Computers & Geosciences 145 (2020) 104554

Contents lists available at ScienceDirect

Computers and Geosciences

journal homepage: www.elsevier.com/locate/cageo

Research paper

An algorithm to reduce a river network or other graph-like polygon to a set of lines

E.I. Schaefer [a,*,1], J.D. Pelletier [b]

[a] University of Arizona, Lunar and Planetary Laboratory, 1629 E. University Blvd., Tucson, AZ, 85721, USA
[b] University of Arizona, Department of Geosciences, 1040 E. Fourth St., Tucson, AZ, 85721, USA

A B S T R A C T

In many geomorphic applications, it is necessary to reduce a polygon to a set of axial lines. For example, consider a fluvial channel network. On the one hand, this network has a polygonal footprint with reaches of finite width organized into a branching network. On the other hand, measuring the length or sinuosity of these reaches, as examples, requires the reduction of their polygonal footprints to axial lines. Here we present a new algorithm that can objectively reduce a polygon to such a set of axial lines, formally called a skeleton. Across four illustrative test cases, we show that this vector-based algorithm has some advantages over three raster-based algorithms in current geomorphic use because it generates smooth, continuous, well-centered skeletons and supports a useful scale-independent metric for the removal of spurious portions. Skeletons in this algorithm are uniquely constructed using a minimum of two parameters: a sampling interval (similar to a resolution) and a numeric pruning criterion (which determines skeleton complexity). We have implemented the algorithm as a freely available, open-source, GIS-ready Python code package without commercial dependency.

1. Introduction

To derive useful metrics, geomorphologists often reduce polygonal surface expressions to linear representations. For example, the length and sinuosity of a river are more precisely the length and sinuosity of its centerline. Similarly, the orientation of a cross-valley profile is determined by the local orientation of the valley's axis. In the simplest cases, such as a canal with parallel sides, the course of a centerline or valley axis may be obvious and trivial to trace or construct. In more complicated cases, however, derivation of such linear representations by a general algorithm provides the consistency and reproducibility that can be essential for geomorphic comparisons, both within a project and between projects. For example, the quantitative description of a cross-valley profile may depend sensitively on both the position of the valley axis and its local direction (Greenwood and Humphrey, 2002; Harbor and Wheeler, 1992; Pattyn and Van Huele, 1998).

In addition, as remotely-sensed data proliferates, the need for automated analyses likewise increases. For example, Allen and Pavelsky (2018) recently used the automated tool RivWidth (Pavelsky and Smith, 2008) to compile a global survey of river morphology.

To provide consistency, reproducibility, and scalability to the derivation of linear representations—properly called skeletons—from polygons in geomorphic applications, we present a new general algorithm. This algorithm builds on earlier work in computer vision and related fields but is specifically tailored for use in geomorphic analysis. To further enhance reproducibility and ease of use, the algorithm only requires two parameters (though more are available): a sampling interval, which is similar to resolution, and a numeric value that specifies how aggressively the skeleton is simplified. Across four experiments, we compare the respective skeletons produced by this vector-based algorithm to those produced by three raster-based algorithms in current use within fluvial geomorphology.

2. Methods

The algorithm has two distinct phases, each of which generates a skeleton (Fig. A1 of Appendix). In Phase 1, the generated skeleton is organized like a graph (network), and we therefore refer to this skeleton as the "graph skeleton." In Phase 2, the graph skeleton is partitioned into a set of paths, each of which is a series of edges (links) from the graph skeleton. We therefore refer to that skeleton as the "partitioned skeleton."

* Corresponding author.
E-mail addresses: ethan.i.schaefer@gmail.com (E.I. Schaefer), jdpellet@email.arizona.edu (J.D. Pelletier).
URL: https://github.com/eischaefer/numgeo (E.I. Schaefer).
[1] Present address: University of Western Ontario, Department of Earth Sciences, 1151 Richmond Street N. London, Ontario, Canada N6A 5B7.

https://doi.org/10.1016/j.cageo.2020.104554
Received 20 March 2019; Received in revised form 26 May 2020; Accepted 11 June 2020
Available online 28 July 2020

2.1. Phase 1

2.1.1. Voronoi analysis

Phase 1 follows a classic procedure that begins with sampling the input polygon's boundary at some regular interval specified by the user (Fig. 1b) (e.g., Brandt and Algazi, 1992; Kirkpatrick, 1979; Lee, 1982). The Voronoi diagram for these boundary samples (Fig. 1c) is then derived, for which we use the qhull library (Barber et al., 1996). The Voronoi diagram describes, for each boundary sample, the polygonal neighborhood in which all enclosed coordinates are closer to that boundary sample than to any other boundary sample. Consequently, each segment that bounds these neighborhoods is a perpendicular bisector to the line segment between two neighboring boundary samples. For brevity, we refer to these neighborhood-bounding segments as "Voronoi segments" hereinafter.

2.1.2. Isolation of the graph skeleton

The next step in the classic procedure is to isolate a set of Voronoi segments that together form a reasonable skeleton for the input polygon. In the simplest case, the sampling interval is set to no greater than half the width of the narrowest part of the polygon. In that case, the Voronoi segments that are completely contained by the input polygon form a skeleton. Furthermore, this skeleton retains the topology of the input polygon, or more casually, its general connectedness (Theorem 4.1 of Brandt and Algazi (1992)). Hereinafter, wherever an input polygon has a width that is narrower than twice the sampling interval, we will describe it as locally "under-sampled."

Unfortunately, under-sampling can be unavoidable for large datasets due to memory constraints (see section A5 of Appendix). In view of that practical limitation, we isolate the graph skeleton as follows. First, we identify and discard each Voronoi segment that is the perpendicular bisector of the line segment between two consecutive boundary samples (Fig. 1d) (Karimipour et al., 2013). If the sampling interval is sufficiently fine, the remaining Voronoi segments then form one graph that represents the skeleton and one or more additional graphs that together form the skeleton's complement (Fig. 1e). The skeleton's complement is essentially the skeleton of the area excluded by the polygon. For example, if the input polygon represents a river, the skeleton's complement is the skeleton of the land near the river.

Second, we generate a proxy polygon that approximates the input polygon but for which all vertices are boundary samples (insets of Fig. 1e). Third, we identify each node where three or more Voronoi segments meet, hereinafter a "hub", and test whether that hub lies within the proxy polygon (Fig. 1e). Testing each hub, rather than each Voronoi segment, is more computationally efficient and preserves the skeleton's topology in some cases of under-sampling (e.g., the top-center (magenta) and top-right (green) insets of Fig. 1e) (cf. Ogniewicz and Ilg, 1992). The choice to test against the proxy polygon, rather than the input polygon, is heuristic. Conceptually, because the skeleton is generated from the same boundary samples that define the proxy polygon, the skeleton is more directly sensitive to the proxy polygon than to the input polygon. The proxy polygon therefore tends to better discriminate between the skeleton and its complement. For example, the skeleton "stub" (node connected to exactly one edge) in the top-right (green) inset of Fig. 1e lies within the proxy polygon but outside the input polygon.

Fourth, that graph for which the greatest number of hubs lie within the proxy polygon is identified as the "candidate graph skeleton" (Fig. 1f). By default, if any of the candidate graph skeleton's hubs lie outside the proxy polygon, the algorithm fails. Alternatively, the user can permit a heuristic resolution in which those hubs in the candidate graph skeleton that lie outside the proxy polygon are removed, along with each Voronoi segment that terminate at those hubs. This strategy, which we call "cutting", is applied in section 3.2 and its shortcomings discussed in section 4.1.6. After cutting, the fourth step—identification of the candidate graph skeleton—is repeated, followed by a second cutting, if necessary. Cutting will only be attempted up to two times, after which the algorithm fails. In testing, cutting tended to ensure that the skeleton's complement was excluded even in cases of significant under-sampling, which is essential to the validity of Phase 2.

2.2. Phase 2

At progressively finer sampling intervals, the input polygon is increasingly well sampled, resulting in a smoother and more well-centered skeleton (Fig. 2). Indeed, Schmitt (1989) proved that in the limit that the spacing between boundary samples goes to zero, the derived skeleton converges on the (rigorously-defined) medial axis everywhere except at the polygon boundary itself, which the skeleton never reaches (his Theorem 15). However, finer sampling intervals also result in a greater sensitivity to the vertices of the input polygon, causing marginal edges (i.e., those with a stub at one end) to proliferate in the graph skeleton (e.g., bottom-left (orange) insets in Fig. 2) (Saha et al., 2016).

In typical use-cases, some of these marginal edges are undesirable and should be pruned. We adopt a pruning criterion (section 2.2.2) that requires the graph skeleton to be first partitioned into paths, that is, sequences of edges that touch end-to-end. Phase 2 therefore begins with the iterative identification of each of these paths, the removal of the path's constituent edges from the graph skeleton, and deposition of the path into a new skeleton that we call the partitioned skeleton. For brevity, we refer to this sequence as "partitioning out" a path.

2.2.1. Partitioning out

To begin, the graph skeleton is solved in a graph traversal sense using Dijkstra's algorithm, that is, the shortest path between each pair of nodes is computed. By default, the longest of these paths that has each end at a stub is designated the "trunk" and partitioned out (Fig. 3b), but this behavior can be customized. The trunk is then extended at each stub end so that it touches the boundary of the input polygon at a boundary sample (Fig. 3c). We call these extensions "tails." Due to the Voronoi analysis that generated the stub, in general, that node lies exactly the same distance from the three closest boundary samples. The central one of these boundary samples is selected as the tail's end.

Because the removal of the trunk from the graph skeleton changes that skeleton's connectedness, the remaining graph skeleton is solved again and each path between any two hubs is designated a "bridge" and partitioned out (Fig. 3d). The remaining graph skeleton is then solved again, and the longest length-optimized path between a hub and a stub is designated a "branch" and partitioned out (Fig. 3e). A tail is added to the branch's stub end. The bridge- and branch-finding steps are then repeated until the graph skeleton is exhausted (Fig. 3f; also Fig. 2a–c).

2.2.2. Pruning

The algorithm's pruning criterion is designed to be both effective and computationally efficient. This criterion depends on a quantity that we call the "normalized length," defined as the length of a branch, including its tail, divided by half the approximate width of the input polygon at that branch's stem (i.e., hub end) (Fig. 2d–f). We calculate a branch's length directly from its coordinates. We estimate a branch's width at its stem as twice the distance between the stem node and the nearest boundary sample (center-left (red) insets of Fig. 2d–f), whose identity is known from the earlier Voronoi analysis. We find that the normalized length reasonably estimates the relative importance of each branch in a largely scale-invariant manner (section 4.2.2). Quantitatively, the least important branches—which are commonly the most abundant—have values near unity, because they approximately span the local half-width, and more important branches have higher values (Fig. 2d–f). In general, the precision with which the normalized length estimates branch importance increases with fining sampling interval (e.g., compare the center-left (red) insets of Fig. 2d–f).

For a river polygon, the normalized length may be approximately

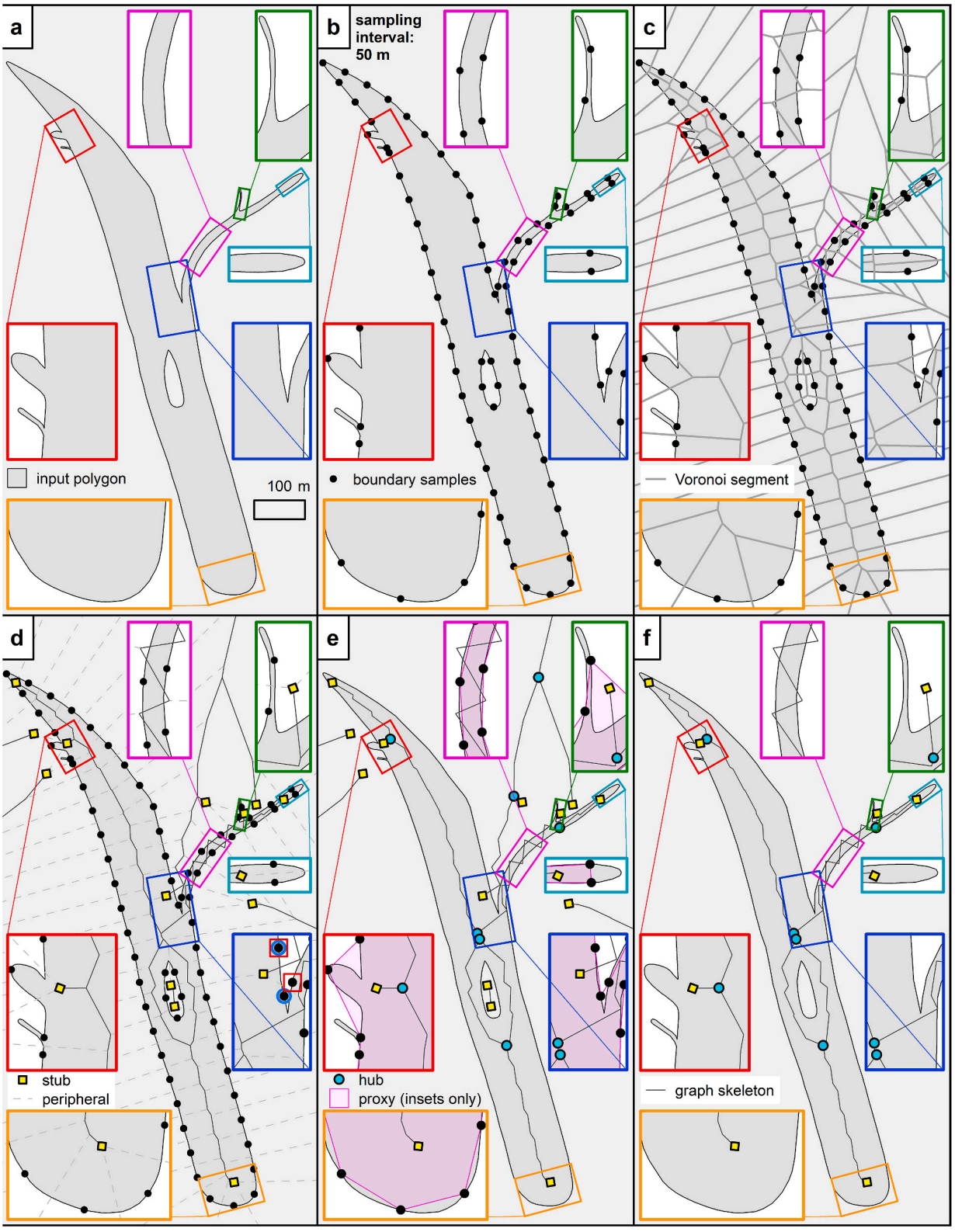


Fig. 1. Phase 1 of algorithm. (a) An example input polygon. (b) Boundary sampling at a regular interval of 50 m. (c) Voronoi diagram of boundary samples. Each segment-bounded neighborhood encloses those coordinates that are closer to single boundary sample within that neighborhood than they are to any other boundary sample. (d) "Peripheral" Voronoi segments exactly halfway between consecutive boundary samples are identified for removal. After their deletion, graph skeleton and its complement will remain. Note that portions of graph skeleton (top-center (magenta) and top-right (green) insets) exit input polygon where under-sampled (section 2.1.2), and a portion of complement (namely, perpendicular bisector of two boundary samples with red squares in bottom-right (blue) inset) enters input polygon (between two boundary samples with blue rings in same inset). Stubs (end nodes) are also labeled. (e) Graph skeleton and its complement, which is also a graph. In insets, proxy polygon, whose vertices are boundary samples, is also shown. Note that no hub (intersection) of graph skeleton lies outside proxy polygon, and no hub of complement lies inside proxy polygon. Based on this geometry, skeleton is isolated from its complement. (f) Isolated graph skeleton. (For interpretation of the references to color in this figure legend, the reader is referred to the Web version of this article.)

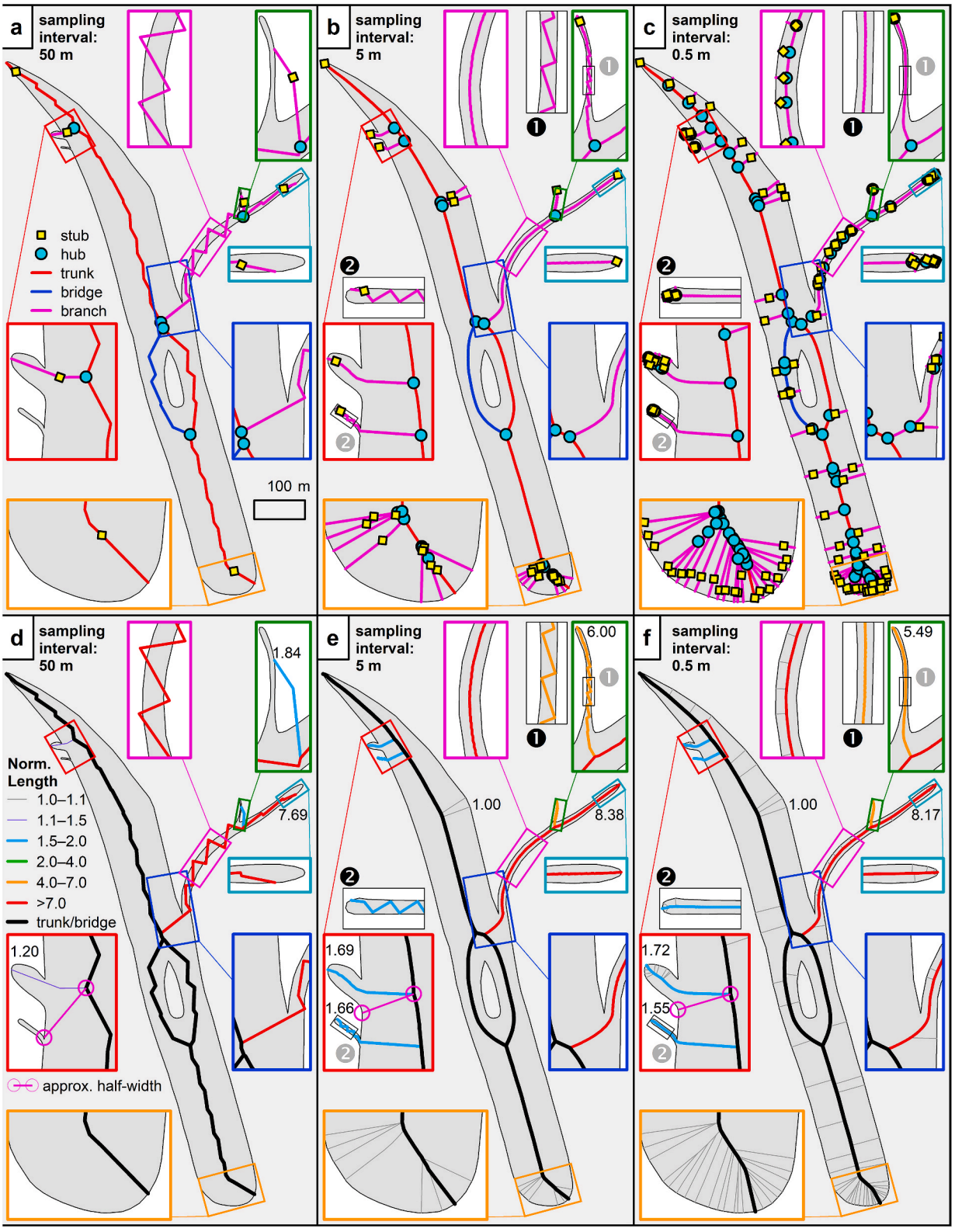


Fig. 2. Partitioned skeleton at (a) 50-m sampling, (b) 5-m sampling, and (c) 0.5-m sampling, and branch normalized lengths (with labeled example values) at (d) 50-m sampling, (e) 5-m sampling, and (f) 0.5-m sampling. Note how smoothness and centering of skeleton improves with fining sampling interval but marginal edges (i.e., those with a stub near one end) proliferate. Note also that normalized length (section 2.2.2) approximately scores importance of each branch and does so more accurately than length. For example, in (f), upper branch in center-left (red) inset is slightly shorter (57.0 m) than lower branch (57.8 m), and both are slightly shorter than branch in main figure labeled 1.00 (58.0 m). On other hand, branch in top-right (green) inset is shorter than all these branches (53.8 m) but has highest normalized length. Finer sampling intervals tend to improve normalized length precision, as suggested by comparing values in respective center-left (red) insets of (d)–(f). (For interpretation of the references to color in this figure legend, the reader is referred to the Web version of this article.)

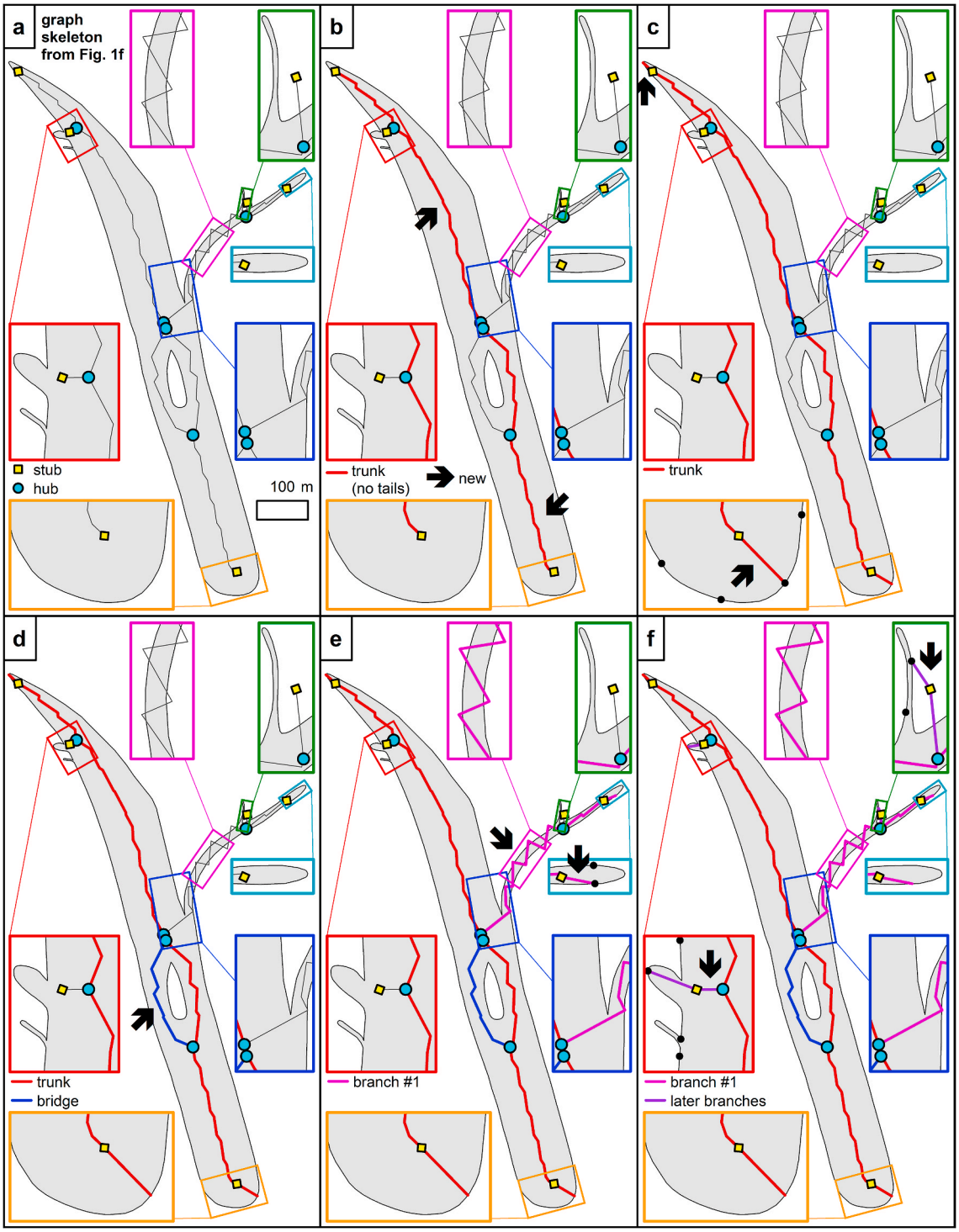


Fig. 3. Phase 2 of algorithm. (a) Graph skeleton from end of Phase 1 (Fig. 1f). (b) By default, longest length-optimized path between two stubs is designated trunk. (c) Trunk is extended at each end to a polygon boundary sample by adding a segment called a tail (e.g., bottom-left (orange) inset). (d) All paths between hub pairs are identified as bridges. For this skeleton, there is only one bridge. (e) Longest length-optimized path between a hub and a stub is identified as first branch and a tail added through its stub end (center-right (teal) inset). After this step, any new bridges would be identified, but there are none for this skeleton. (f) Subsequent branches (center-left (red) and top-right (green) insets) are iteratively identified in same way as first. For each inset in which a tail is added, local boundary samples are shown as black dots. (For interpretation of the references to color in this figure legend, the reader is referred to the Web version of this article.)

conceptualized as the ratio between a stream's length and the half-width of the more major stream that truncates it at its confluence. However, this intuition is not exact. Paths are not constrained to follow stream direction, and therefore a single branch may have both upstream- and downstream-oriented intervals. Additionally, even though branches that are partitioned out later (and therefore are truncated by branches that were partitioned out earlier) tend to be shorter, and shorter streams tend to have a lower Horton stream order (Horton, 1945), neither of these trends is monotonic or universal.

The user can specify any value as the minimum normalized length to be retained. Any branch with a smaller normalized length is pruned, and any branches or bridges that depend on that branch are also pruned, to

ensure that the final pruned skeleton is fully connected.

2.3. Templating

For large datasets analyzed at a fine sampling interval, the memory footprint of our implementation of the present algorithm can be challenging (discussed in section A5 of Appendix). To benefit from the small memory footprint of a coarse sampling interval but partially recover the advantages of a finer sampling interval, we implement a heuristic that we call "templating" (Fig. 4). This heuristic begins with derivation of a graph skeleton in the standard manner described above at some coarse sampling interval, except that tails are added to this graph skeleton (rather than to the partitioned skeleton, which is never constructed). We call this result the "template" skeleton. A second graph skeleton is then generated at some finer sampling interval. Each tailed end from the template skeleton is then matched to the nearest stub from this "fine" skeleton. Each edge in the fine skeleton that terminates at an unmatched stub is then deleted along with that stub. After these deletions, a final "hybrid" graph skeleton remains. Ideally, this hybrid skeleton retains the topological completeness of the template skeleton but adds the greater positional precision and smoothness of the fine skeleton. In practice, we have found this goal to be reasonably well approximated (see sections 3.2 and 4.1.6).

2.4. Raster-based algorithms

Alternative to vector-based skeletonization, as used in the present algorithm, are raster-based approaches (Lam and Lee, 1992; Saha et al., 2016).

In iterative digital erosion (also called thinning), the boundary of the polygonal input region is iteratively eroded inward, pixel by pixel. Where opposite erosion fronts meet, a pixel-wide skeleton is retained. The algorithm by Zhang and Suen (1984) is a classic example of iterative digital erosion and is widely used, including in recent fluvial geomorphic studies (Clubb et al., 2017; Grieve et al., 2018; Monegaglia et al., 2018).

Another approach calculates or approximates the distance from each pixel within the polygonal input region to the nearest pixel outside that region, that is, in the exterior. Based on this distance transform, a skeleton may be derived by identifying local distance-to-exterior maxima or low values in some spatial derivative of the distance-to-exterior values. The RivWidth package uses this latter (spatial-derivative) approach and was expressly developed for analyzing fluvial geometry (Pavelsky and Smith, 2008). It has recently been updated as RivWidthCloud and adapted to use Google Earth Engine, with a focus on processing Landsat images directly (Yang et al., 2019). RivWidthCloud also offers optional branch removal, which trims back a specified number of pixels from skeleton termini, but halts at junctures. This branch removal is intended to address the problem of spurious edges, which we encounter in our experiments (section 3) and discuss along with other results (section 4.1.5).

Machine learning offers yet another raster-based approach. In a recent study, Isikdogan et al. (2018) used deep learning to train a model called DeepRiver. DeepRiver assigns to each pixel a pseudoprobability that that pixel lies on the centerline. To derive a pixel-wide centerline from these pseudoprobabilities, a threshold must be chosen and the Zhang and Suen (1984) algorithm applied to thin those (typically narrow) regions with pseudoprobabilities exceeding that threshold.

2.5. Algorithms used in the present study

Together, the Zhang and Suen, RivWidthCloud, and DeepRiver algorithms represent three distinct raster-based approaches to skeletonization, and each has seen recent use in geomorphic analysis (section 2.4). We therefore compare the skeletons derived by the present algorithm to those derived by each of these algorithms across four

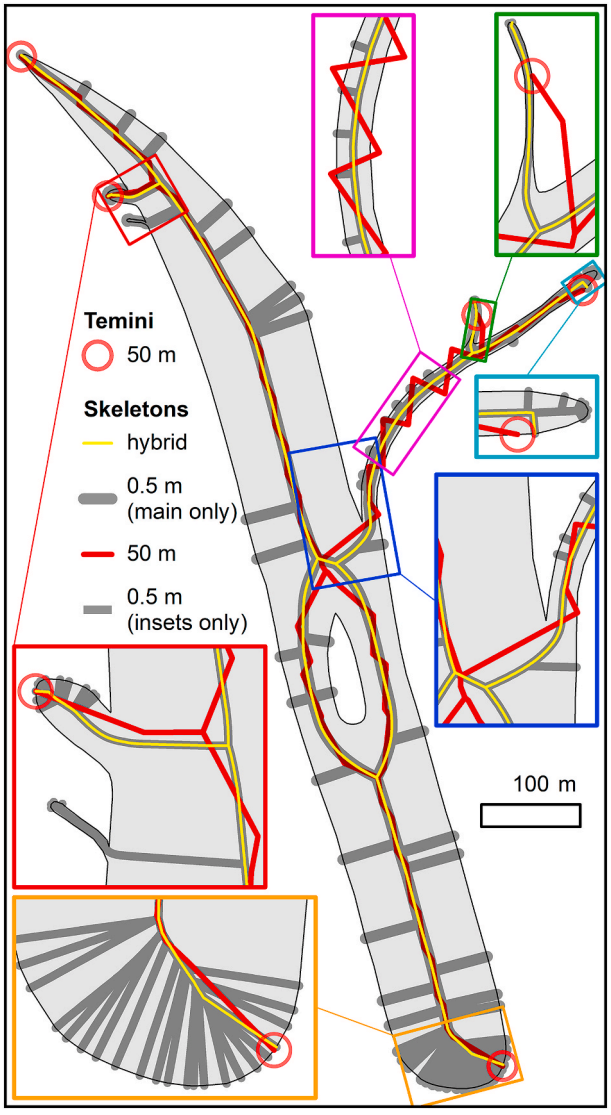


Fig. 4. Templating example (section 2.3). Figure depicts a hybrid skeleton (yellow) that was derived from a graph skeleton (dark gray; sampled at 0.5 m; cf. Fig. 2f) by filtering its stubs to approximate tailed termini of a template skeleton (red; sampled at 50 m; cf. Fig. 2c). Consequently, each terminus (red circle) of template skeleton corresponds to a single terminus in hybrid skeleton. For example, hybrid skeleton lacks a terminus for lower protrusion in center-left (red) inset because template skeleton also lacks a terminus for that protrusion. Most hybrid skeleton termini are close spatial matches to termini of template skeleton, including terminus for polygon's rightmost protrusion (center-right (teal) inset), which replicates truncated behavior of template skeleton. This close correspondence arises because graph skeleton from which hybrid skeleton is sourced typically provides a dense distribution of stubs for each protrusion (Fig. 2c) so that a close match to a template skeleton terminus is possible. However, in case of protrusion featured in top-right (green) inset, source graph skeleton has only one stub, which is therefore matched to corresponding template skeleton terminus, even though that terminus is somewhat distant. (For interpretation of the references to color in this figure legend, the reader is referred to the Web version of this article.)

experiments. For the Zhang and Suen algorithm, we specifically use the implementation in the scikit-image Python code package (van der Walt et al., 2014). For RivWidthCloud, we generate and analyze two skeletons in each experiment: one without branch removal and one with the default 500-pixel branch removal. Although the choice to adopt the default 500-pixel branch removal is arbitrary, we believe that this length is suitable for illustrative purposes; of course, other lengths may be

preferable depending on the specific application. For DeepRiver, we adopt a pseudoprobability threshold of 0.4 and process each input region by use of a sliding window.

We also generate two skeletons with the present algorithm in each experiment, one without a minimum normalized length and one with a minimum normalized length of two (see section 2.2.2). Except where specifically noted, only the latter (pruned) skeleton is used in quantitative analyses.

Section A1 of Appendix provides additional details on how each algorithm was applied in our study.

3. Results

For each experiment described in this section, and for Figs. 1–4, the principal data (algorithm inputs and outputs, and some additional data) are available online (Schaefer, 2019).

3.1. Fractal binary trees

In this section, we skeletonize polygonal regions that are created by buffering fractal binary trees (Mandelbrot, 1982; Mandelbrot and Frame, 1999). The details of how we construct and buffer these trees and prepare each input region for analysis are provided in section A2 of Appendix, and section A3 of Appendix describes how we calculated metrics to compare each skeleton to its source tree.

3.1.1. Constant buffer experiment

We begin with the simplest case and apply a constant buffer of ~0.108 m to the tree, producing a footprint that is typically nine pixels wide after rasterization (Fig. 5). For each algorithm, the skeleton generally lies near the trace of the tree except at intersections, where the tree's sharp 60° bifurcation is replaced by a wider opening that is offset distally (i.e., toward the next branch generation) relative to the tree's bifurcation (Fig. 5c and f). The skeleton generated by the present algorithm also departs from the (unbuffered) tree wherever that skeleton extends to the boundary of the buffered tree; this typically occurs near tree termini where that skeleton splits into prongs, each of which ends near a vertex of the input polygon (Fig. 5b and f).

Of the five skeletons, the skeleton generated by the present algorithm tends to be the smoothest at scales similar to 0.025 m and is generally the closest to the tree, whereas the DeepRiver skeleton tends to be the furthest from the tree (Fig. 5d and Table A1). The DeepRiver skeleton is also the only one with discontinuities (Fig. 5c–e), of which there are several hundred. These discontinuities form gaps as large as 16.45 m and leave 23 branches as low as the third generation eliminated or shortened by $\geq$ 75% (Fig. 5e). E–W elements seem to be disproportionately affected, with 9.8% (22 of 512) so shortened. Each of the other five orientations has $\geq$112 elements but only one element not oriented E–W lacks so much of its length. For the RivWidthCloud skeleton, 500-pixel branch removal eliminates or shortens by $\geq$ 75% most eighth- and ninth-generation branches (Fig. 5e) as well as truncates the trunk (Fig. 5b). (The RivWidthCloud skeletons without branch removal have no such eliminations or truncations in any of the three experiments with fractal binary trees.) The total respective lengths of the DeepRiver and RivWidthCloud-with-branch-removal skeletons are 84.5% and 74.6% that of the tree. The remaining skeletons all have lengths similar (98.7–102.2%) to that of the tree (Table A1).

In addition to terminal prongs, the skeleton generated by the present algorithm also includes several stray edges that are well removed from tree termini. Such features, whether near a tree terminus or along a tree mid-section, are common in skeletonization and are conventionally called "spurious branches" (Saha et al., 2016). However, as we have already used "branch" in two distinct senses (for binary trees and Phase 2 of the present algorithm), we instead use "spurious edges" hereinafter. Note that, in each set of prongs near a tree terminus, one prong forms part of a Phase-2 branch whereas each other prong is its own Phase-2 branch (Fig. 5b and f). If we treat only the isolated prongs as spurious edges, all spurious edges in this skeleton have a normalized length <1.34, and no other (Phase-2) branch has a normalized length <2.55. Moreover, among these legitimate branches, only the 16 ninth-generation (binary-tree) branches that merge at their respective stems have a normalized length <3.61. Spurious edges are absent in the other skeletons.

3.1.2. Variable buffer experiment

In our second experiment, we apply a variable buffer equal to 10% of the length of each element in the tree (Fig. 6). Such buffering crudely mimics the general downstream trend of increasing stream width.

In this experiment, the DeepRiver skeleton is completely missing the trunk, both first-generation branches, and the two non-vertical second-generation branches. A ridge of locally high pseudoprobability lies well off-center in the trunk and first-generation branches, ~3 m from the left bank, then swings toward the tree axis at the base of the second-generation branches (Fig. 6b and c). In total, 35 tree elements lack $\geq$75% of their length in the DeepRiver skeleton. Of those elements, 30 are ninth-generation branches, and 31 are parallel to N 30° E (Fig. 6e). For the RivWidthCloud skeleton, branch removal again eliminates or shortens by $\geq$ 75% eighth- and ninth-generation branches (Fig. 6e and f), but a much smaller proportion of eighth-generation branches are affected than in the first experiment (Table A1). The total respective lengths of the DeepRiver and RivWidthCloud-with-branch-removal skeletons are 68.4% and 82.1% of that of the tree in this experiment.

In the skeleton generated by the present algorithm, spurious edges are organized in ray-like forms at the base of the trunk (Fig. 6b) and near most intersections (Fig. 6c and d), and are approximately symmetric across the tree. All spurious edges have a normalized length <1.39, no other (Phase-2) branch has a normalized length <3.67, and except for the 16 ninth-generation (binary-tree) branches that merge at their respective stems, no legitimate (Phase-2) branch has a normalized length <5.64. Spurious edges are absent in the other skeletons, but a single isolated 10-cm-long segment appears in the RivWidthSkeleton without branch removal (Fig. 6b inset).

The general behavior of each skeleton is otherwise similar to that described for the first experiment.

3.1.3. Variable buffer with noise experiment

In our third experiment, we explore the effect of noise. We introduce noise by first inserting eight vertices, evenly spaced, along the length of each element in the tree and then offset the x- and y-coordinates of each of these vertices independently by a random value in the range [-0.07L, 0.07L), where L is the element's original length. Noise is not added to the vertex at either end of each element, to retain continuity. A variable buffer is then applied to this noisy tree as in the second experiment (Fig. 7).

The kinks in the noisy tree are rounded in each skeleton (see especially Fig. 7f), contributing to a total length deficit of ~7–9% for those skeletons that had very similar lengths to the tree in the first two experiments, namely, those generated by the Zhang and Suen, RivWidthCloud (without branch removal), and present algorithms (Table A1).

Of the 32 elements lacking $\geq$75% of their length in the DeepRiver skeleton, 11 are oriented N–S and 16 are oriented parallel to N 30° E (Fig. 7d). For the RivWidthCloud skeleton, branch removal again eliminates or shortens by $\geq$ 75% eighth- and ninth-generation branches (Fig. 7f), but a much smaller proportion of eighth-generation branches are affected than in the second experiment (Table A1). Unlike in previous experiments, four elements of lower-generation branches are also eliminated or shortened by $\geq$ 75% by branch removal due to truncation back from small discontinuities in the RivWidthCloud skeleton (Fig. 7e). The total respective lengths of the DeepRiver and RivWidthCloud-with-branch-removal skeletons are 61.4% and 78.8% of that of the tree in this experiment (Table A1).

In the skeleton generated by the present algorithm, spurious edges

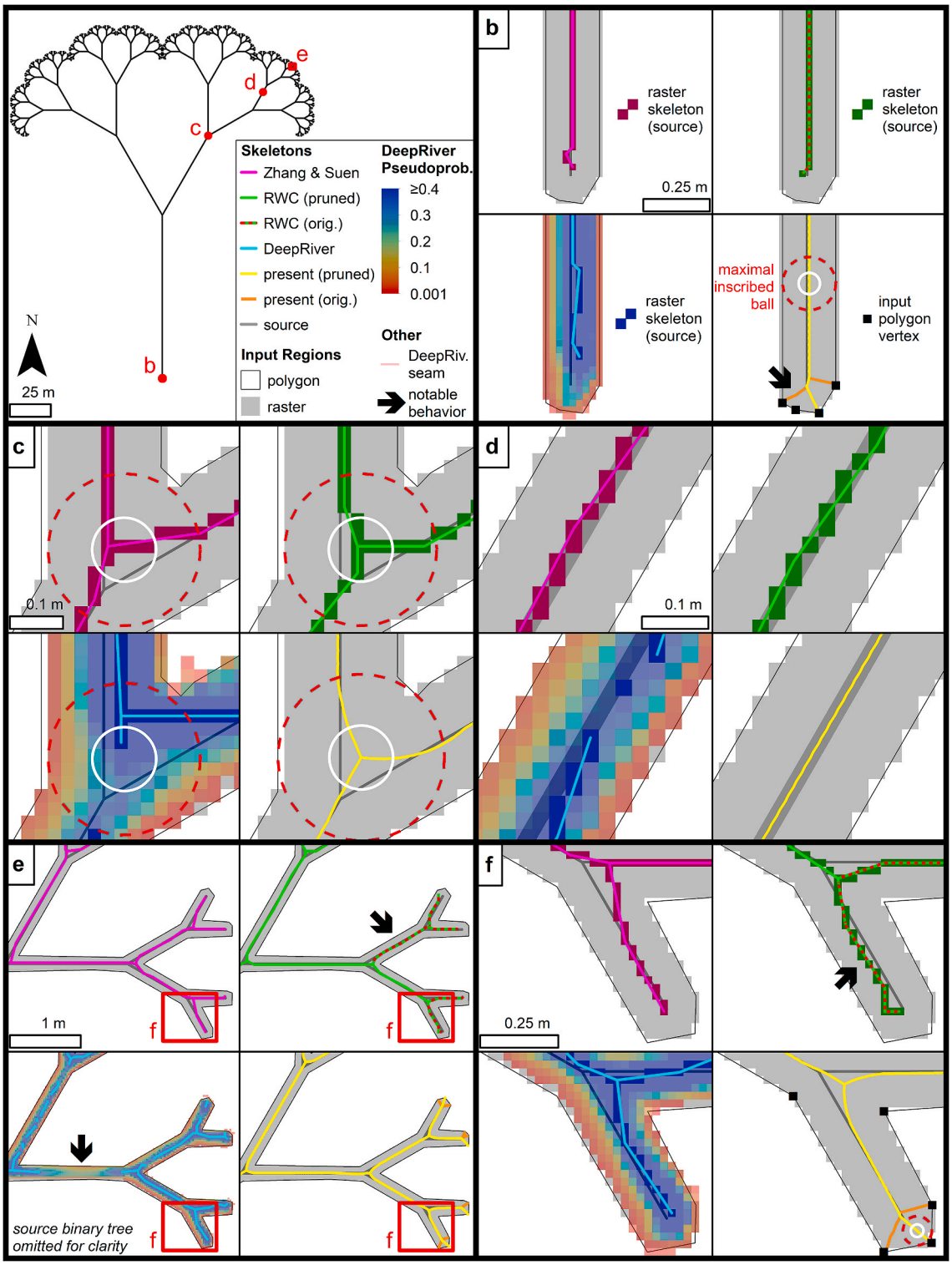


Fig. 5. First experiment: fractal binary tree with constant buffer. (a) Overview. (b) Base of trunk. Respective skeletons generated by Zhang and Suen, RivWidthCloud, DeepRiver, and present algorithms are shown, as is binary tree source (gray line) and DeepRiver pseudoprobabilities (color ramp, bottom-left pane only). Maximal inscribed ball (section 4.1.1) is drawn in bottom-right pane as red dashed circle with white solid inner circle to help pinpoint ball's center. Arrow in that pane points to a terminal prong. (c) An intersection between first- and second-generation branches. Maximal inscribed ball in each pane extends to outer edge of raster (three panes) or polygon (bottom-right pane) input region, as appropriate. (d) Representative midsection of third-generation branch. Note relative alignment of each skeleton with binary tree (also Table A1), which here is symbolized with a thickness of 0.025 m to correspond to one pixel's width for raster input region and one sampling interval for input polygon. (e) Seventh- through ninth-generation branches. Arrow in top-right pane points to eighth-generation branch eliminated by RivWidthCloud's branch removal. Our analysis (section A3 of Appendix) indicates that this tree element is short by 84%. Arrow in bottom-left pane points to notable gap in DeepRiver skeleton accompanied by low pseudoprobabilities. Our analysis indicates that this tree element is short by 63% and that intermittently expressed tree element in top-left of pane is short by 34%. (f) Ninth-generation branch. Arrow in top-right pane points to branch eliminated by RivWidthCloud's branch removal. Our analysis indicates that this tree element is short by 84%. Seamlines do not appear in bottom-left panes because none pass through depicted extents. (For interpretation of the references to color in this figure legend, the reader is referred to the Web version of this article.)

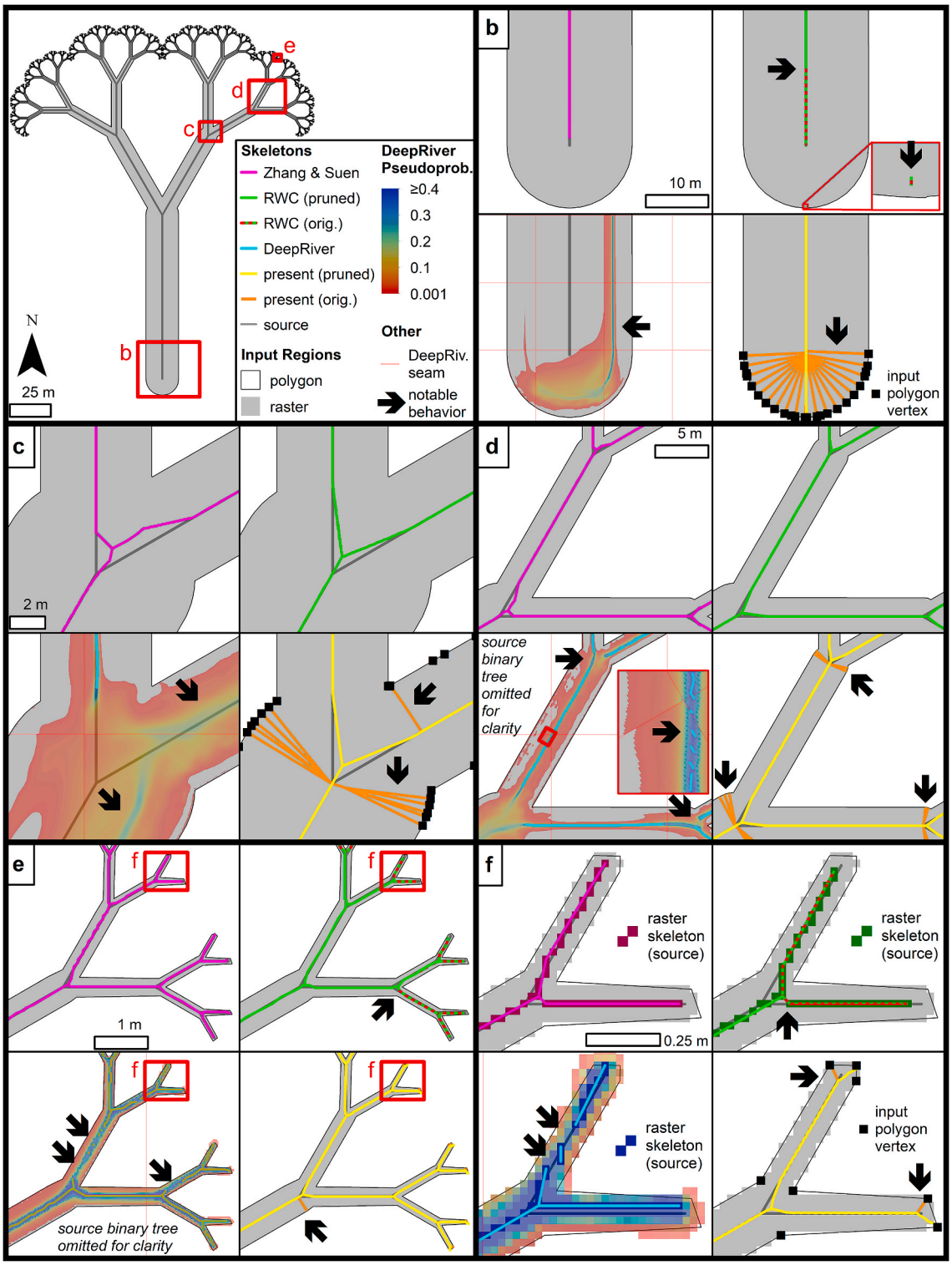


Fig. 6. Second experiment: fractal binary tree with variable buffer. In each lettered figure, arrows point respectively to truncation by RivWidthCloud's branch removal in top-right panes, isolated segment in inset of (b), high-pseudoprobability ridge in bottom-left panes of (b) and (c), skeleton discontinuities in bottom-left panes of (d)–(f) and inset of (d), and spurious edges in bottom-right panes. For reference, tree element with two arrows in lower-left pane of (e) is short by 50% in our analysis (section A3 of Appendix) (See additional descriptions in caption to Fig. 5.).

are again organized in ray-like forms near most intersections (Fig. 7c–e) and at the base of the trunk (Fig. 7b). However, unlike in the second experiment, these edges locally occur on only one side of the tree at intersections (cf. Fig. 6c and d). In this skeleton, all spurious edges have a normalized length <1.39, no other (Phase-2) branch has a normalized length <2.51, and except for 12 ninth-generation (binary-tree) branches that merge at their respective stems, no legitimate (Phase-2) branch has a normalized length <4.59 (Fig. 8).

Unlike previous experiments, the Zhang and Suen, RivWidthCloud, and DeepRiver skeletons also exhibit rare spurious edges (arrows in Fig. 7), though 500-pixel branch removal eliminates them from the RivWidthCloud skeleton.

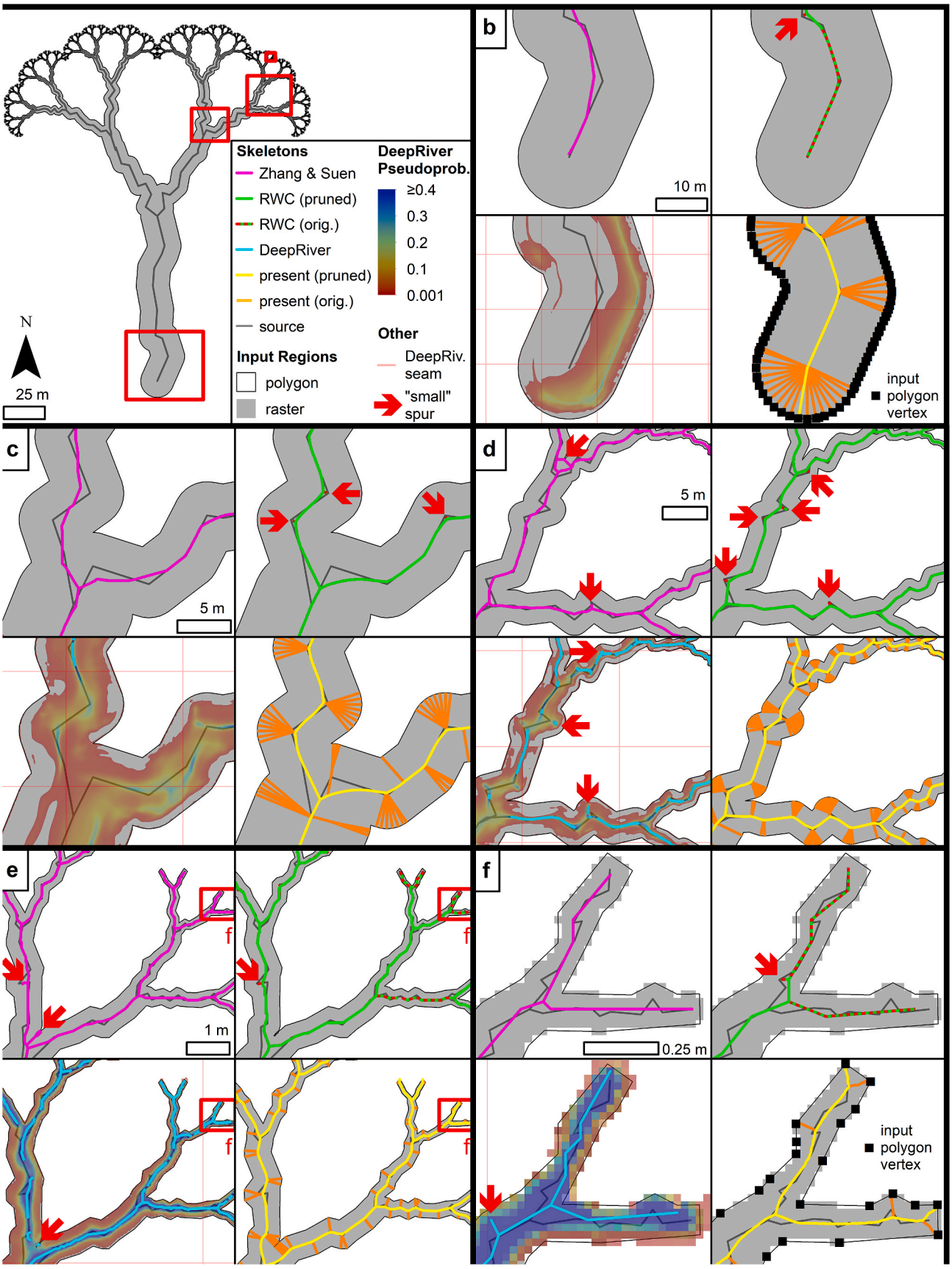


Fig. 7. Third experiment: fractal binary tree with variable buffer and noise. Arrows point to spurious edges that are short relative to figure extent except in lower-right panes, where they are too abundant to highlight individually. For reference, tree element in left half of bottom-left pane of (d) is short by 75% in our analysis (section A3 of Appendix) (See additional descriptions in caption to Fig. 5.).

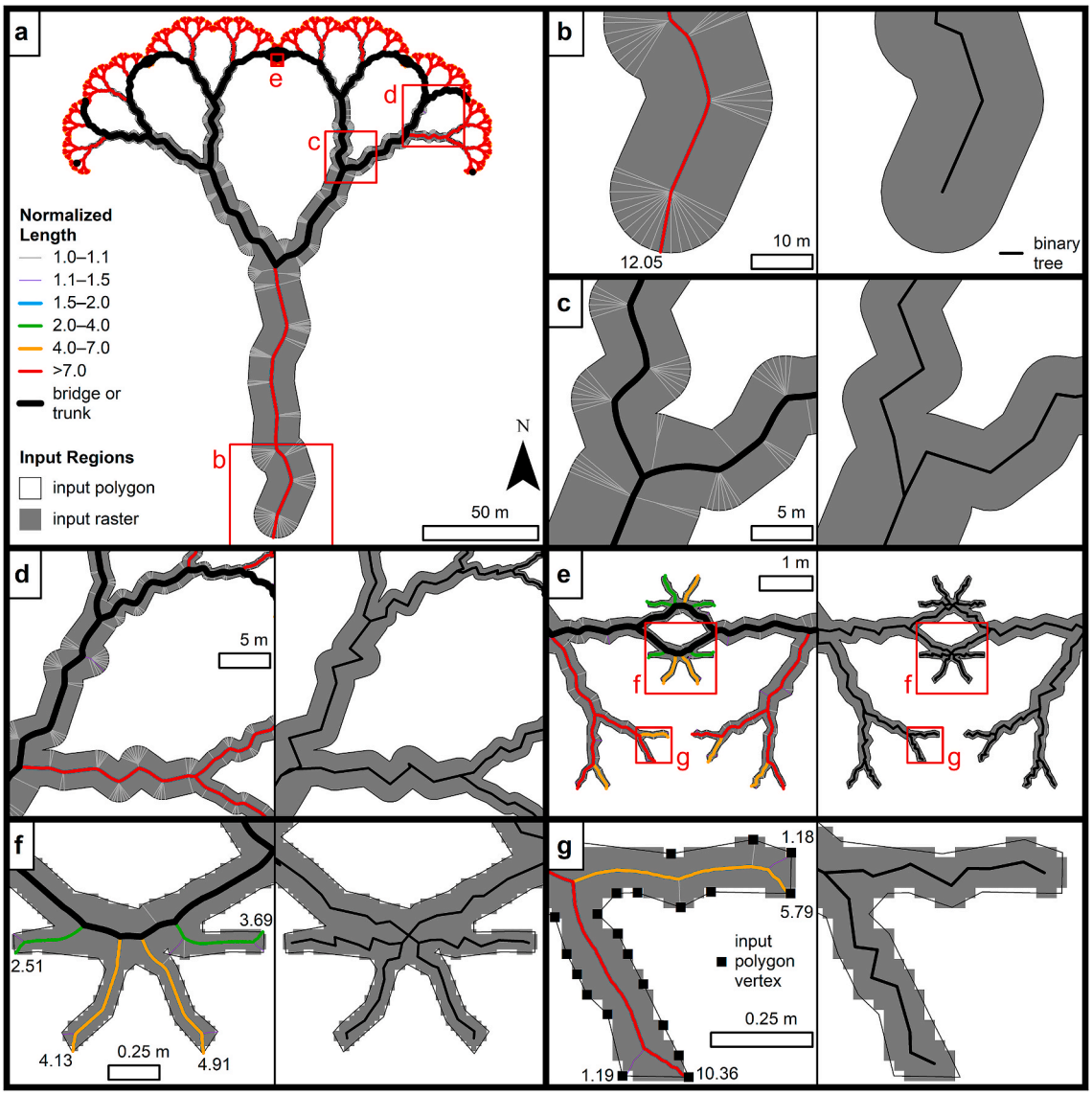


Fig. 8. Another view of third experiment (cf. Fig. 7) showing only results from present algorithm ((a) and left panes of (b)–(g)) and source binary tree (right panes of (b)–(g)), for comparison. In this skeleton, there is a gap in normalized lengths between 1.39 and 2.51 that separates all spurious edges, which have values below that gap, from all legitimate edges. Furthermore, only 12 of 495 legitimate branches have a normalized length in range 2.51–4.59, and all of these are ninth-generation branches that intersect at their respective bases ((e) and (f)). Numeric labels indicate normalized lengths of seven legitimate branches ((b), (f), and (g)) and two spurious branches ((g) only). Normalized lengths of these two spurious branches (1.18 and 1.19) are in top 4% of all spurious branch normalized lengths in this experiment.

The general behavior of each skeleton is otherwise similar to that described for the second experiment.

3.2. Yukon River reach

In our fourth experiment, we examine a highly braided reach of the Yukon River through the Yukon Flats (Fig. 9). The derivation of the region used in analysis and our experimental procedure are described in section A4 of Appendix. We highlight here that we use both templating (section 2.3)—with a coarse scale of 15 m and a fine scale of 0.25 m—and cutting (section 2.1.2) to generate the present algorithm's skeleton. The input raster for the remaining algorithms has a pixel scale of 1 m.

Even at coarse scales, there are conspicuous differences between the skeletons derived by each algorithm. The Zhang and Suen skeleton (Fig. 9b) and the skeleton generated by the present algorithm (Fig. 9e) both have abundant spurious edges, and the spurious edges of the latter commonly have normalized lengths <2. Unlike the third experiment, the spurious edges of that skeleton are also distributed along region intervals, similar to those of the Zhang and Suen skeleton, rather than forming localized ray-like bundles. The Zhang and Suen skeleton is more jagged than the other skeletons (arrows in Fig. 9b). In the RivWidthCloud skeleton, there are abundant discontinuities that truncate or isolate portions of that skeleton (arrows in Fig. 9c). Many of these portions are then eliminated by 500-pixel branch removal (arrows in Fig. 10b), which nearly halves the total length of that skeleton (Table A1). Where the channel is ~200-m-wide, the DeepRiver pseudoprobability peaks near the channel's center but varies between coherent and diffuse (two lower arrows in Fig. 9d). The associated skeleton is consequently discontinuous in those reaches. Where the channel is significantly wider than 200 m, a diffuse ridge of locally high pseudoprobability below the 0.4 threshold often runs ~125 m off either or both banks, and the skeleton is absent (two upper arrows in Fig. 9d).

At finer scales, it is clear that edges stemming from the local main

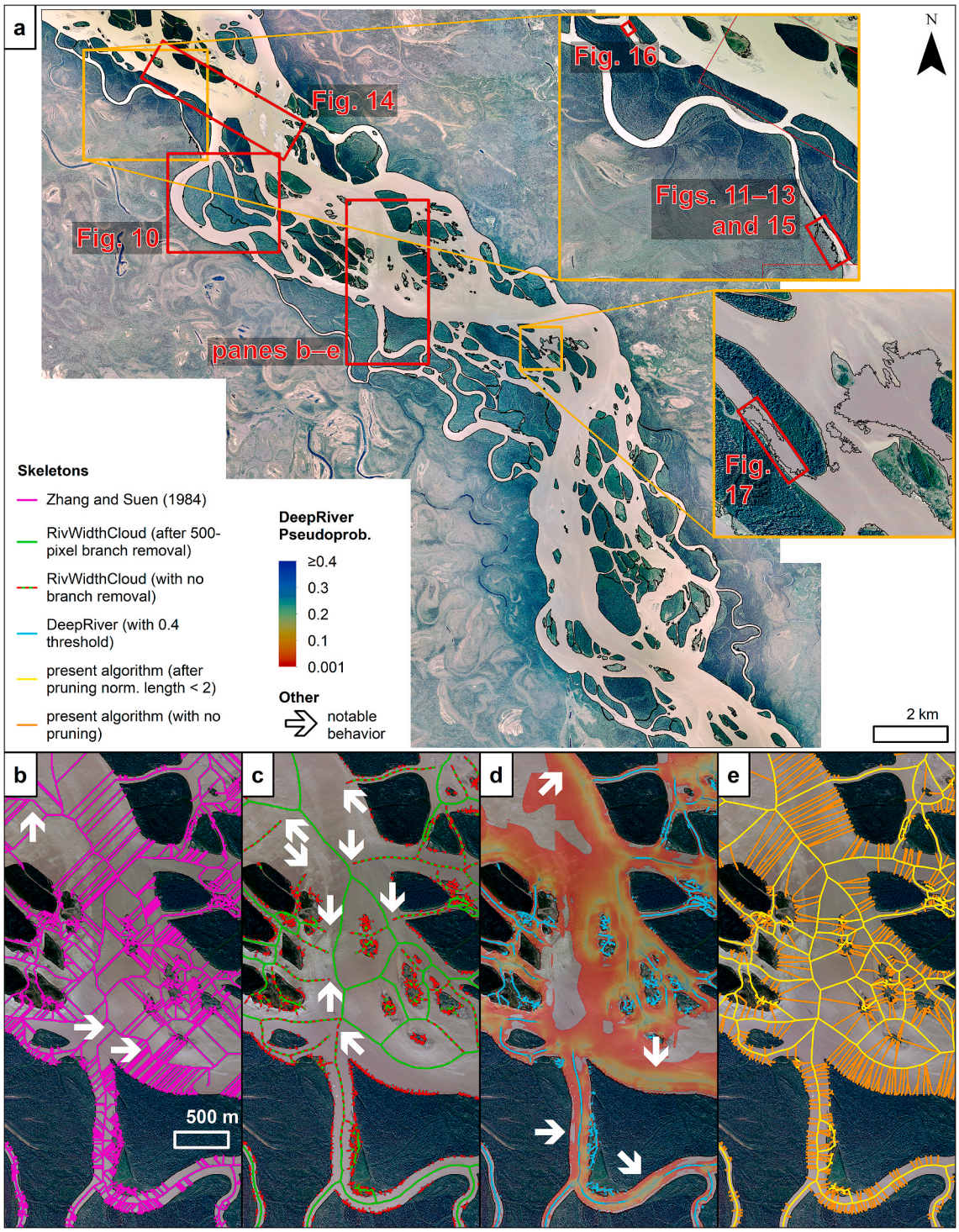
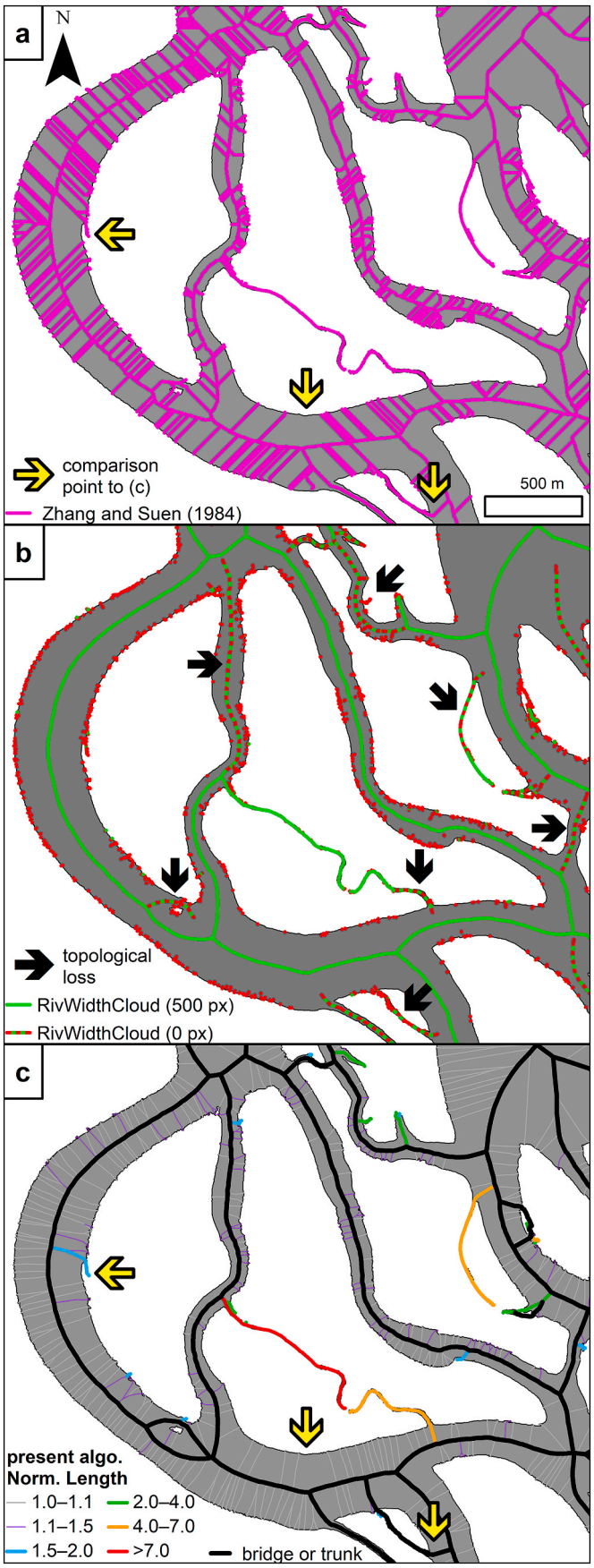


Fig. 9. Fourth experiment: highly braided reach of Yukon River. (a) Input polygon and orthomosaic from which it was derived. Top-left corner is ~11 km southeast (upstream) of Fort Yukon, Alaska. Note that artifacts are intentionally retained as challenging tests for algorithms. For example, dark water in center-right inset is excluded as though it were land. (b) Zhang and Suen skeleton, with some jagged portions highlighted by arrows. (c) RivWidthCloud skeleton, with some discontinuities highlighted by arrows. (d) DeepRiver results. In widest intervals, high-pseudoprobability ridges are diffuse and run near one bank (two upper arrows) rather than near central axis. In intervals ~200-m wide, high-pseudoprobability ridges are centered but vary between diffuse and coherent, resulting in a discontinuous skeleton (two lower arrows). High-pseudoprobability ridges are centered and coherent in narrower intervals. (e) Skeleton generated by present algorithm. Note that spurious edges usually have normalized lengths <2. All imagery is from the United States Geological Survey.

course of the Zhang and Suen skeleton are usually oriented NE–SW or NW–SE (Figs. 10a and 11). This strong orientation preference results in kinks and, especially for spurious edges, abrupt changes in orientation at the apex of smoothly curving braids (arrows in Fig. 10a). In contrast, the orientation of the corresponding edges in the skeleton generated by the present algorithm varies continuously, such that these edges are approximately perpendicular to the bank (arrows in Fig. 10c).

These finer scales also reveal the RivWidthCloud and DeepRiver skeletons to have frequent spurious edges and isolated components, typically a few meters to a few tens of meters in size (Figs. 12 and 13).



(caption on next column)

Fig. 10. Representative subset of select skeletons generated in fourth experiment. (a) For the Zhang and Suen skeleton, edges stemming from local main course are usually oriented NE–SW or NW–SE, resulting in abrupt orientation changes along curved braids (left and center arrows) and kinks (right arrow). (b) For RivWidthCloud skeleton, 500-pixel branch removal eliminates topologically important intervals (arrows) that are isolated by discontinuities prior to branch removal. (c) Skeleton generated by present algorithm. Compare to (a) and (b).

These artifacts are more abundant in the RivWidthCloud skeleton but are mostly eliminated by the 500-pixel branch removal. Very rarely, the RivWidthCloud skeleton also has 2-pixel-wide knots (inset of Fig. 12b). Aside from these artifacts, the RivWidthCloud skeleton generally lies very close to the skeleton generated by the present algorithm (Fig. 14). In fact, 98.0% of the vectorized RivWidthCloud skeleton with branch removal lies within 1 m of the skeleton with a minimum normalized length of two generated by the present algorithm, and 84.5% of the former skeleton lies within one half-meter of the latter skeleton.

Boundary irregularities with dimensions smaller than ~15 m are frequently associated with segments in the Zhang and Suen skeleton and in the RivWidthCloud skeleton without branch removal, and some irregularities in that size range have expression in the DeepRiver skeleton as well (Figs. 11–13). However, these irregularities are generally unrepresented in the skeleton generated by the present algorithm (Figs. 14c and 15). In a few extreme cases in that skeleton, protrusions <15 m wide are truncated by up to ~50 m. Conversely, the respective lengths of these features are generally preserved in each of the other four skeletons (Fig. 16).

The terminal one to a few meters of skeletal branches generated by the present algorithm are sometimes kinked (Figs. 15c, d, and 16d). That skeleton also invariably terminates at the boundary of the input region. The Zhang and Suen skeleton frequently reaches or very nearly reaches this boundary as well. The remaining skeletons do not generally reach the boundary (e.g., Fig. 14c).

4. Discussion

4.1. Causes of skeleton behaviors

4.1.1. Distal offset at binary tree intersections

The distal offset of skeleton intersections relative to binary tree intersections, regardless of the algorithm used, ultimately arises because each skeleton is designed to approximate the medial axis. In one conventional conceptualization (e.g., Saha et al., 2016), each point in the medial axis of a region is defined by the center of the largest circle that is locally bounded by that region at two or more points, traditionally termed a "maximal inscribed ball." Along constant-width intervals of the input region, the binary tree satisfies this definition and therefore is coincident with the medial axis (Fig. 5b). However, at each intersection, the maximal inscribed ball fills the juncture so that its center, through which the medial axis passes, lies beyond the tree's bifurcation (Fig. 5c). The skeletons approximately pass through those centers and therefore have intersections distally offset from those of the tree. These skeleton intersections must then open wider than the corresponding tree intersections so that each skeleton can lie along the axis of the next branch generation.

4.1.2. General behavior near termini

Locally maximal inscribed balls also explain the forking near input region termini in the skeletons generated by the present algorithm in the first three experiments (Fig. 5f). At these termini, each corner of the region essentially funnels a series of progressively finer maximal inscribed balls, causing the medial axis to splay and head to each of these corners. In the Yukon River experiment, the skeleton derived by the present algorithm mostly lacks this forking because templating renders the skeleton only sensitive to corners resolved at a sampling interval of

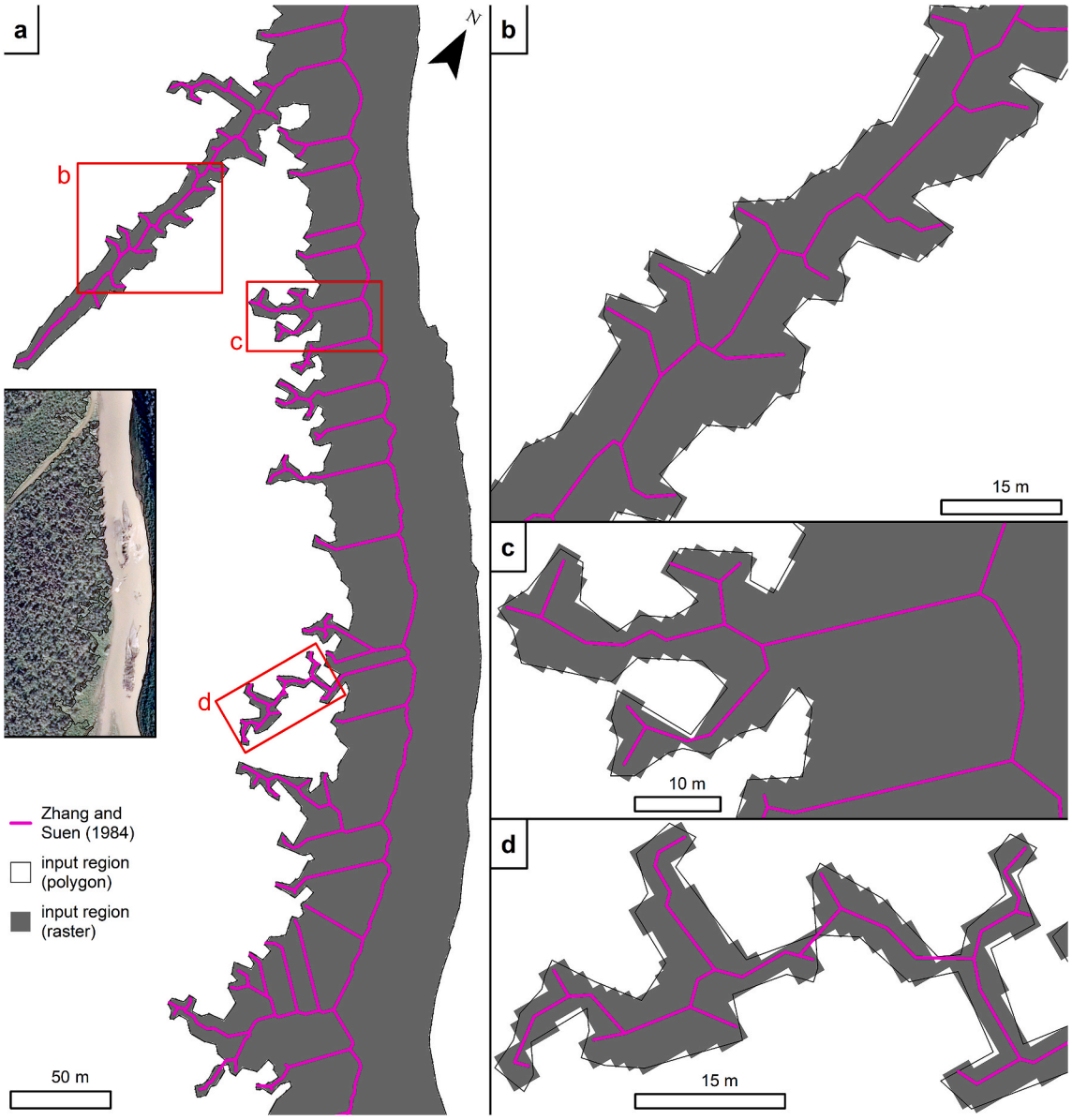


Fig. 11. Representative subset of Zhang and Suen skeleton from fourth experiment. Note jagged appearance, abundant spurious edges, uninterrupted continuity, and near topological completeness, with skeleton representing even most minor boundary irregularities (e.g., in (c)).

15 m. However, deflections of this skeleton near termini (e.g., Fig. 15d) arise from the same corner-attraction phenomenon, which, in places, is compounded by templating (e.g., Fig. 15c; section 4.1.6).

The extension of the skeletons generated by the present algorithm to each region's boundary is due to a combination of both the skeletons' correspondence to the medial axis (Schmitt, 1989) and the addition of tails (section 2.2.1). For reference, the median tail length for each of the skeletons with normalized length >2 across the four experiments is 2.5, 1.8, 1.9, and 35 cm, respectively.

4.1.3. Relative smoothness

Unlike the skeletons generated by the present algorithm, the skeletons generated by other algorithms in the first three experiments are jagged near intersections (Fig. 5c) and tree termini (e.g., Fig. 7f) at scales of ~0.025 m. This jaggedness arises from the limitations intrinsic to the raster representation of linear features. Finer cell sizes would depress the scale of this roughness but could not eliminate the roughness completely.

In the fourth experiment, the Zhang and Suen skeleton also appears jagged at coarser scales (arrows in Fig. 9b), including spurious edges usually oriented either NE–SW or NW–SE (Fig. 10a). This jaggedness is due to the discrete nature of iterative digital erosion and the generating algorithm's use of a binary 9-pixel (3x3) neighborhood to reconcile converging erosion fronts (Zhang and Suen, 1984). The relative smoothness of the skeletons generated by the RivWidthCloud, DeepRiver, and present algorithms (e.g., Fig. 14) is due to the more nearly continuous mapping of boundary geometry onto the skeleton in those algorithms. Moreover, the RivWidthCloud and present algorithms similarly depend on direct measurements of boundary distance, which accounts for the markedly similar traces of their respective skeletons.

4.1.4. Absent, shortened, and/or discontinuous intervals

Discontinuities occur in RivWidthCloud skeletons from the third and fourth experiments (e.g., Fig. 9c), and in DeepRiver skeletons from all experiments (e.g., Figs. 5e and 9d). Fundamentally, these discontinuities exist because each algorithm uses thresholding: RivWidthCloud thresholds Laplacian values derived from the distance map (Pavelsky and Smith, 2008) whereas DeepRiver thresholds skeleton

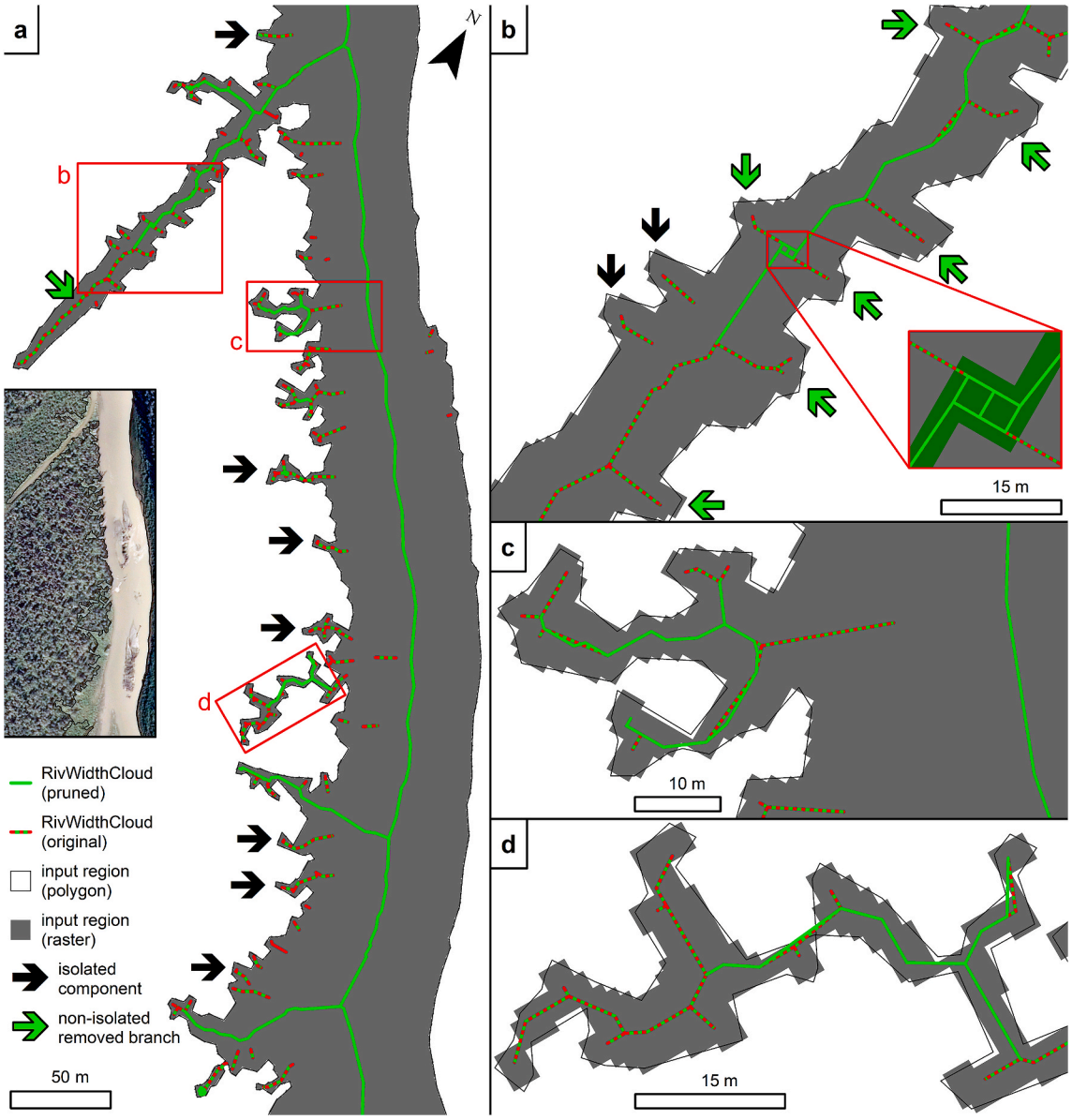


Fig. 12. Representative subset of RivWidthCloud skeleton from fourth experiment. Note smooth appearance (cf. Fig. 11) and marginal isolated components (black arrows in (a) and (b)). After branch removal, skeleton is topologically simple, with most minor boundary irregularities lacking expression (e.g., (b)), but some isolated components remain ((c) and (d)). Green arrows in (a) and (b) point to edges that are deleted by 500-pixel branch removal but are connected to main part of skeleton prior to removal. Loss of these edges results in undesirable truncation in (a) but removal of unimportant edges in (b). Inset of (b) highlights very rare example of a two-pixel wide knot in RivWidthCloud skeleton. (For interpretation of the references to color in this figure legend, the reader is referred to the Web version of this article.)

pseudoprobabilities (Isikdogan et al., 2018).

For the RivWidthCloud skeletons, the gap formed by a discontinuity can be widened when branch removal truncates the skeleton on either side of the discontinuity. Isolated skeleton intervals up to ~1000 pixels long can thus be deleted altogether by truncation back from both ends, which accounts for the most noticeable effects of branch removal in the fourth experiment (Figs. 9c and 10b). On the other hand, because branch removal halts at junctures, terminal portions of a skeleton may survive branch removal if barbs occur near termini (arrow in Fig. 7f). This behavior accounts for the increasing preservation of eighth- and ninth-generation branches in RivWidthCloud skeletons with branch removal across the first three experiments (Table A1).

The tendency of the DeepRiver skeleton to be absent in wider intervals in the second, third, and fourth experiments suggests that the algorithm could be sensitive to local width. Furthermore, this behavior seems to depend on a critical width. In the second experiment, for the pseudoprobability threshold of 0.4 that we use, half (two out of four) of the second-generation binary tree branches lack expression in the DeepRiver skeleton (Fig. 6c). These buffered branches are 6.272 m wide. Likewise, intervals wider than ~200 m in the Yukon reach generally lack expression in that experiment's DeepRiver skeleton (Fig. 9d). These respective widths correspond to 250.9 and 200 pixels, which is similar to the 255-pixel width that was the maximum used in DeepRiver's training set. This inferred critical width is also consistent with the location of the observed pseudoprobability ridge ~125 pixels from the input region boundary in the second through fourth experiments (e.g., Fig. 6b), if we interpret this ridge to lie at the center of the maximum width to which DeepRiver is sensitive.

We cannot identify any cause for the apparent azimuthal dependence of highly shortened or eliminated DeepRiver skeleton elements in the

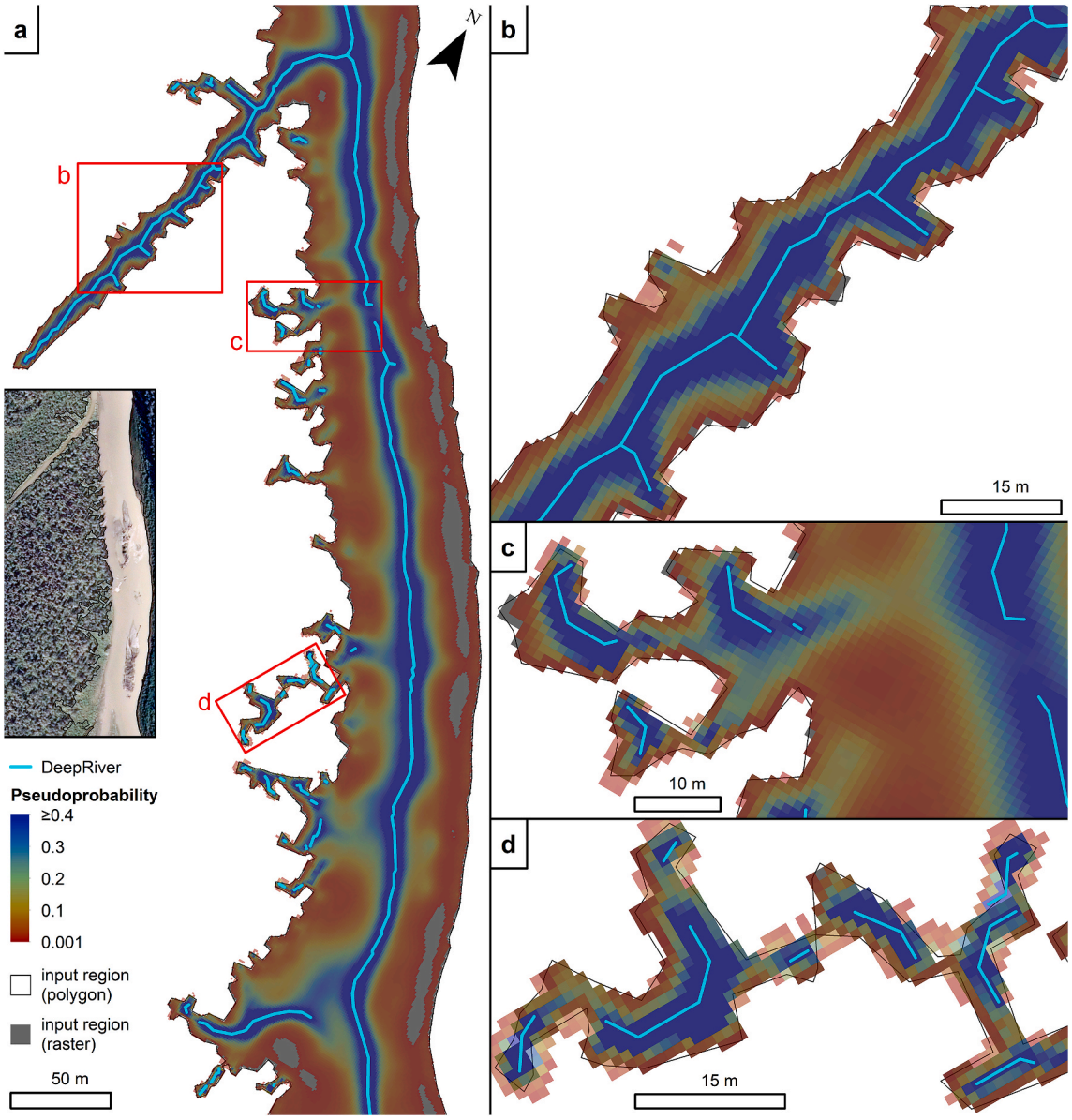


Fig. 13. Representative subset of DeepRiver skeleton from fourth experiment. Note smooth appearance (cf. Fig. 11), marginal isolated components (e.g., in (c) and (d)), spurious edges (especially in (b)), discontinuities (especially in (c) and (d)), and moderately simple topology, with many minor boundary irregularities lacking expression (cf. Figs. 11 and 12).

first three experiments. DeepRiver's training set well represents these orientations, and because images in that set were randomly rotated by 90° prior to processing, we would further expect orthogonal directions to yield equivalent results.

4.1.5. *Spurious edges*

As noted by Saha et al. (2016), each vertex of the input polygon represents a potential attractor for a Voronoi segment. More precisely, Voronoi segments on the concave side of the boundary are attracted, whether those segments belong to the skeleton or to its complement (Brandt and Algazi, 1992; Ogniewicz and Ilg, 1992), due (conceptually) to the funneling of minimal inscribed balls (section 4.1.2). For the skeleton generated by the present algorithm, the organization of spurious edges into ray-like forms in the second (Fig. 6b–d) and third (Fig. 7b–d) experiments is due to a combination of this attraction and the high local symmetry, which causes multiple spurious edges to stem from nearly the same point. In the fourth experiment, spurious edges are likewise attracted to input region vertices but their number is reduced by templating (section 2.3). Spurious edges also occur in the raster-derived skeletons of the third (arrows in Fig. 7) and fourth (Figs. 11–13) experiments and are well-documented for raster-based skeletonization in general (Lam and Lee, 1992; Saha et al., 2016). Therefore, pruning is commonly an essential step in geomorphic skeletonization, as we discuss in section 4.2.

4.1.6. *Consequences of cutting and templating*

In the Yukon River experiment, the cutting and templating used by the present algorithm results in some skeletal artifacts. In Fig. 15c, the lower protrusion is truncated because the template skeleton was cut (at point 3) and the hybrid skeleton then conformed to that truncated geometry (by matching point 1 to point 2). In a few extreme cases, this scenario results in truncation by up to ~50 m (Fig. 16). There are also instances where the simple proximity-based matching used in templating causes an end of the template skeleton to be matched to an end of the hybrid skeleton that lies in a different part of the input polygon (points 4 and 5 in Fig. 16).

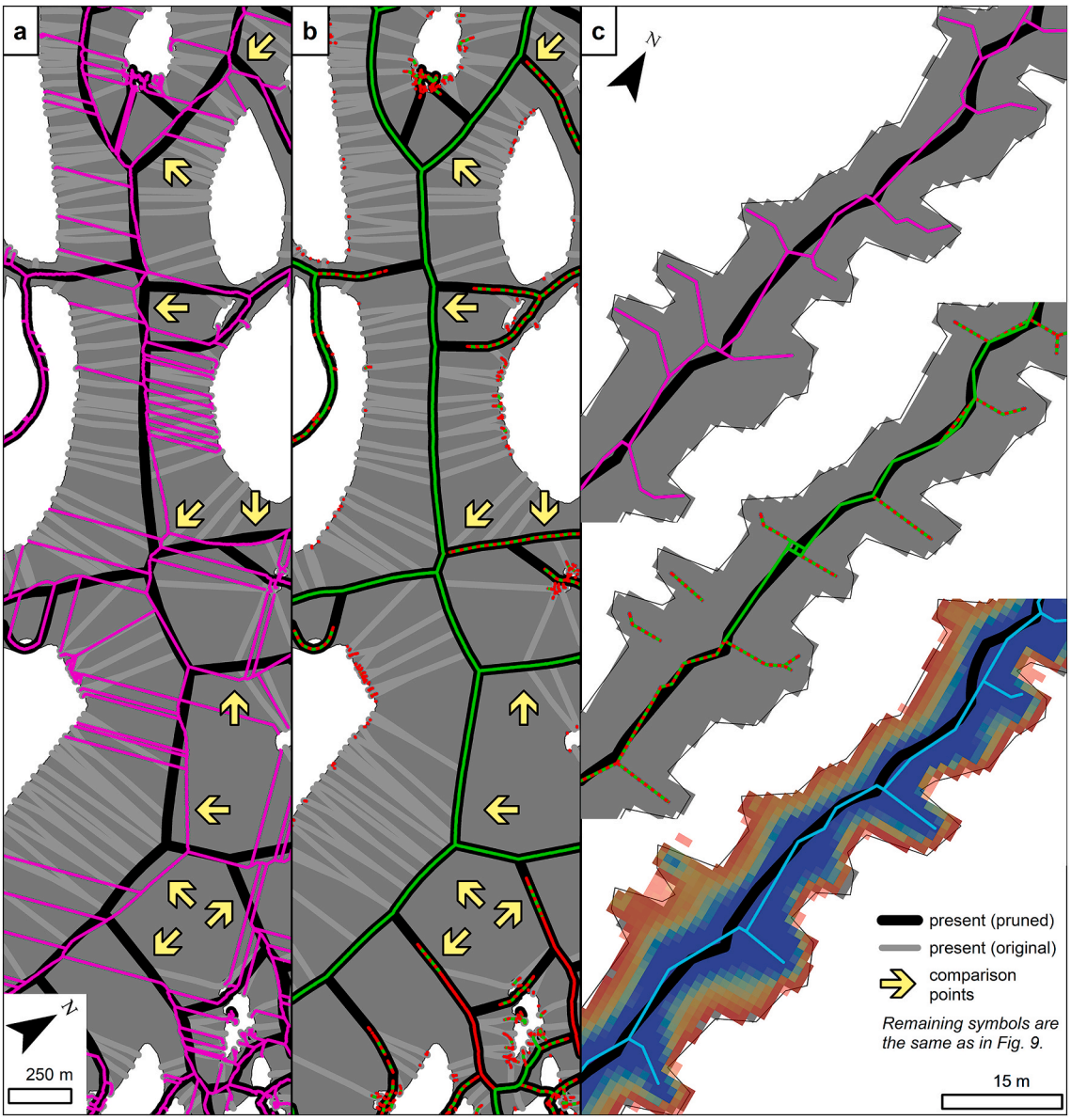


Fig. 14. Comparison, at two scales, of representative skeleton traces from fourth experiment. Note similarity between skeletons generated by RivWidthCloud and present algorithms relative to Zhang and Suen skeleton (arrows). Fig. (c) has same extent as Figs. 11b, 12b and 13b, and 15b.

Although these anomalies clearly highlight the limitations of the heuristics used in the present algorithm, it should be noted that these heuristics typically functioned as designed. For example, the proxy polygon effectively widens two constrictions of just 1 m (one pixel) in Fig. 15d to >5 m, thereby preserving the local topology in the hybrid skeleton. This is especially impressive given that a minimum width of 30 m would be required to ensure that all nodes from the template skeleton lie within the protrusion (Theorem 4.1 of Brandt and Algazi (1992)). Likewise, two neighboring, spire-like protrusions (and geomorphic artifacts) are well-represented in the hybrid skeleton despite a combined length of >500 m along which the width is everywhere <30 m, including a few necks narrower than 5 m (Fig. 17). More generally, the hybrid skeleton notably lacks pervasive issues even though 4473 nodes were cut to isolate the template skeleton.

4.2. Pruning

4.2.1. RivWidthCloud branch removal

The length-based pruning of marginal edges used by RivWidthCloud's branch removal step is generally successful at removing spurious edges in all four experiments. However, this step also removes some topologically important portions of the skeleton, including many ninth-generation branches in the first three experiments (e.g., Fig. 7f) and some braids in the fourth experiment (Figs. 9c and 10b).

Although a branch removal length tailored to each experiment would have improved results, any length would pose some problems. For example, the longest spurious edges in the RivWidthCloud skeleton of the third experiment, without branch removal, are ~1 m long, excluding one 4-m-long spurious edge that is disconnected from the main skeleton. These 1-m-long spurious edges are more than double the length of the ninth-generation branches in that experiment. Additionally, any branch removal length would widen discontinuities in topologically important portions of the skeleton (section 4.1.4).

4.2.2. Normalized length

The third experiment demonstrates the significant utility of the normalized length (Fig. 8). Of the 5146 Phase-2 branches identified by

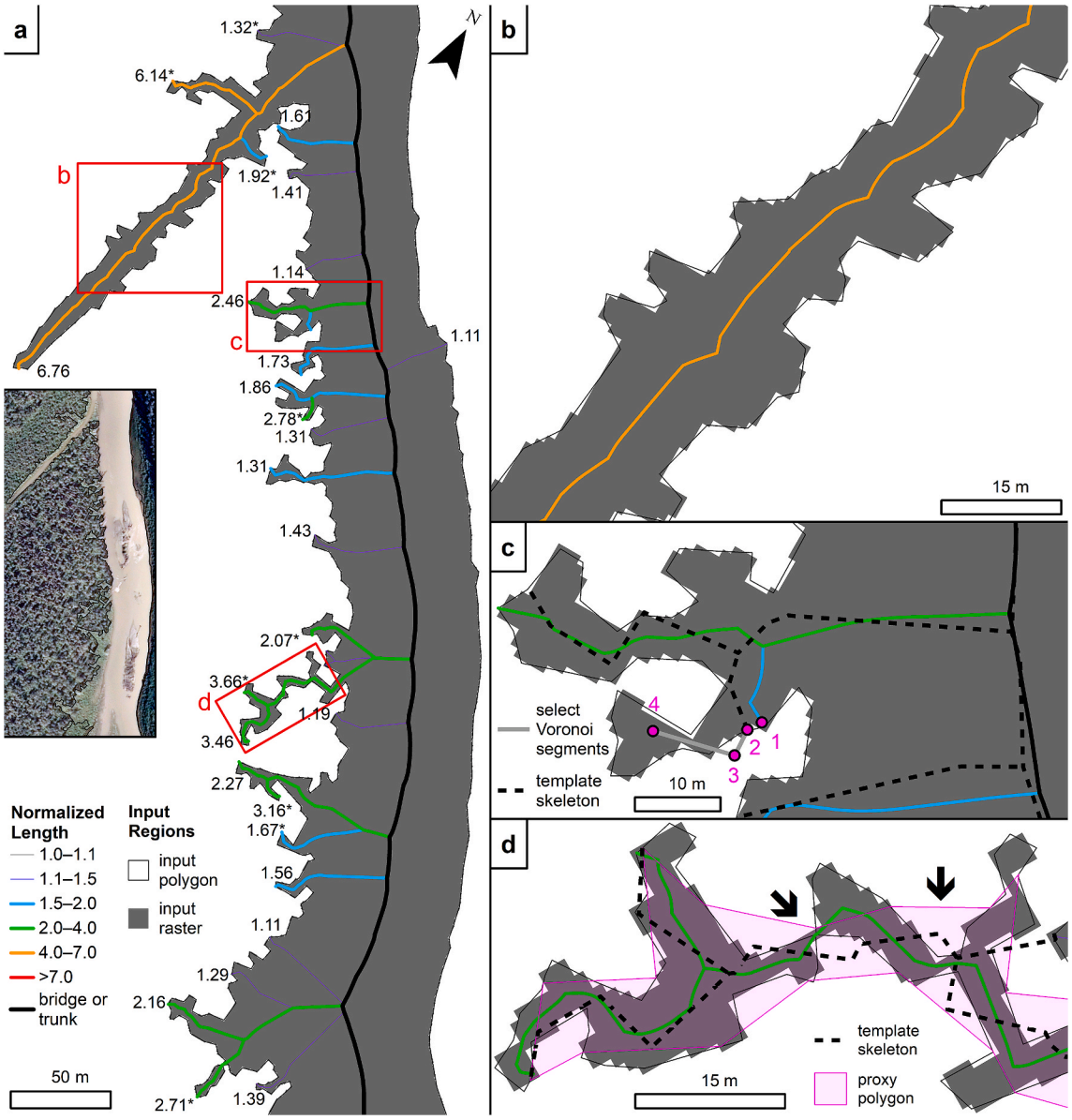


Fig. 15. Representative subset of skeleton generated by present algorithm in fourth experiment. Note smooth appearance (cf. Fig. 11), spurious edges, and uninterrupted continuity. (a) Normalized lengths (numeric labels) are intended to score branch importance; an asterisk is appended to normalized lengths of sub-branches (i.e., branches that stem from branches). (c) Truncation of a protrusion's representation due to cutting (section 2.1.2) and templating (section 2.3). To isolate template skeleton, hub at (3) was cut, thereby discarding node at (4), and other nodes, as though part of complement. Stub (1) at end of a spurious edge in fine graph skeleton was therefore matched by proximity to tailed end (2) of (cut) template skeleton during derivation of hybrid graph skeleton (Fig. A1). A tail was then added at (1) during Phase 2 (section 2.2.1). If template skeleton node at (4) had been retained, representation would not be truncated, as protrusion is represented in fine graph skeleton. (d) Proxy polygon used to isolate template skeleton is shown. This polygon's vertices are sourced from boundary samples whose Voronoi analysis is used to generate template skeleton. Note how this proxy polygon widens two single-pixel constrictions (arrows) of input region, thereby safeguarding topological representation of protrusion in template skeleton, which is then replicated in final skeleton.

the present algorithm in that experiment, all 4651 branches (66.7% of the total skeleton length) that do not correspond to tree elements have a normalized length <1.39. All but 12 of the remaining 495 (legitimate) branches have normalized lengths >4.59. These 12 Phase-2 branches correspond to ninth-generation branches in the binary tree, specifically, to 12 of the 16 such branches that merge at their respective stems (Fig. 8f). These Phase-2 branches have normalized lengths in the range 3.37–4.56, except for a single branch with a normalized length of 2.51. The normalized length therefore strongly distinguishes between branches, despite spurious branches up to 11.7 m long and legitimate branches as short as 0.31 m.

The normalized length likewise provides a good estimate of branch significance in the Yukon River experiment across a wide range of scales (Figs. 10c and 15). However, in typical geomorphic applications, significance also depends on branch length. For this reason, our code supports combining minimums for both normalized length and branch length.

4.3. The present algorithm in context

The basic framework of Phase 1 (section 2.1), namely Voronoi analysis followed by segment filtering, is decades old (e.g., Brandt and Algazi, 1992; Kirkpatrick, 1979; Lee, 1982). Although we independently developed some of the implementation details of Phase 1 (section 2.1.2), including the cutting heuristic and testing hubs against the proxy polygon, it is difficult to determine whether these details are novel. More

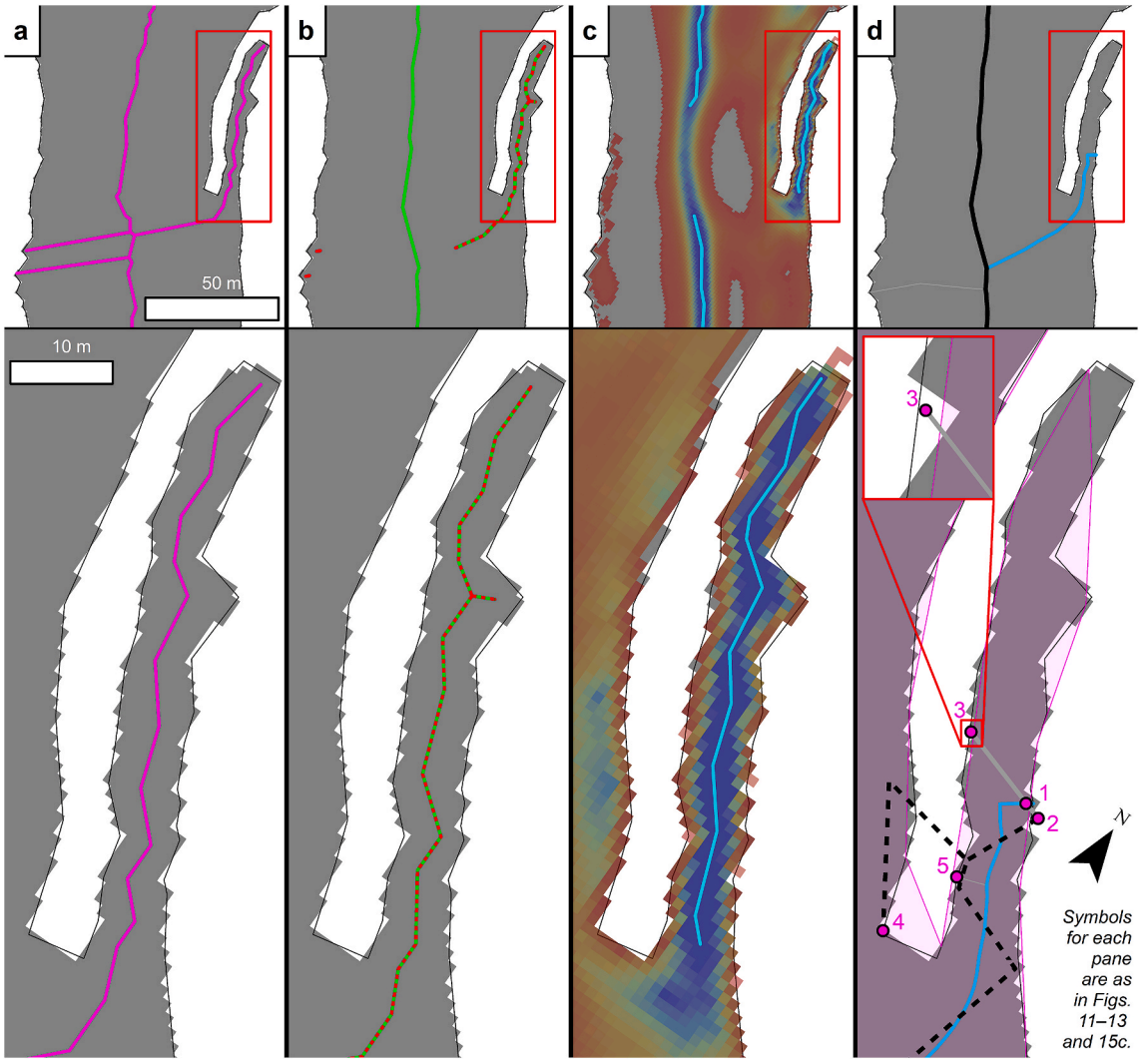


Fig. 16. Extreme example of truncation of skeleton generated by present algorithm due to cutting (section 2.1.2) and templating (section 2.3). Although protrusion is well represented in (a) Zhang and Suen, (b) RivWidthCloud without branch removal, and (c) DeepRiver skeletons, its expression in (d) skeleton generated by present algorithm is truncated by ~45 m. To isolate template skeleton, hub at (3) was cut because it lies outside proxy polygon, which is locally narrower than input polygon by ~0.4 m (inset). This cutting caused nodes nearer protrusion terminus to be discarded as though part of complement. Stub (1) at end of a spurious edge in fine graph skeleton was therefore matched by proximity to tailed end (2) of (cut) template skeleton during derivation of hybrid graph skeleton (Fig. A1). A tail was then added at (1) during Phase 2 (section 2.2.1). Protrusion was well represented in fine graph skeleton prior to conforming it to template. In addition, note how simple proximity-based matching of templating procedure matched tailed end of template skeleton (4) to fine graph skeleton stub (5) on opposite side of an input region gap, thereby altering local topology intrinsic to template skeleton and causing spurious edge to which stub (5) belongs to be retained (until filtered out by normalized length criterion).

generally, the proxy polygon and especially the template skeleton (section 2.3) belong to the well-explored category of multi-scale skeletonization (e.g., Pizer et al., 1987; Saha et al., 2016), in which the scale-dependence of skeletonization is leveraged to modulate sensitivity.

Compared to Phase 1, Phase 2 is relatively novel. Although Attali Sanniti di Baja and Thiel (1997) (in one of their algorithms) and Marie et al. (2016) each adopt a pruning metric conceptually related to the scale of the boundary protrusion, neither is scaled to a width, as the normalized length is. In the algorithm of Giesen et al. (2009) and its extension by Postolski et al. (2014), pruning is based on a metric that is scaled to width, but that width is measured near the stub end rather than at the branch stem, as for the normalized length. Shortest-path routing has previously been used in the analysis (Bai and Latecki, 2008) and extraction (Pavelsky et al., 2013) of skeletons, but we are not aware of a prior use to directly support a pruning criterion.

5. Conclusions

1. The vector-based algorithm that we describe generates skeletons that are simultaneously smooth and continuous, unlike those generated by each of the three raster-based algorithms that we considered.
2. The novel normalized length criterion provides a highly useful metric for the removal of undesired skeleton components in geomorphic applications.
3. For a river polygon, the normalized length criterion can be approximately conceptualized as the ratio between a stream's length and the half-width of the stream into which it flows at its confluence.
4. Built-in heuristics significantly facilitate the skeletonization of noisy and/or complicated polygons by the present algorithm but can result in artifacts.

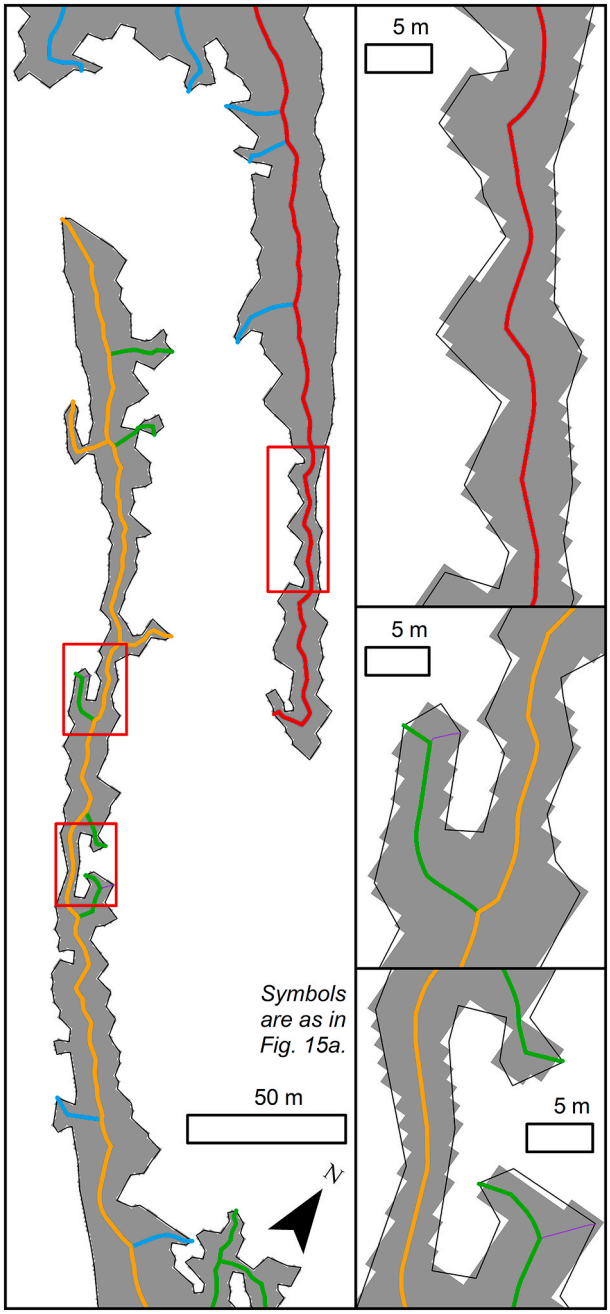


Fig. 17. Extreme example of topological preservation along narrow intervals in skeleton generated by present algorithm. These neighboring spire-like protrusions, which are geomorphic artifacts (center-right inset of Fig. 9), have a combined length of >500 m along which their width is <30 m, including a few necks narrower than 5 m (right panes). Nonetheless, these protrusions are well represented in skeleton despite use of a 15-m template sampling interval, which only guarantees topological preservation along intervals at least 30 m wide (Theorem 4.1 of Brandt and Algazi (1992)).

Computer code availability

The present algorithm is implemented as part of the numgeo Python package (~1 MB), developed by EIS and first available March 2019. Example data and full source code are freely available at https://github.com/eischaefer/numgeo. All required dependencies are open-source and freely available from https://pypi.org: gdal, numpy, psutil, and scipy.

Authorship statement

EIS developed the algorithm and its implementation, and was the primary author of the manuscript. JDP advised on experiments and revision of the manuscript.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

This material is based upon work supported by the National Science Foundation Graduate Research Fellowship Program under Grant No. DGE-1143953. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation. The National Science Foundation was not involved with the design or execution of the study, writing of this manuscript, or the decision to submit. We thank Tamlin Pavelsky and an anonymous reviewer for helpful comments that improved this manuscript.

Appendix A. Supplementary data

Supplementary data to this article can be found online at https://doi.org/10.1016/j.cageo.2020.104554.

References

Allen, G.H., Pavelsky, T.M., 2018. Global extent of rivers and streams. Science 361, 585–588. https://doi.org/10.1126/science.aat0636.
Attali Sanniti di Baja, G., Thiel, E.D., 1997. Skeleton simplification through non significant branch removal. Image Process. Commun. 3, 63–72.
Bai, X., Latecki, L.J., 2008. Path similarity skeleton graph matching. IEEE Trans. Pattern Anal. Mach. Intell. 30, 1282–1292. https://doi.org/10.1109/TPAMI.2007.70769.
Barber, C.B., Dobkin, D.P., Huhdanpaa, H., Huhdanpaa, H., 1996. The quickhull algorithm for convex hulls. ACM Trans. Math Software 22, 469–483. https://doi.org/10.1145/235815.235821.
Brandt, J.W., Algazi, V.R., 1992. Continuous skeleton computation by Voronoi diagram. CVGIP Image Underst. 55, 329–338. https://doi.org/10.1016/1049-9660(92)90030-7.
Clubb, F.J., Mudd, S.M., Milodowski, D.T., Valters, D.A., Slater, L.J., Hurst, M.D., Limaye, A.B., 2017. Geomorphometric delineation of floodplains and terraces from objectively defined topographic thresholds. Earth Surf. Dyn. 5, 369–385. https://doi.org/10.5194/esurf-5-369-2017.
Giesen, J., Miklos, B., Pauly, M., Wormser, C., 2009. The scale axis transform. In: Proceedings of the Annual Symposium on Computational Geometry. ACM Press, New York, New York, USA, pp. 106–115. https://doi.org/10.1145/1542362.1542388.
Greenwood, M.C., Humphrey, N., 2002. Glaciated valley profiles: an application of nonlinear regression. In: Proceedings of the 34th Symposium on the Interface, pp. 452–460. Montréal, Canada.
Grieve, S.W.D., Hales, T.C., Parker, R.N., Mudd, S.M., Clubb, F.J., 2018. Controls on zero-order basin morphology. J. Geophys. Res. Earth Surf. https://doi.org/10.1029/2017JF004453, 2017JF004453.
Harbor, J.M., Wheeler, D.A., 1992. On the mathematical description of glaciated valley cross sections. Earth Surf. Process. Landforms 17, 477–485. https://doi.org/10.1002/esp.3290170507.
Horton, R.E., 1945. Erosional development of streams and their drainage basins; Hydrophysical approach to quantitative morphology. GSA Bull. 56, 275–370. https://doi.org/10.1130/0016-7606(1945)56[275:edosat]2.0.co;2.
Isikdogan, F., Bovik, A., Passalacqua, P., 2018. Learning a river network extractor using an adaptive loss function. IEEE Geosci. Remote Sens. Lett. 15, 813–817. https://doi.org/10.1109/LGRS.2018.2811754.
Karimipour, F., Ghandehari, M., Ledoux, H., 2013. Watershed delineation from the medial axis of river networks. Comput. Geosci. 59, 132–147. https://doi.org/10.1016/J.CAGEO.2013.06.004.
Kirkpatrick, D.G., 1979. Efficient computation of continuous skeletons. In: 20th Annual Symposium on Foundations of Computer Science. IEEE, pp. 18–27. https://doi.org/10.1109/SFCS.1979.15.
Lam, L., Lee, S.W., 1992. Thinning methodologies—a comprehensive survey. IEEE Trans. Pattern Anal. Mach. Intell. 14, 869–885. https://doi.org/10.1109/34.161346.
Lee, D.T., 1982. Medial Axis transformation of a planar shape. IEEE Trans. Pattern Anal. Mach. Intell. 363–369. https://doi.org/10.1109/TPAMI.1982.4767267.

Mandelbrot, B.B., 1982. The Fractal Geometry of Nature. W.H. Freeman, New York. https://doi.org/10.2307/3615422.
Mandelbrot, B.B., Frame, M., 1999. The canopy and shortest path in a self-contacting fractal tree. Math. Intel. 21, 18–27. https://doi.org/10.1007/BF03024842.
Marie, R., Labbani-Igbida, O., Mouaddib, E.M., 2016. The Delta Medial Axis: a fast and robust algorithm for filtered skeleton extraction. Pattern Recogn. 56, 26–39. https://doi.org/10.1016/J.PATCOG.2016.02.011.
Monegaglia, F., Zolezzi, G., Güneralp, I., Henshaw, A.J., Tubino, M., 2018. Automated extraction of meandering river morphodynamics from multitemporal remotely sensed data. Environ. Model. Software 105, 171–186. https://doi.org/10.1016/J.ENVSOFT.2018.03.028.
Ogniewicz, R., Ilg, M., 1992. Voronoi skeletons: theory and applications. In: Proceedings 1992 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. IEEE Comput. Soc. Press, pp. 63–69. https://doi.org/10.1109/CVPR.1992.223226.
Pattyn, F., Van Huele, W., 1998. Power law or power flaw? Earth Surf. Process. Landforms 23, 761–767. https://doi.org/10.1002/(SICI)1096-9837(199808)23:8<761::AID-ESP892>3.0.CO;2-K.
Pavelsky, T.M., Smith, L.C., 2008. RivWidth: a software tool for the calculation of river widths from remotely sensed imagery. IEEE Geosci. Remote Sens. Lett. 5, 70–73. https://doi.org/10.1109/LGRS.2007.908305.
Pavelsky, T.M., Smith, L.C., Durand, M., Alsdorf, D., 2013. RivWidth 0.4.
Pizer, S.M., Oliver, W.R., Bloomberg, S.H., 1987. Hierarchical shape description via the multiresolution symmetric Axis transform. IEEE Trans. Pattern Anal. Mach. Intell. PAMI 9, 505–511. https://doi.org/10.1109/TPAMI.1987.4767938.
Postolski, M., Couprie, M., Janaszewski, M., 2014. Scale filtered Euclidean medial axis and its hierarchy. Comput. Vis. Image Understand. 129, 89–102. https://doi.org/10.1016/J.CVIU.2014.07.003.
Saha, P.K., Borgefors, G., Sanniti di Baja, G., 2016. A survey on skeletonization algorithms and their applications. Pattern Recogn. Lett. 76, 3–12. https://doi.org/10.1016/J.PATREC.2015.04.006.
Schaefer, E.I., 2019. Data for: an algorithm to reduce a river network or other graph-like polygon to a set of lines, V1. Mendeley Data. https://doi.org/10.17632/h4pygk4xrd.1.
Schmitt, M., 1989. Some Examples of Algorithms Analysis in Computational Geometry by Means of Mathematical Morphological Techniques. Springer, Berlin, Heidelberg, pp. 225–246. https://doi.org/10.1007/3-540-51683-2_33.
van der Walt, S., Schönberger, J.L., Nunez-Iglesias, J., Boulogne, F., Warner, J.D., Yager, N., Gouillart, E., Yu, T., 2014. Scikit-image: image processing in python. PeerJ. https://doi.org/10.7717/peerj.453, 2014.
Yang, X., Pavelsky, T.M., Allen, G.H., Donchyts, G., 2019. RivWidthCloud: an automated Google Earth engine algorithm for river width extraction from remotely sensed imagery. IEEE Geosci. Remote Sens. Lett. 1–5. https://doi.org/10.1109/LGRS.2019.2920225.
Zhang, T.Y., Suen, C.Y., 1984. A fast parallel algorithm for thinning digital patterns. Commun. ACM 27, 236–239. https://doi.org/10.1145/357994.358023.