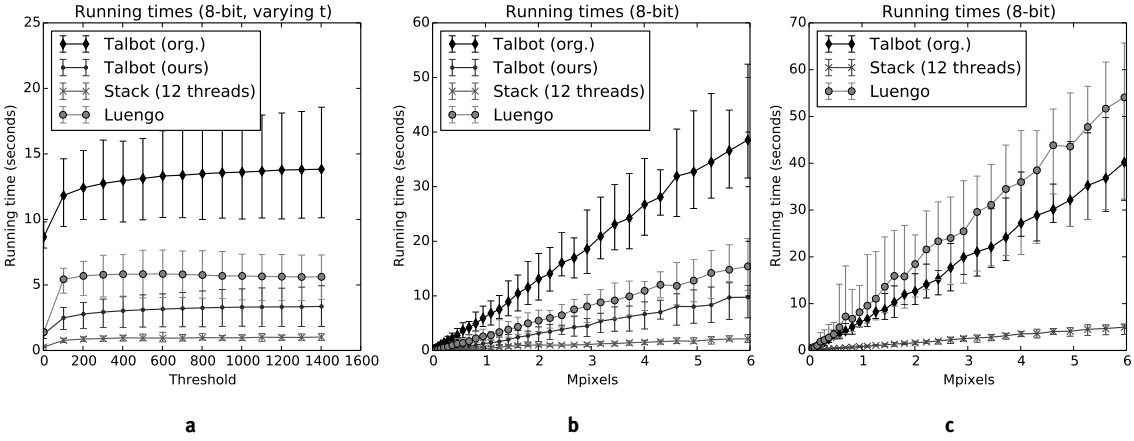

Figure 7: Running times of various path opening algorithms (all applying a regular opening). (a) Varying the threshold on 2 M-pixel 8-bit images on vertical and horizontal graphs; (b) varying the size by using different sized crops of Fig. 5 with $t = \infty$ on horizontal and vertical graphs, and (c) the same as Fig. 7b but then on the diagonal graphs as well as the horizontal and vertical graphs. Code available at http://bit.ly/1BTC2Je.

from a database of Aerial photographs taken from [19]. Instead of taking the raw images, these images were processed by computing its gradient magnitude using the classical Sobel operator, followed by an opening with a disk-shaped structural element. This process highlights curvilinear structures which can henceforth be filtered by path openings. Similar procedures have been used as a preprocessing step for path openings in other works as well [4, 6, 11].

In Fig. 7a we show the behaviour of the implementations when we vary the opening threshold t. In all implementations, lower thresholds give better performance (as expected). This is related to the various optimizations which are used in the implementations. In all implementations of Talbot's algorithm (including ours), path lengths lower than the threshold are no longer processed in the upper level sets. Additionally, all path lengths above a certain threshold are considered to be equivalent. In the stack opening we can only do the second optimization, as we have no information on the complete path lengths when traversing the partial opening transforms. Thus, it makes sense that the stack opening is not as sensitive to changing the opening threshold. On this set of images, the stack opening is the fastest, followed by our Talbot implementation. In practice we noticed that Luengo-Hendriks's algorithm appears to be very fast at low threshold levels ($t < 30$), but is slower than our Talbot implementation when higher threshold levels are used.

We also varied the image size, as is shown in Figs. 7b and 7c. Just as in Fig. 6a we use different sized random crops to create multiple scales of the aerial images. All methods show roughly linear behaviour. This is also true when diagonal graphs are used. This should be no surprise, as the diagonal graphs have a maximum length of $w + h - 1$, where w and h are respectively the width and the height of an image. Therefore the time complexity does not change when diagonal graphs are used instead of vertical or horizontal graphs. Substituting d with $w + h - 1$ in Theorem 1 gives us $O(\min(\max(w, h), |L|)\,|\Omega|)$, where $|L| \leq 256$, which shows linear behaviour at higher scales. In practice, including the diagonal graphs means that roughly twice to four times the amount of work is needed depending on the implementation of the graph traversal method. Interestingly, the ordering of Talbot's algorithm and Luengo-Hendriks's algorithm seems to be swapped when the diagonal graphs are used. This is likely due to the different approaches of traversing the graph in the diagonal case. We note that that Luengo-Hendriks' algorithm shows the most variance. We suspect this is related to the fact that it does not process the updates in topological order, making it far more sensitive to image content.

6 Discussion

6.1 Extension to 3D and overcoming memory bottlenecks

Recently, path openings on voxel volumes have gained some momentum (see [12] and [4]). Voxel volumes typically take up to 1000^3 voxels, and thus also require significantly more memory than common 2D images. Here we discuss some mitigation strategies to prevent memory bottlenecks. First, we would like to note that in the 3D case the opening transform takes up less space per pixel than in the 2D case. Let d be the depth, width and height of a volume image and $n = d^3$ the number of voxels. Then the opening transform takes at most $O(\min(d, |L|)\,|\Omega|) = O(\min(n^{1/3}, |L|))\,n)$ memory. In the 2D case we take the approach to compute each direction in parallel. In the 3D case storing the opening transform for 13 directions (in case of 26-connectivity) might be prohibitively expensive on some systems. Thus it is better to perform the stack opening only one direction at a time, and merge the consecutive partial opening transforms using Algorithm 4. In this case we only have to store 2 partial transforms when processing all directions. This can be simplified even further, one does not need to compute Λ_+ and Λ_- independently. In one pass we can compute Λ_+, and in the reverse pass Λ_+ can be reused to compute the full opening transform $\Lambda_\pm$ in-place. The result is that – assuming the width, height, and depth of the volume are similar in size – the stack opening requires only $O(\min(n^{1/3}, |L|)\,n)$ memory, independent of the number of directions.

6.2 Better bounds

The reader might wonder whether it is possible to give better bounds than the ones above. Well, it is possible to provide examples for which the path opening actually does require the amount of space suggested by our bounds. 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, it is definitely not beyond the realm of possibility that there exists a more efficient representation of the opening transform as a whole. And although we have had no success so far, we still suspect it may be possible to find one. In addition, perhaps knowing the threshold beforehand allows for an algorithm for path *openings* whose time complexity does not depend on the size of the opening transform.

So why do we believe that it might be possible to represent the opening transform in less space than we currently require, while still needing only (roughly) linear time to find an arbitrary opening from the opening transform? A first clue is that neither of the current n-D algorithms gracefully degrade to the linear 1D algorithm [15]. That is, the algorithms discussed here have a time complexity in $O(n^2)$ on 1D sequences of length n. The 1D algorithm only takes linear time because there is only a single stack, and we only need to access it from the top (if we wish to compute the opening transform, we can use functional data structures to avoid needless copying, while still preserving all versions of the stack). In our generalized stack-based algorithm, however, we need to merge stacks, causing a (potentially) massive slowdown. Still, if we are able to speed up the merges (or show that we need less of them), this could provide an enormous speed boost (and also lower the memory requirements).

More speculatively, it might be possible to show that we only need to consider a limited number of paths – or a subset of the original edges – to compute the exact path opening, similar to the approximate parsimonious path openings [2, 16]. One reason that this approach might prove to be interesting is that if one considers a (1D) gradient, this is essentially a worst-case scenario for the methods discussed here, while from the point of view of the number of paths one needs to traverse, it is optimal (just one path). So far we have not been able to find a useful bound on the number of paths needed in general though. Instead of paths, it could be equally useful to consider (directed/rooted) trees contained in the original image, as, at least for computing $\Lambda_\pm$, these also avoid the need for merging stacks (provided the leaves and root are positioned such that Algorithm 3 encounters the root first and the leaves last).

7 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 [17], 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 other algorithms in opening transforms and regular openings. 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. We also wonder whether it would not be possible to find an algorithm for the path opening whose time complexity does not depend on the size of the opening transform. Additionally, it would be interesting to try adapting the stack-based algorithm to robust [4] and incomplete [8] 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), as well as efficient GPU implementations as has been explored in the 1D case [10].

References

[1] Appleton B., Talbot H. Efficient Path Openings and Closings. In C. Ronse, L. Najman, E. Decencière, editors, Mathematical Morphology: 40 Years On, volume 30 of Computational Imaging and Vision, pages 33–42. Springer Netherlands, 2005. 10.1007/1-4020-3443-1_4

[2] Asplund T. Improved Path Opening by Preselection of Paths. Master's thesis, Uppsala Universitet, 2015

[3] 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 N. Ayache, H. Delingette, P. Golland, K. Mori, editors, Med. Image Comput. Comput. Assist. Interv., volume 7511 of LNCS, pages 9–16. Springer Berlin Heidelberg, 2012. 10.1007/978-3-642-33418-4_2

[4] Cokelaer F., Talbot H., Chanussot J. Efficient Robust d-Dimensional Path Operators. IEEE J. Sel. Top. Signal. Process., 2012. 6(7), 830–839. 10.1109/jstsp.2012.2213578

[5] van de Gronde J.J., Lysenko M., Roerdink J.B.T.M. Path-Based Mathematical Morphology on Tensor Fields. In I. Hotz, T. Schultz, editors, Visualization and Processing of Higher Order Descriptors for Multi-Valued Data, Math. Vis., pages 109–127. Springer International Publishing, 2015. 10.1007/978-3-319-15090-1_6

[6] van de Gronde J.J., Offringa A.R., Roerdink J.B.T.M. Efficient and robust path openings using the scale-invariant rank operator. Journal of Mathematical Imaging and Vision, 2016. Accepted

[7] van de Gronde J.J., Schubert H.R., Roerdink J.B.T.M. Fast Computation of Greyscale Path Openings. In J.A. Benediktsson, J. Chanussot, L. Najman, H. Talbot, editors, Mathematical Morphology and Its Applications to Signal and Image Processing, volume 9082 of LNCS, pages 621–632. Springer International Publishing, 2015. 10.1007/978-3-319-18720-4_52

[8] Heijmans H., Buckley M., Talbot H. Path Openings and Closings. J. Math. Imaging Vis., 2005. 22(2), 107–119. 10.1007/s10851-005-4885-3

[9] Kahn A.B. Topological Sorting of Large Networks. Commun. ACM, 1962. 5(11), 558–562. 10.1145/368996.369025

[10] Karas P., Morard V., Bartovskỳ J., Grandpierre T., Dokládalová E., Matula P., Dokládal P. Gpu implementation of linear morphological openings with arbitrary angle. Journal of Real-Time Image Processing, 2015. 10(1), 27–41

[11] Luengo Hendriks C.L. Constrained and dimensionality-independent path openings. IEEE Trans. Image Process., 2010. 19(6), 1587–1595. 10.1109/tip.2010.2044959

[12] Merveille O., Talbot H., Najman L., Passat N. Ranking Orientation Responses of Path Operators: Motivations, Choices and Algorithmics. In J.A. Benediktsson, J. Chanussot, L. Najman, H. Talbot, editors, Mathematical Morphology and Its Applications to Signal and Image Processing, volume 9082 of LNCS, pages 633–644. Springer International Publishing, 2015. 10.1007/978-3-319-18720-4_53

[13] Morard V., Decencière E., Dokládal P. Geodesic Attributes Thinnings and Thickenings. In P. Soille, M. Pesaresi, G.K. Ouzounis, editors, Mathematical Morphology and Its Applications to Image and Signal Processing, volume 6671 of LNCS, pages 200–211. Springer Berlin Heidelberg, 2011. 10.1007/978-3-642-21569-8_18

[14] 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). IEEE, 2011 pages 1457–1460. 10.1109/icassp.2011.5946767

[15] Morard V., Dokládal P., Decencière E. One-Dimensional Openings, Granulometries and Component Trees in O(1) Per Pixel. IEEE J. Sel. Top. Signal. Process., 2012. 6(7), 840–848. 10.1109/jstsp.2012.2201694

[16] Morard V., Dokládal P., Decencière E. Parsimonious Path Openings and Closings. IEEE Trans. Image Process., 2014. 23(4), 1543–1555. 10.1109/tip.2014.2303647

[17] Talbot H., Appleton B. Efficient complete and incomplete path openings and closings. Image Vis. Comput., 2007. 25(4), 416–425. 10.1016/j.imavis.2006.07.021

[18] 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., 2010. 31(10), 1120–1127. 10.1016/j.patrec.2009.12.018

[19] Yuan J., Gleason S.S., Cheriyadat A.M. Systematic benchmarking of aerial image segmentation. Geoscience and Remote Sensing Letters, IEEE, 2013. 10(6), 1527–1531