Interactive comment on Computing water flow through complex landscapes, Part 3: Fill-Spill-Merge: Flow routing in depression hierarchies

Richard Barnes[1,2,3], Kerry L. Callaghan[4,5], and Andrew D. Wickert[4,5]

[1]Energy & Resources Group (ERG), University of California, Berkeley, USA
[2]Electrical Engineering & Computer Science, University of California, Berkeley, USA
[3]Berkeley Institute for Data Science (BIDS), University of California, Berkeley, USA
[4]Department of Earth Sciences, University of Minnesota, Minneapolis, MN, USA
[5]Saint Anthony Falls Laboratory, University of Minnesota, Minneapolis, MN, USA

Correspondence: Richard Barnes (richard.barnes@berkeley.edu)

1 Anonymous ref 1:

Comment: figure 1: - I suggest to number the depressions in the figure to make it easier for the reader to follow the explanations.

Response: Thank you for the suggestion, we've added numberings to the depressions.

Comment: lines 70–83: I think that if the authors refer to the subfigures it will be easier to read. I found that fig. 2e has the elements I needed to understand the text.

Response: Thank you for this suggestion, we've added subfigure references to Figure 2 throughout the paper, except in cases where it seems as though referencing the entirety of the figure is more appropriate.

Comment: line 78: when #0 is mentioned in fig. 2, I had a bit of trouble finding it in the figure (it's a bit hidden under "Ocean", and the purple color of the square in 2a is a bit dark).

Response: We've left 0 under the ocean, but have changed the text in the purple square to white.

Comment: line 93: figure 2e; line 100: figure 2a

Response: We have added these specific figure numbers. The sentence in question previously read

> [...] margin of water contained within the parent (i.e., the “marginal volume" indicated on Figure 2).

It now reads

> [...] margin of water contained within the parent (i.e., the “marginal volume" indicated on Figure 2e).

and

> These are the dashed lines shown in Figure 2.

now reads

> These are the dashed lines shown in Figure 2a

Comment: line 173: I was a bit confused by the “water than they can hold spill" here. Is it "water than they can hold will spill"?

Response: The sentence previously read

> Water must be redistributed such that leaf depressions containing more water than they can hold spill over into their neighboring depression.

We have rephrased this sentence to clarify:

> When a leaf depression initially contains more water than it can hold, the water will be redistributed by spilling over into the neighboring depression.

Comment: line 382: maybe insert a citation for GRASS GIS here? (Neteler et al 2012)

Response: We have added the requested reference. The text previously read

> [...] for analysis and manually removed several road bridges using GRASS GIS to prevent artificial pooling behind these [...]

It now reads

> for analysis and manually removed several road bridges using GRASS GIS (Neteler et al., 2012) to prevent artificial pooling behind these [...]

Comment: figures 6 and 7: Have you thought about using a divergent colorscale here? Since you are showing positive and negative differences?

Response: We considered several colorscale options, including a divergent colorscale, in light of this comment. After comparing several options, we decided that a continuous colorscale with discrete classes was most effective, since this allows a viewer to more easily differentiate positive and negative values. Since the values in question are generally small, we also switched to a logarithmic scaling of the colorbar.

Comment: Is the drainage network from both algorithms compared here are identical, differing only at the depressions? I was curious to see the drainage.

Response: We have included figures 1–4 showing the drainage networks from both algorithms in this response, but have not added these to the paper because we did not think that these added significant value to the paper. The drainage networks are extremely similar between the two sets of results, with minor differences at locations where the depression fills differ.

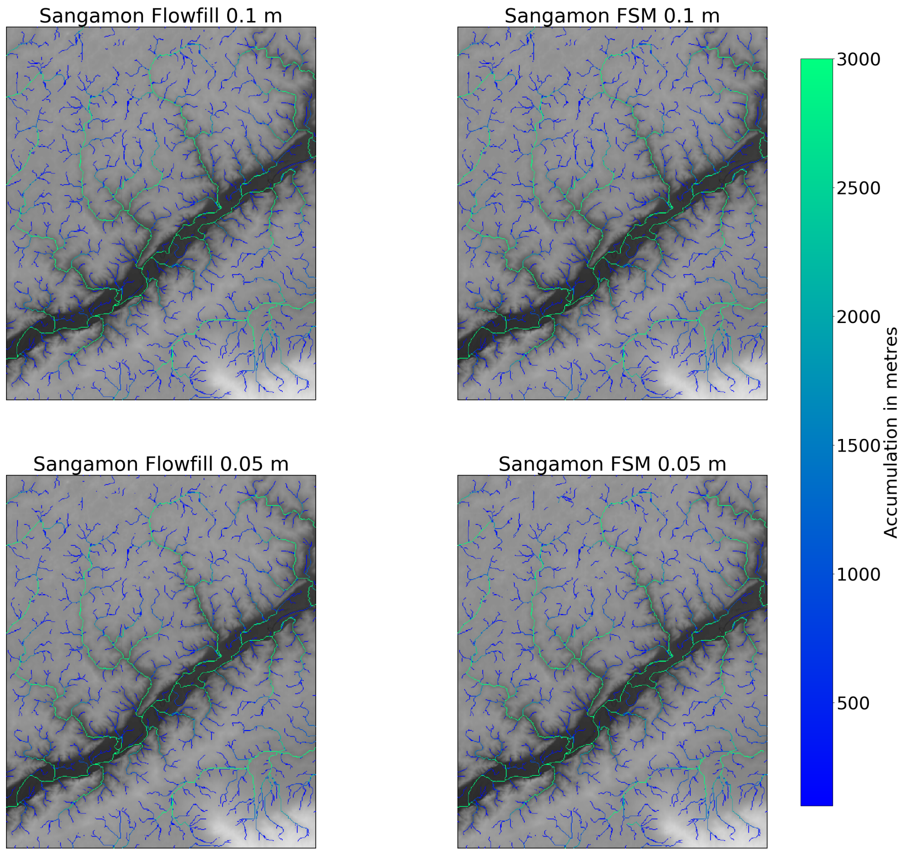

Figure 1. Comparison between drainage networks created over a surface filled using FlowFill versus a surface filled using FSM in the Sangamon River basin. Results appear close to identical for 0.1 m of runoff. With 0.05 m of runoff, minor differences are visible in the upper left portion of the figure.

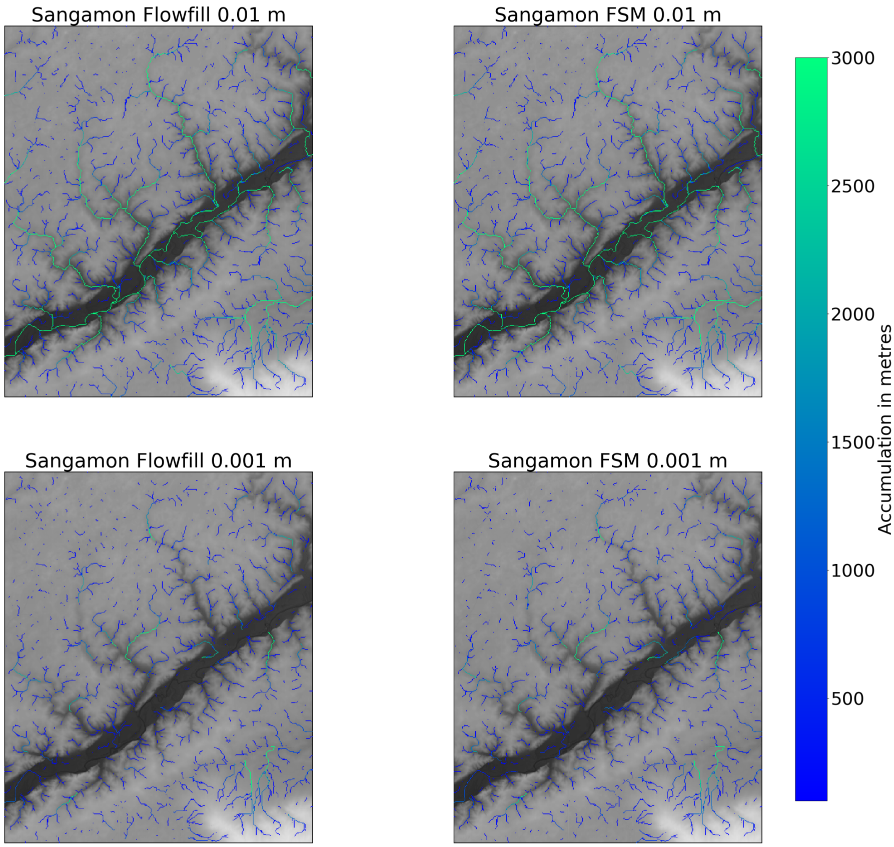

Figure 2. Comparison between drainage networks created over a surface filled using FlowFill versus a surface filled using FSM in the Sangamon River basin. Results appear near identical for 0.01 m and 0.001 m of runoff.

2 Ref 2, Daniel Hobley

Comment: I found the way the depression hierarchy is talked about in the Algorithm section quite confusing. I think the reason is that the authors convolve the object that is the hierarchy itself (i.e., the input to this algorithm), the conceptual idea of the DH, and the method used to build that hierarchy (as explained in Barnes et al. (2020)). For example

- around P5ln30 “The specific outputs from the depression hierarchy [the method] that are used in the Fill-Spill-Merge algorithm are: — DH: the depression hierarchy [the object] itself....";
- P8ln2 “By routing water according to the DH..."

So is this the object, or the method? Finding a distinct way to refer to the method/algorithm that produces a DH, the ordering that is described by a DH, and a DH structure itself would significantly enhance clarity throughout. (See line item P9ln26 also.)

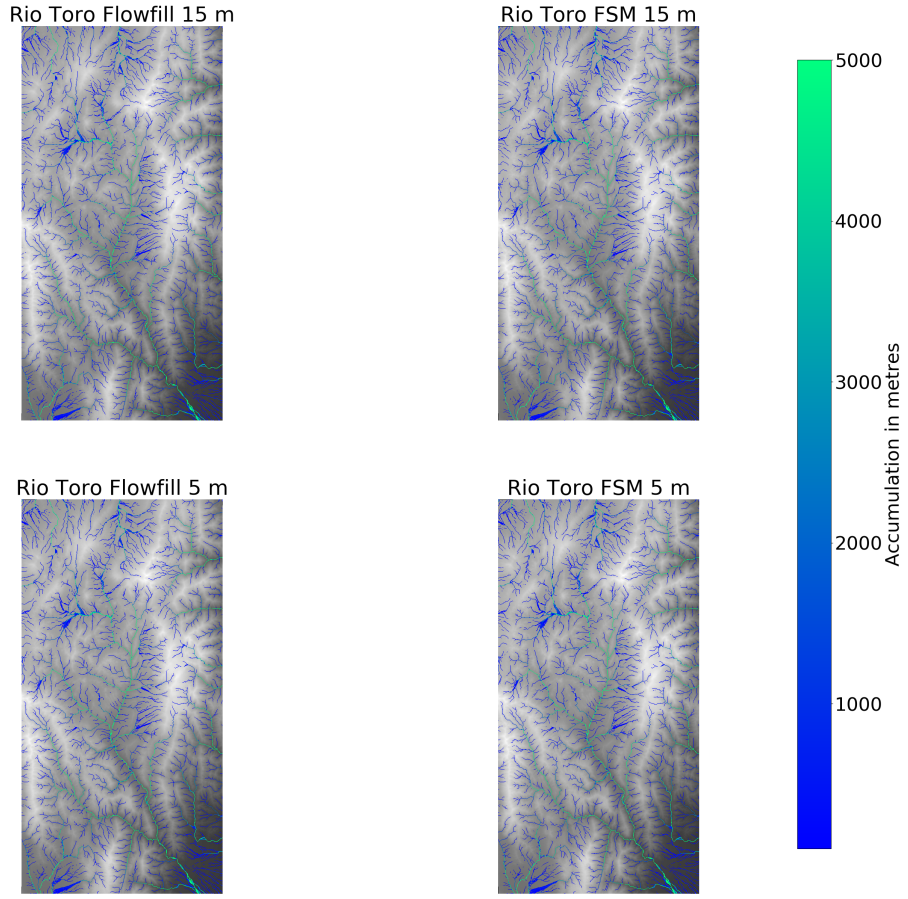

Figure 3. Comparison between drainage networks created over a surface filled using FlowFill versus a surface filled using FSM in the Río Toro basin. Results appear near identical for 15 m and 5 m of runoff.

Response: Thank you for pointing this out. We have normalized the terminology so that “depression hierarchy" should always refer to the data structure itself. We feel that the data structure and the conceptual idea of the DH are closely aligned and so do not distinguish explicitly between them. However, we now make it clear when we are refer to the algorithms or construction of the DH.

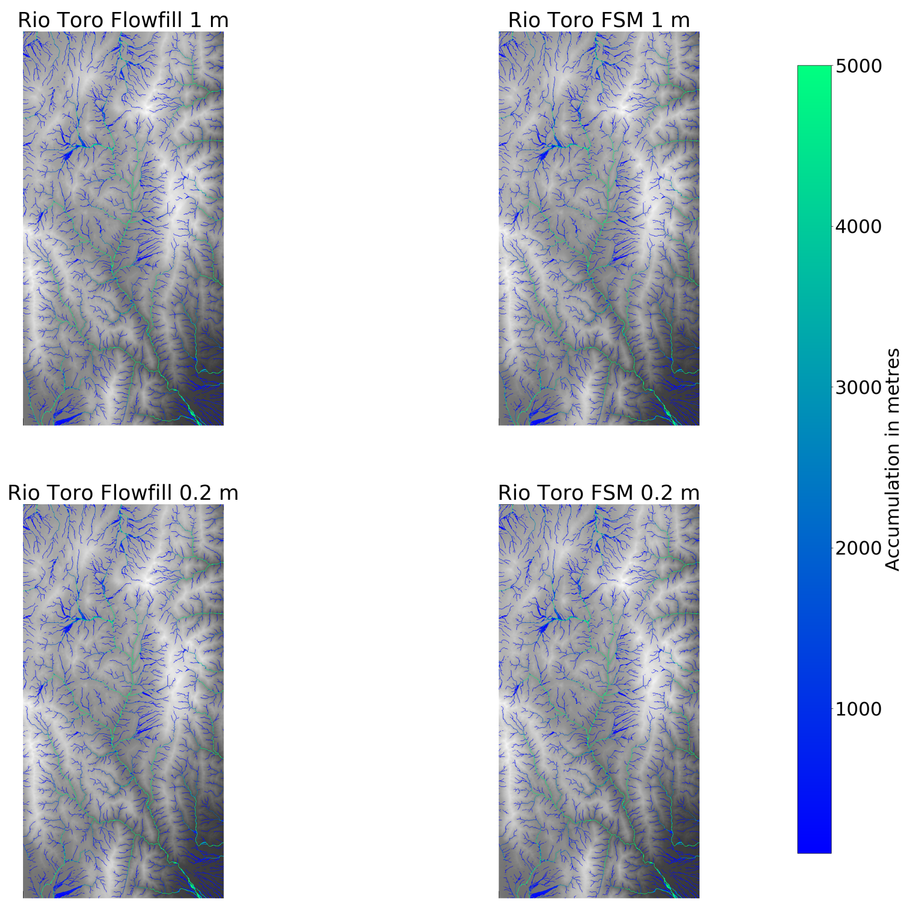

Figure 4. Comparison between drainage networks created over a surface filled using FlowFill versus a surface filled using FSM in the Río Toro basin. Results appear near identical for 1 m and 0.2 m of runoff.

Comment: Jargon suddenly appears at the end of p. 9. The whole section P9ln25–P10ln17 I found very confusing. I ramble about this at length in the line comments below, but at heart this needs a. the jargon explaining, b. the text simplifying, and c. the text expanding as necessary.

Response: As detailed below, we have added additional background material and citations to the paper to help explain the terminology. We have also gone through and tried to clarify the wordings.

Comment: Regardless of that, I would strongly advocate that you put some actual pseudocode into the paper – maybe as supplemental material, but in the body would also be good. This would really help in sections like that one, and the one to follow (both noted below).

Response: We had meant the source code accompanying the paper to serve as an additional reference and now state this explicitly in the paper:

> This paper is also accompanied by complete, well-commented source code; the reader may find it helpful to download this code and refer to it as an additional reference.

We have also included pseudocode for the nested traversals, since they seemed particularly problematic. This is included at the end of the paper and referred to at various points in the text.

Comment: Similarly, if iterations like those on P10 are described, then I think they would be much clearer with cartoons illustrating the possible steps next to them. You use fig 2 as a case study for the sequence on P11, but it would be much clearer to produce a second version along the lines of fig 2 but with multiple parts. These would highlight each inner loop as described in the text, what happens under the possible configurations of that loop, and the sequencing of the water filling in under the rules for those configurations.

Response: We considered creating a figure to highlight the inner and outer loops of the traversal, but decided against it for two reasons:

1. The pseudocode, written in response to your above comment, as well as our revised text, significantly clarify the outer and inner traversals and their roles.

2. The traversals are important to the algorithmic approach, but are not essential for a conceptual understanding of how FSM moves water across a landscape.

As a result of these considerations, we thought that the best layering of information would be a figure to help guide the reader through the essential conceptualization of the flow-routing process (cf. especially Fig. 2 in the revised manuscript), with the improved textual description and pseudocode available to help the reader curious to understand the inner workings of the algorithm. We hope that this nonetheless addresses the spirit of this comment.

Comment: Oceanlink-geolink terminology. I like the conceptual framework of the paper a lot, but the description of “paired oceanlink-geolinks" I found confusing. Because there can never be an oceanlink without a geolink, I would tweak the terminology such that an oceanlink is a subset of a geolink, and the pair concept goes away.

Response: We now separate the concepts of geolinks and oceanlinks entirely. Geolinks move water within a binary tree and oceanlinks move water between binary trees.

Comment: The maths as described on P13 is not strictly correct. The equations give summed elevations, but they are variously described as “capacity" or “volume". There should be cell areas in here somewhere. Also, there is a missing definition of N. Check the maths again for precision of terms.

Response: Thank you for this comment. We now define N, as requested. The other difficulty arose because we implicitly assumed that the problem was scaled such that cells have an area of 1. We have now generalized the math to cells of arbitrary area before showing this special case.

Comment: Section 5 – The text asserts that $O(N \log N)$ is the best fit, but the paper does not show it. At the very least, plot the data in Table 1... however, doing this will reveal that the data points do not in fact give a good smooth correlation, convincing of $N \log N$ (which is presumably why you didn’t do it in the first place!) I would retain Table 1 as nice examples of “real world" model performance, but add a synthetic test with random noise elevation on varying size grids. This should give you the convincing $N \log N$ scaling to plot up and push a $N \log N$ scaling best fit through, and is also a cleaner — and more standard - way making the point about actual performance of your implementation. Some light restructuring (see below) would help here too.

Response: We have added an addition figure showing how the performance of the algorithm scales on synthetically generated datasets. We also add an additional paragraph:

> To more precisely demonstrate performance, we run Fill-Spill-Merge on synthetic landscapes of various sizes generated using RichDEM's Perlin noise random terrain generator. Multiple landscapes are generated and timed at each size to smooth timing variation due to both the data and fluctuations in the testing environment. This results in Figure X, which again shows that the performance data gives a good fit to an $N \log N$ function.

Comment: You could add a statement somewhere that this algorithm also permits water to be retained in the depressions between runs, if that is desired – so you can get a timeseries as the lakes fill over time. It would also easily let you perform true water balances on the lakes if the user wanted to, by subtracting water between steps. This is implicit in the text, but it's a cool feature you could emphasise a bit more if you want.

Response: Thank you for this suggestion, we have added several notes about this.

– §3: "Note that the landscape may already have standing water at this stage."

– §3.1: "(perhaps referring to a matrix of rainfall values, but also existing stores of standing water)"

– §6: "If standing water is retained between invocations of Fill-Spill-Merge, and new water added at each invocation, the algorithm can be used to simulate the movement of water across landscapes; we will explore this further in future work."

Comment: P1ln16 – misformatted ref

Response: Fixed, thank you. The sentence previously read

> [...] and cratering Cabrol and Grin (1999).

It now reads

> [...] and cratering (Cabrol and Grin, 1999).

Comment: P3ln15 — 0 seems a special case here. Refer also to 12 or 11?

Response: We believe you’re referring to the definition of root. The sentence previously read

> Root: A depression, such as #0 in Figure 2, that has no parent. This term may also refer to any node that is used as the starting point for a traversal that only considers the node and its descendants.

It now reads

> Root: A depression, such as #0, #11, and #12 in Figure 2, that has no parent. This term may also refer to any node that is used as the starting point for a traversal that only considers the node and its descendants.

Comment: P5ln10 – The ocean links. You should make it clear here why, from a modelling perspective, you need to separate ocean links from geolinks, so we know where we’re going. Also, you have inconsistency between “ocean link” and “oceanlink" – check for consistency (incl. In figure captions).

Response: To address this comment we have added a new paragraph to the end of §2:

> Within the algorithm, oceanlinks and geolinks are used for different purposes: an oceanlink tells us that the depression in question has grafted onto the leaf node of another tree of the depression hierarchy, locating a route for overflowing water to eventually reach the ocean. The depression to which it is oceanlinked is considered its parent, but it is not the child of that depression because water flows only one way along an oceanlink. In Figure 2a, depression #4 can be considered the parent of #12, but #12 is not the child of #4. This is because #12 is not physically contained within #4, but #12 will send all of its overflowing water to #4, as shown in Figure 2b–e. #4 will not contain the total water volume contained within #12, unlike other parents. Geolinks are more general: not every depression has an oceanlink, but every depression has a geolink. The oceanlink provides information about the type of overflow happening, but it is the geolink that will be used to track the actual direction of spill of water in the depression hierarchy.

We have normalized to the term “oceanlink".

Comment: Fig 4. This caption could be a lot more generous to the reader. Explain the colorcoding; what *exactly* the blue shading is; and exactly what you mean by “non-additive”. I don’t quite follow, and feel there must be a clearer way of expressing this. Also, would it not be clearer to actually let area represent volume of water? (I assume that this is not the case is what you are getting at by “non-additive”?) In particular g–j are quite hard to follow and need more text. Isn’t j inconsistent with f? You appear to suggest in f the whole system floods out, but j does not show this. Format-wise, you have a,b,c on the fig and A,B,C in the text.

Response: We adjusted this figure to make the height of blue bars represent an actual volume of water. This removes the need for the 'non-additive' comment, and should reduce confusion in interpretation of the figure. We have also changed the caption to add more information. The caption used to read:

> **Visual Overview of the Algorithm.** In this figure the heights of the water bars are non-additive: only the changes between panels are important. The algorithm consists of three major stages (Figure 5). From its initial distribution (A), water is moved downhill into pit cells (B, §3.1). Water is then moved within the depression hierarchy (C–F, §3.2): water in depressions with insufficient volume overflows first into their sibling depressions (D) and then – if the sibling depression becomes filled – passes to their parents (E, F). Any leftover water overflows into the ocean (F) and is forgotten. Depressions to be flooded are then identified and flooded (§3.3) starting from an arbitrarily-chosen pit cell (G–J).

It now reads:

> **Visual Overview of the Algorithm.** Black outlines represent the elevations of the cells. Blue areas are the heights of water in each cell or depression within the depression hierarchy. Capital letters label cells, and numbers on colored dots label depressions. Colors at the base of each panel match the colored dots and indicate to which depression each cell belongs. The algorithm consists of three major stages (Figure 5). From its initial distribution (a), water is moved downhill following flow directions in the steepest downslope direction from each cell, as indicated by the arrows. Water continues to move downslope until it reaches the pit cells (b, §3.1). Water is then moved within the depression hierarchy (c–f, §3.2). (c) shows the initial distribution of water within the depression hierarchy, based on how much water was in the pit cell of each depression. Water in depressions with insufficient volume overflow first into their sibling depressions and then – if the sibling depression becomes filled – passes to their parents. All of the leaf depressions in (c) are completely filled, so no sibling depressions can accommodate more water. Therefore, depressions 1 and 2 pass their overflowing water up to their parent, depression 6, and depressions 3 and 4 pass their overflowing water up to their parent, depression 5 (d). Depression 6 is now overflowing, but its sibling, depression 5, is not full, so depression 6 passes as much of its overflowing water as it can to depression 5 (e). Once depression 5 is full, some overflowing water still remains, so this is passed to the parent, depression 7 (f). In this case, depression 7 is able to accommodate the remainder of the water. Had depression 7 also overflowed, the leftover water would have overflowed into the ocean and been disregarded. Depressions to be flooded are then identified and flooded (§3.3). Since depression 7 contains water, we know that all of its descendants must be completely full. Therefore, we can flood these all at the same time, on the level of depression 7. Any one of the pit cells within depression 7 is arbitrarily selected as the starting point (g). More cells are added until all of the water has been accommodated. (h-j) are a visual representation of this process, although the algorithm would first locate affected cells C-J, and then calculate the final height of water in all of these cells in a single step.

60 **Comment:** P9ln10 – “depth/breadth-first traversal". Please explain what this is in plainer English.

Response: These terms are fundamental to the study of algorithms and would be taught in an intro level course, as such we think it’s alright to assume reader familiarity. We now state this directly at the beginning of §2.

> Many of the techniques in this paper are based on binary tree data structures and their traversals. Although we define terms below, more complete explanations and visual examples can be found in the text for any introductory undergraduate course on data structures. We recommend Skiena (2008) and Sedgewick and Wayne (2011) as good references. In particular, a good understanding of recursion will be helpful.

Further, we add to the end of §2:

> With these linkages in place, we can consider various ways of traversing the trees. Given a binary tree T with left and right children $T.L$ and $T.R$, a breadth-first traversal considers both $T.L$ and $T.R$ before considering any of $T.L.L$, $T.L.R$, $T.R.L$, or $T.R.R$. A depth-first traversal, on the other hand, will consider $T.L$ and all of its descendants before considering $T.R$ or any of its descendants. The tree traversals we perform in this paper are all depth-first.

We now note in the text that an invariant is “(a property which is true before and after each call a function)".

This page demonstrates a use of invariants to prove the correctness of insertion sort: http://www.cs.xu.edu/csci220/01f/insertionProof.html.

Comment: P9ln25 – Suddenly, the paper becomes much more jargon-y and hard to follow. Clarifying depth-first at ln 10 will help, but what is “post-order”? “an invariant”? Please explain these in plainer English. This paragraph and the next are also quite confusing aside from the jargon, so please expand and be more generous with the text as well.

Response: Again, this terminology is standard to introductory algorithms. However, to the end of §2 we add:

Depth-first traversals are most naturally expressed via recursion and come in three types: in-order, pre-order, and post-order. Let a recursive traversal function be called $r(\cdot)$ and the processing we perform on a particular node in the tree $p(.)$, then the traversals are given by:

- in-order: $r(T.L)$ then $p(T)$ then $r(T.R)$
- pre-order: $p(T)$ then $r(T.L)$ then $r(T.R)$
- post-order: $r(T.L)$ then $r(T.R)$ then $p(T)$

Comment: Things I am struggling with: How does a depth-first traversal have an "outer" rather than a "deeper"? Clearly the water starts in the leaves, so how can the outer (first??) traversal begin in the ocean (ln 30), which is a trunk? How are the traversals nested, and when do we shift from outer to inner and vice versa? An explanation of the jargon in a pre-paragraph will probably sort most of this – i.e., what are outer and inner traversals, what is pre- and post-order, and how are these concepts interrelated? Another flowchart, a diagram, or chunk of pseudocode would also really help here. The steps on P10 help piece together what these paragraphs must mean, but they come after the confusing text not with it, and the text should really stand on its own.

Response: To address this we've added a new section ("Traversals") more clearly defining the different traversals that can be used on binary trees.

We have also added and modified the language in this section:

> To effect the intuition developed above, we need a well-defined way to visit all of the nodes in the depression hierarchy. A post-order traversal allows us to visit both of a node's children and their descendants before calculating any quantities on the node itself. The result is that leaves get processed before their parents. However, a single traversal is insufficient: we need one traversal (the “outer" traversal) to identify nodes that have excess water and another traversal (the “inner traversal") to distribute this water. The outer traversal may launch the inner traversal many times as it works its way up hierarchy.
>
> To efficiently redistribute water, the Fill-Spill-Merge algorithm performs nested depth-first traversals of the DH. The outer traversal is post-order and considers each meta-depression in turn, from the most deeply nested to the least. For each meta-depression, an inner traversal handles its overflows by moving water to its sibling (starting by filling the sibling's descendants) and, if there's any left, passing it to the depression's parent. In this way, the outer traversal maintains an invariant (a property which is true before and after each call a function): any meta-depression it has processed does not contain an overflow. Put another way, the outer traversal finds problems and the inner traversal fixes them.

We have added pseudocode for the traversals and refer to it at appropriate points in the text.

Comment: P10ln10 – "We add water *from A* to B"? Where does the added water come from? Is it the original volume or the passed volume? (Must be the latter, but this is not as written) Tweak the phrasing here. P10ln11 – "this part of the algorithm". Do you mean point 1, the inner traversal, or the section 3.2? Point (iii) in this list is also very confusing. Break this case out and say what you mean (and why this is a terminating case) more clearly. The manuscript has not described the concept of an oceanlink-geolink pair (surely the link between two nodes is either an oceanlink or a geolink, so there can be no pairing?), so clarify that too (see main comment). Again, this feels like a diagram or pseudocode or both would add a lot here. The diagram I have in mind would show the possible cases for this 1-3 loop as a sequence of filling lakes, as in Fig 2 or fig 4a,b, g-j (but where water volume is honoured). Although I appreciate this section looks like this because it explicitly mirrors the structure of the code, there is surely a clearer way to express 1-3. E.g.– 1. If a node has excess water, attempt to pass that water to an unfilled sibling.– 2. If no unfilled sibling is available, pass the water to the parent and terminate inner loop.– 3. If no parent exists, the node must have an oceanlink, so pass the water along the oceanlink and terminate inner loop.– 4. If the oceanlink ends in the ocean, discard the excess water and terminate inner loop. Does this not encompass all the logic without the detail in the subcases, the pseudo-parents, etc.?

Response: We have tried to clarify the wording here as follows

When an overfilled depression is located by the outer traversal above, its water needs to be redistributed to neighbouring depressions. If we call the overfilled depression D, then the water can be redistributed by starting a second, inner post-order traversal at D. This inner traversal carries Excess Water from one depression to another until it has found a home for all of it. When we pass water into a depression, it can go to one of three places: the depression itself, its sibling, or its parent. Distributing the water to any of these places may itself cause an overflow. Therefore, the inner (pre-order) traversal comprises the following steps:

1. Call the depression that we are currently considering B. This may be the depression we originally considered, depression D, or it may be some other depression reached during the steps detailed below. If B is overflowing, we add the overflow to the Excess Water the inner traversal is carrying. If B has spare capacity we add water from the Excess to B until either it fills or all of the Excess Water the inner traversal is carrying is used.
2. At this point, the inner traversal can terminate if: (i) there is no water left, (ii) B is the parent of D, or (iii) B acts as a parent of A by receiving its overflow via an oceanlink.

...

We have added pseudocode for the traversals and refer to it at appropriate points in the text.

Comment: P10ln31 – apart from still disliking the "paired" terminology here, this also brings back the pt (iii) I didn't understand above. This makes it clear what you meant there, but the idea that 12 is sort of a parent of 4, but not really, and only sometimes, is a confusing one. I would stick with the idea that parents are strictly in the same tree, and replace this idea with simply "passing water down the oceanlink to move it between trees". I don't see what else it is adding for the cost of being confusing. In general though, this stuff on P11 is really good.

Response: As above, we've tried to better separate the concepts of geolink and oceanlink. We've also tried to simplify the language in this section to separate the concepts of parent and oceanlink:

1. Call the depression that we are currently considering B. This may be the depression we originally considered, depression D, or it may be some other depression reached during the steps detailed below. If B is overflowing, we add the overflow to the Excess Water the inner traversal is carrying. If B has spare capacity we add water from the Excess to B until either it fills or all of the Excess Water the inner traversal is carrying is used.
2. At this point, the inner traversal can terminate if: (i) there is no water left, (ii) B is the parent of D, or (iii) B was reached via an oceanlink.
3. Otherwise, if B has a sibling and the sibling's water volume is less than its depression volume, then start from Step 1 with the new B set as the depression pointed to by the current B's geolink.
4. Otherwise, if B has no sibling or the sibling's water volume is equal to its depression volume, then start from Step 1 with the new B set as the parent of the current B or, if B has no parent, then whatever depression B oceanlinks to.

Comment: P12lns3-4 – I don't see why the algorithm needs to start at the ocean and iterate back to every leaf, only to iterate forward again. Given we already maintain a list of all the leaves, associated pits, and labels, why can't me just start at the leaves and only go forward? If I'm missing something, it means this needs clarifying anyway. Are you trying to say, iterate from the ocean down along all the branches, but pinch out the branch whenever you find water and don't go further?

Response: As noted in §2, "a depression hierarchy (DH) is a data structure representing a forest of binary trees."

There are no lists of leaves or pits: this information is all organized into trees. The only way to find the leaves is by traversing the tree. The only way to move information around in the tree is via (recursive) depth-first traversals. Iteration, in the strict sense of the word, doesn't apply in this context.

It is possible we could design a different algorithm that somehow tracks leaves and moves from them towards the ocean via a breadth-first traversal, but this would likely be more difficult to describe, less intuitive, and certainly more difficult to implement.

Comment: P12lns15-28: Another place where a block of pseudocode alongside the text would add a lot. Arguably it could replace the text entirely, but having both would be extra clear.

Response: We have added pseudocode for this function as well and references to it in the text.

Comment: P12ln29. “To do so, we imagine a hypothetical outlet that drains the depression. . . " add: “at this elevation". This section for me says the same thing three times in different words. I would have simply: “Step 1 in this approach requires an efficient way to determine the volume of a depression below any given elevation. To do so, we imagine a hypothetical outlet that drains the depression at that elevation. If we call the elevation of this hypothetical outlet o and...” (also see comment below about maybe removing the idea of a hypothetical outlet entirely.)

Response: Thank you, we appreciate your wording suggestion here and below and have adopted the strategy you suggest. This section used to read:

> Step 1 in this approach requires an efficient way to determine the volume of a depression below any given elevation. To do so, we imagine a hypothetical outlet that drains the depression. If the depression is full enough that of its all cells receive water, then the elevation of this hypothetical outlet is simply that of the topographic outlet from the depression. If the depression is not yet completely filled, it can be visualized as a pipe in the side of the depression that is an infinite sink for any water entering it, thereby acting analogously to an overflow drain below the edge of a sink or bathtub. If we call the elevation of this hypothetical outlet o and a depression contains N cells of elevations $\{a, b, c, d, \ldots\}$, then the total volume per unit area (i.e., height) of water that the depression can accommodate is

It now reads:

> Step 1 in this approach requires an efficient way to determine the volume of a depression below any given elevation. If we call this elevation o and the depression below the outlet contains N cells with elevations $\{a, b, c, d, \ldots\}$, then the total volume per unit area (i.e., height) of water that the depression can accommodate is

Comment: Equation 1: You don't actually define N

Response: We have now defined N:

> If we call the elevation of this hypothetical outlet o and a depression contains N cells of elevations $\{a, b, c, d, \ldots\}$

Comment: Equation 3: I don't object to it, but why is there a bar through V?

Response: We have barred the bar from the paper.

Comment: P13ln4 – Run-on; end with a period not comma or tweak phrasing.

Response: This sentence previously read

> Now, consider cells $c_i = c_1, \ldots, c_N$ in the plain queue (i.e., those that have been visited and popped from the priority queue), we can calculate the volume of the depression below that of the last cell popped from the priority queue, the sill z_s, as: [EQU] Here, V_{dep,z_s} is the volume of the depression below z_s, and z_i is the elevation of cell c_i.

It now reads

> Now, consider cells $c_i = c_1, \ldots, c_N$ in the plain queue; that is, those cells that have been visited and popped from the priority queue. We can calculate the volume of water that can be accommodated in the depression below the elevation z_s of the last cell c_N (the sill) as: [EQU] where z_i is the elevation of cell c_i and a_i is the area of cell c_i.

Comment: P13ln5 – “the sill”. Which? The hypothetical one, or the real one? In general here, I think the concept of an “imaginary” outlet at every possible node height is getting in the way of what you’re trying to say. Isn’t it just that you can track the volume of a depression below various depths using the equations you give by cumulative summing along the plain queue, and then can compare that increasing volume as you move along the queue to the water volume available, which tells you the fill level? I would rephrase the way you present P12ln29–P13ln9 without this imaginary sill/outlet concept.

Response: We’ve eliminated the “imaginary" outlet in accordance with your suggestion, thank you. (See also, above.)

Comment: P14ln32 — If you rename the section, I don’t think this final sentence is needed.

Response: We feel the sentence contributes to the flow and so have opted to retain it, but we appreciate the suggestion.

Comment: P15ln11 – consider “Applications and testing on real-world data” or words to that effect (assuming you renamed 4 above, and remove the subheading below).

Response: We have renamed the section per your suggestion.

Comment: P13ln21 – I don't think this is the right heading for the section. Isn't this "Algorithm performance" or something like that more specific? 4 and 5 would then be subsections,e.g., "theory", "computational performance". The section itself reads really well though — and I assume by this point you will have defined the iteration jargon a bit better at first use.

P16ln1 – there is no 6.2!? Remove the subheading. Otherwise, I like this section a lot....overall, I think a light restructuring (i.e., broadly just some section renaming and text shunting) through 4-6 would make this a bit stronger. What you have is testing of performance on both synthetic (please!) and real-world data, then a real-world case-study that attempts some model validation. I think the sectioning should reflect that a bit more explicitly, e.g. something like,4. Model performance– 4.1 Theoretical analysis– 4.2 Performance on synthetic data 5. Field testing and inter-model comparison– 5.1 Performance on real world data(brief chat about Table 1 to give people an idea of general performance on real-world data, or just remove and junk these subsections)– 5.2 Model intercomparison (This is basically just 6.1 – and by all means, keep the performance data for this case study within this section)

Response: Thank you for this suggestion, we have organized these sections under "Algorithm performance" with subsections for "Theory", "Computational Performance", and "Model Intercomparison". We have also added performance tests on synthetic data.

Comment: (Technical) Install instructions in the readme require cmake – and so should probably explicitly direct the user to acquire this first. I note configuration issues are also likely, which should probably also be clarified (see below); if they are not, the authors are dramatically reducing the potential reach and impact of their work. In general, I think the readme needs expanding a bit (see technical comments).

Response: We make a note that cmake is required in `7929c96a3832e819c46e3812618a118681467852`. Configuration issues are always a possibility, but the use of cmake and the opportunity for users to raise configuration issues via Github should reduce this potential.

> **Comment:** (Technical) I severely struggled with the install for the software, and am probably significantly more tech-savvy than many target users. Did the authors test installs thoroughly against a full range of OSes? I am running macOS Catalina, XCode 11.4 (i.e., fully modern installs) and anaconda, and am seeing a spew of cmake errors which some investigation reveals are related to incompatibilities between the installer, the shipped SDKswith XCode, and anaconda. This is going to be a common user case, and I think the authors need to address it, if only briefly, in the docs – and test some other common environments too for safety's sake. [Part of the answer to my problem is here: https://www.anaconda.com/blog/utilizing-the-new-compilers-in-anaconda-distribution-5. The install process isn't compatible with the most up-to-date XCode, and the user needs to get the 10.9SDK off Github as described there, then point the compiler at it with "export CONDA_BUILD_SYSROOT=/Opt/MacOSX10.9.sdk" (assuming it's downloaded to Opt).] (I should note the above broadly duplicates an issue I have raised directly on Github.)

Response: We've tested the code on the range of systems we have access to and are responsive to issues raised on Github. The reviewer's particular issues have been addressed at:

– https://github.com/r-barnes/Barnes2020-FillSpillMerge/issues/3

– https://github.com/r-barnes/Barnes2020-FillSpillMerge/issues/5

We have also removed the NetCDF dependency.

> **Comment:** (Technical) The current install instructions appear to assume GDAL is installed on the target machine, and do not say so. They must make this explicit, as the triggered install errors are extremely opaque. I gather from asking on Github that these errors do not actually prevent install, but the readme needs to provide that guidance because the errors look pretty scary – and also should I think recommend a GDAL install anyway. [I got GDAL in a pain-free fashion using homebrew on my Mac (https://brew.sh), if that helps – or just let users figure it out for themselves.]

Response: GDAL is not required by either algorithm, but is used by the demonstration/example code. We now clarify this in the cmake files and README. Per the Issues you raised on Github (see above), we now provide Mac installation instructions in the README.

> **Comment:** (Technical) There should also be a line in the readme telling the user how to launch the software after install, as there is over at r-barnes/Barnes2019-DepressionHierarchy.

Response: The README now contains the following:

> A test program, `fsm.exe` is generated by make. This program simulates pouring a given amount of water onto every cell on a landscape and determining where it all ends up. Running the program shows its command-line arguments.

> **Comment:** (Technical) My familiarity was C++ testing is not very complete, but I don't believe this code has a *formal* testing framework attached to it. This is more an observation than anything else, but I would informally recommend that the authors consider building one at some point in the future. This is the way the wind seems to be blowing in research software development at the moment, and it might be good to get ahead of the pack on this. On the basis that the code is not formally tested, I have not attempted to extensively review the code itself. However, informal tests are applied in the manuscript and in the code (the correctness tests), and this for me meets the basic requirements for a submission like this. Could you add a line in the readme telling the user how to launch the correctness tests for themselves...? This would also serve as a nice example of the code doing its thing.

Response: For this revision, we have substantially expanded the tests accompanying our code. The README file explains how to compile and run these tests. We use *random testing* in conjunction with known reductions (such as depression filling) and multiple-paths analysis (such as progressively adding small amounts of water to a landscape versus adding a large amount all at once) to perform both unit tests as well as end-to-end tests on the code. As of `a350c410cafbb4caef4`, our test suite subjects our code to 208,490 distributed over 11 test cases; all assertions pass.

> **Comment:** (Technical) Code availability statement is great – but you should consider actually putting an open source licence onto the code and making that explicit here (after all, this is already defacto open source code, so you should probably formalise it). E.g. add a LICENSE.txt and add "The code is released under an MIT Open Source license [or your preferred license]" at the end of this section. But your call.

Response: Thank you for the suggestion, we have moved to the MIT license in `a28264bd337975e5fa4e335c03`.

3 Other:

While adding additional tests to the code in response to the reviewers' requests we discovered a bug in Fill-Spill-Merge that had caused some discrepancies in the results in our original submission. We have since corrected this bug, and reran the model to obtain correct results. The figures in the "Model Intercomparison" section have been changed to show these updated results; they also incorporate a new colour scheme based on the reviewers' requests.

Computing water flow through complex landscapes ~~,~~ – Part 3: Fill-Spill-Merge: Flow routing in depression hierarchies

Richard Barnes[1,2,3], Kerry L. Callaghan[4,5], and Andrew D. Wickert[4,5]

[1]Energy & Resources Group (ERG), University of California, Berkeley, USA
[2]Electrical Engineering & Computer Science, University of California, Berkeley, USA
[3]Berkeley Institute for Data Science (BIDS), University of California, Berkeley, USA
[4]Department of Earth & Environmental Sciences, University of Minnesota, Minneapolis, USA
[5]Saint Anthony Falls Laboratory, University of Minnesota, Minneapolis, USA

Correspondence: Richard Barnes (richard.barnes@berkeley.edu)

Abstract. Depressions—inwardly-draining regions—are common to many landscapes. When there is sufficient moisture, depressions take the form of lakes and wetlands; otherwise, they may be dry. Hydrological flow models used in geomorphology, hydrology, planetary science, soil and water conservation, and other fields often eliminate depressions through filling or breaching; however, this can produce unrealistic results. Models that retain depressions, on the other hand, are often undesirably expensive to run. In previous work we began to address this by developing a depression hierarchy data structure to capture the full topographic complexity of depressions in a region. Here, we extend this work by presenting a Fill-Spill-Merge algorithm that utilizes our depression hierarchy data structure to rapidly process and distribute runoff. Runoff fills depressions, which then overflow and spill into their neighbors. If both a depression and its neighbor fill, they merge. We provide a detailed explanation of the algorithm as well as results from two sample study areas. In these case studies, the algorithm runs 90–2,600× faster (with a 2,000–63,000× reduction in compute time) than the commonly-used Jacobi iteration and produces a more accurate output. Complete, well-commented, open-source code with 97% test coverage is available on Github and Zenodo.

1 Introduction

Depressions (see Lindsay (2015) for a typology) are inwardly-draining regions of a DEM that lack any outlet to an ocean or other designated base elevation. Depressions occur naturally, and can be formed by glacial erosion and/or deposition (Breckenridge and Johnson, 2009), compressional and/or extensional tectonics (Reheis, 1999; Hilley and Strecker, 2005), and cratering ~~Cabrol and Grin (1999)~~(Cabrol and Grin, 1999). They often host lakes and wetlands by retaining water locally. Depressions may themselves contain depressions. Such regions confound algorithms for geomorphological and terrain analysis, as well as those for hydrological modeling, because many such algorithms simply route water down topographic slope following the local gradient: depressions neither fill with water, nor drain.

Many hydrological models deal with the complexity of depressions by removing them. This can be done either by filling the depressions with earth so that they form a flat region of landscape (e.g. Jenson and Domingue (1988); Martz and Jong (1988)); breaching (Martz and Garbrecht, 1998) or carving them (Soille et al., 2003) so that water flows from their lowest point through the carved channel and onward to downstream regions; or some combination of these (Lindsay and Creed, 2005b; Schwanghart and Scherler, 2017; Soille, 2004; Lindsay, 2016). This approach is justified for situations in which spatiotemporal aspects of the analysis allow depressions to be ignored or for cases in which all depressions can be considered to be data errors (Lindsay and Creed, 2005a). Historically, many DEMs were constructed from sparse data, and small data errors produced depressions, especially in flat areas (O'Callaghan and Mark, 1984). Such an assumption is no longer justified, as improved and increasingly high-resolution data have become available (Li et al., 2011). Even coarse-resolution data are capable of resolving real-world depressions (e.g. Riddick et al., 2018; Wickert, 2016). With this in mind, new approaches are beginning to be examined, particularly in post-glacial landscapes where depressions have a significant impact on local hydrology ~~(e.g. Lai and Anders (2018))~~ (e.g., Lai and Anders, 2018) and therefore cannot be ignored during modeling.

FlowFill (Callaghan and Wickert, 2019) began to combat this problem by routing water across landscapes in a way that conserved water volume, creating flow-routing surfaces that could still contain real depressions. Under reasonable runoff conditions, their results show landscapes that still contain depressions and disrupted flow routes. The FlowFill method iteratively routes water from higher to lower terrain. As depressions fill, they pose an extreme challenge to such a method: since water seeks a level surface, the surface of a filled depression must eventually become flat and any fluid flowing onto the surface diffuses across it. Even for moderately-sized surfaces it can take many iterations for a solver to reach steady state; we provide a theoretical analysis of this in Section 4.1. Runtimes for FlowFill ranged from seconds to days: large datasets quickly became unwieldy. Of those examples tested by Callaghan and Wickert (2019), the slowest was a dataset of 4,176,000 cells which took approximately 33 hours for FlowFill to process. In contrast, the Fill-Spill-Merge algorithm presented here fills a similarly-sized dataset in 8.7 s.

Other authors have considered the problems of extracting nested depression hierarchies and dynamically routing water through them. However, these previous approaches are either slow, inexact, or both; additionally, most previous efforts were not accompanied by source code, limiting their utility. Barnes et al. (2020) provide a more thorough literature review which we briefly recap here. A hierarhical segmentation by Beucher (1994) did not produce a data structure on which flow could be routed. Salembier and Pardas (1994) generated a hierarchical segmentation by repeatedly simplifying source images; hydrologically speaking, this can lead to unacceptable degradation of terrain information. Arnold (2010) developed an algorithm similar to the one here, but without source code; the algorithm also generates looping topologies that require correction. Wu et al. (2015) and Wu and Lane (2016) constructed depression hierarchies by first smoothing a DEM and then extracting vector contour lines from it. Wu et al. (2018) build on this approach by discretizing the DEM into a number of horizontal slices. Both approaches sacrifice exactness and the latter requires $O(N^2)$ time. Cordonnier et al. (2018) use planar graph minimum spanning trees to construct a hierarchy of depressions, but do not produce a data structure water can be routed on. In contrast, the Fill-Spill-Merge algorithm relies on a well-defined data structure (Barnes et al., 2020); has complete, well-commented source code with extensive correctness tests (Barnes and Callaghan, 2019, 2020); has strong efficiency guarantees (§4.1) which are realized on actual and simulated datasets (§4.1); and provides exact answers.

To achieve this, we developed a data structure—the *depression hierarchy*—which represents the topologic and geographic structure of depressions. In an accompanying paper, we provide details concerning ~~the depression hierarchy and its construction~~how

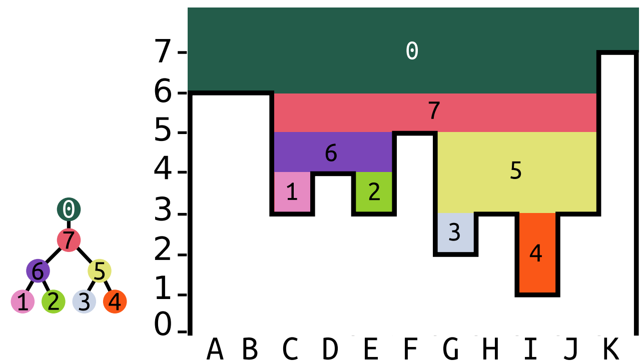

Figure 1. A single subtree of a depression hierarchy and the depression it represents. Depressions 1–4 are leaf depressions. Depression 6 is a parent depression (also termed a meta-depression) that contains depressions 1 and 2. Water from the plateau on the left above cells A and B might *fill* Depression 1 (cell C), causing it to *spill* into Depression 2 (cell E). Only when both depressions are full do they *merge* and begin filling Depression 6 (cells C, D, and E). Modified from Barnes et al. (2020).

a depression hierarchy is constructed (Barnes et al., 2020). In this paper, we explain how ~~the~~ a depression hierarchy can be leveraged to accelerate hydrological models using a paradigm we call *Fill-Spill-Merge*.

2 Using The Depression Hierarchy

Many of the techniques in this paper are based on binary tree data structures and their traversals. Although we define terms below, more complete explanations and visual examples can be found in the text for any introductory undergraduate course on data structures. We recommend Skiena (2008) and Sedgewick and Wayne (2011) as good references. In particular, a good understanding of recursion will be helpful.

2.1 Terminology

Depressions can themselves contain depressions, as shown in Figure 1. A depression hierarchy (DH) is a data structure representing a forest of binary trees, as shown in Figure 2a, that represents the relationships between depressions (Figure 2a–d). Each node in the DH represents a depression. Nodes higher in the DH are depressions that themselves contain depressions; we term these *meta-depressions*. Although the depression hierarchy could be generalized to n-ary trees using multiple flow direction routing, the binary simplification is sufficient to cover most use cases. A node in the DH can have several classifications:

- **Parent**: A node, such as #10 and #12 in Figure 2a, that represents a meta-depression, and whose topological descendants therefore also form depressions.
- **Child**: A depression, such as both #10 and #1 in Figure 2a, that geographically and topologically exists within the meta-depression formed by its parent.

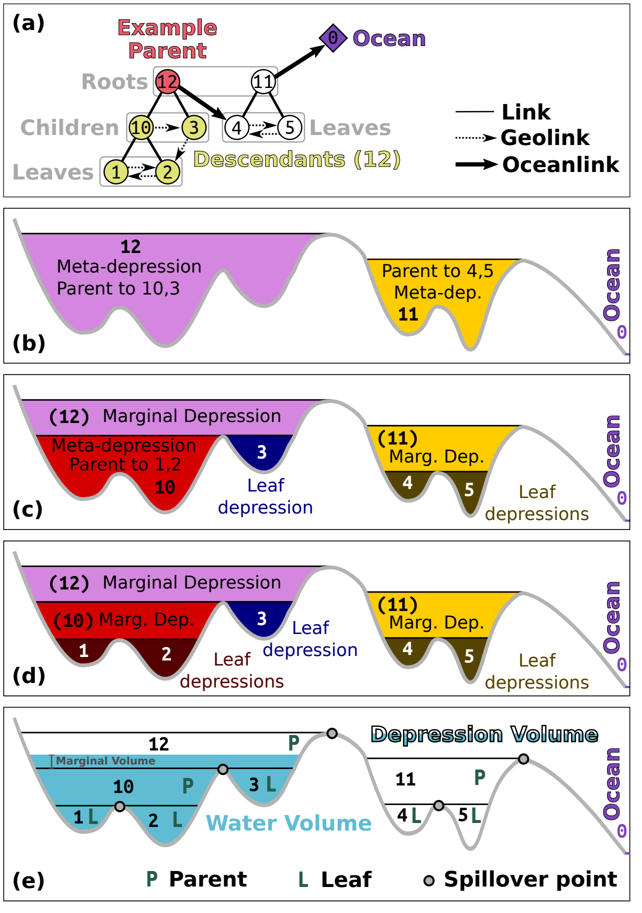

Figure 2. Terminology for the depression hierarchy and water flow through it. The depression hierarchy shown here is drawn from the left hand side of Figure 1 from the companion paper by Barnes et al. (2020). **(a)** Topology. A *parent* and its *descendants* are associated with depressions (b–d). Direct descendants are called *children*. *Leaves* are the terminal members of the depression hierarchy; they have no children and represent simple depressions (i.e., those that are not meta-depressions). Members of a single *binary tree* are joined in their hierarchy through *links*; directional links that represent water-spillover directions between geospatially adjacent depressions are called *geolinks*. Flow from one binary tree into another and towards the ocean follows the *oceanlinks*. Though only one binary tree is shown, the ocean may be the parent to an arbitrarily large *forest* of binary trees. **(b)** Parents in the hierarchy form *meta-depressions* — depressions that encompass other depressions. **(c)** These meta-depressions contain *leaf depressions* — depressions that themselves contain no depressions. These are associated with leaves in the depression hierarchy. Meta-depression 12 also contains another meta-depression, 10. The regions of Depressions 11 and 12 that lie above their child depressions are termed "marginal depressions". **(d)** Meta-depression 10 contains leaf depressions 1 and 2. **(e)** ~~Water flow in~~ Using the depression hierarchy to simulate water flow. Water first fills *leaf depressions* before flooding into neighboring *depressions*. Once a depression and its neighbor are completely filled, their *parent* begins to flood. The *depression volume* is the full geometric volume of the depression. The *water volume*, ~~nautrally~~naturally, is the volume of water within a given depression. The *marginal volume* is the volume of water partially filling the top-level meta-depression; appropriately spreading this water across the landscape is the topic of Section 3.3.

- **Leaf**: A depression, such as #1 and #2 in Figure 2a and Figure 2d, that has no children. The leaves of the binary trees represent the smallest, most deeply-nested depressions. If a landscape were initially devoid of water, then water flowing down slopes would begin to collect in some subset of these leaf depressions before it would begin to fill their parent depressions.

- **Root**: A depression, such as #0, #11, and #12 in Figure 2, that has no parent. This term may also refer to any node that is used as the starting point for a traversal that only considers the node and its descendants.

- **Descendant**: A child of a given parent, or the child of a child of that parent, and so on. In Figure 2a, #1, #2, #3, and #10 are all descendants of #12.

- **Sibling**: Every node has either no children (leaf nodes) or two children. Nodes which share a parent are siblings. In Figure 2a, #1 and #2 are siblings, as are #4 and #5.

As depressions fill, their water surfaces eventually reach a *spill elevation* (Figure 2e) at which they overflow into neighboring depressions. During this spilling, water flows from a depression into a geographically neighboring leaf depression, topologically connected by a *geolink*. The spill elevations in Figure 1 are the highest points of each band of color.

Each node in the DH is associated with several properties:

- **Depression volume**: This is the *total* volume of water that the depression, including all of its descendants, can contain before spilling over.

- **Water volume**: This is the *total* volume of water *actually being stored* in the depression. A parent depression will have a non-zero water volume only if ~~all~~ both of its children are completely full and the parent itself contains some additional volume of water. In this case, the water volume will be the sum of the water volumes of the children and the additional margin of water contained within the parent (i.e., the "marginal volume" indicated on Figure 2e). Parents whose children are not ~~all~~ both filled with water will have a water volume equal to zero. In this way, we can use this property to determine which portions of the DH are fully or partially filled, and which are the highest water-containing nodes in any of the binary trees.

- **Geolink**: When a depression spills, its water passes into the subtree rooted by its sibling. However, in a full model of flow, the water would move downslope from the spill cell into whichever leaf depression of the sibling is geographically proximal to the spill cell. *Geolinks* are pointers from depressions higher in the DH to the leaf depressions that receive their water if they overflow. These are the dashed lines shown in Figure 2a. Geolinks are similar to the connections used in a threaded binary tree (Fenner and Loizou, 1984).

- **~~Ocean link~~Oceanlink**: Depressions high in the mountains may overflow down escarpments to depressions far below. In this case, the depressions do not overflow into each other: the relationship is one-way. There can be multiple such escarpments, so this can happen multiple times. In such cases, each group of depressions forms a proper binary tree.

However, the root of one of the trees has ~~both an~~ an ~~*ocean link*~~*oceanlink* ~~and a geolink~~ to a leaf node of the downstream binary tree. In Figure 2, both #11 and #12 are the root nodes of a set of nested depressions. #12 has an ~~ocean link~~ oceanlink (heavy arrow) to #4, one of the leaf depressions of #11. #~~12 also has a geolink (dotted arrow) to #4. #~~11 itself has an ~~ocean link and a geolink~~ oceanlink to the ocean. In many of the algorithms discussed below, ~~ocean-linked~~ oceanlinked nodes are processed similarly to children~~; however, information is usually not passed across ocean links. Oceanlinks are used solely for guiding traversals of~~ .

Within the algorithm, oceanlinks and geolinks are used for different purposes: an oceanlink tells us that the depression in question has grafted onto the leaf node of another tree of the depression hierarchy~~whereas water is passed through geolinks.~~ , locating a route for overflowing water to eventually reach the ocean. The depression to which it is oceanlinked is considered its parent, but it is not the child of that depression because water flows only one way along an oceanlink. In Figure 2a, depression #4 can be considered the parent of #12, but #12 is not the child of #4. This is because #12 is not physically contained within #4, but #12 will send all of its overflowing water to #4, as shown in Figure 2b–e. #4 will not contain the total water volume contained within #12, unlike other parents. Geolinks route water within geographically adjacent depressions contained in the same meta-depression.

2.2 Traversals

With these linkages in place, we can consider various ways of traversing the trees. Given a binary tree T with left and right children $T.L$ and $T.R$, a breadth-first traversal considers both $T.L$ and $T.R$ before considering any of $T.L.L$, $T.L.R$, $T.R.L$, or $T.R.R$. A depth-first traversal, on the other hand, will consider $T.L$ and all of its descendants before considering $T.R$ or any of its descendants. The tree traversals we perform in this paper are all depth-first.

Depth-first traversals are most naturally expressed via recursion and come in three types: in-order, pre-order, and post-order. Let a recursive traversal function be called $r(\cdot)$ and the processing we perform on a particular node in the tree $p(\cdot)$, then the traversals are given by:

- in-order: $r(T.L)$ then $p(T)$ then $r(T.R)$
- pre-order: $p(T)$ then $r(T.L)$ then $r(T.R)$
- post-order: $r(T.L)$ then $r(T.R)$ then $p(T)$

3 The Algorithm

The Fill-Spill-Merge algorithm consists of several steps, outlined here, depicted in Figures 3 and 4, and shown in flowchart form in Figure 5. ~~First (~~This paper is also accompanied by complete, well-commented source code; the reader may find it helpful to download this code and refer to it as an additional reference. First (§3.1), surface water needs to move downhill, either to the ocean (i.e., a designated sink region or the map edge) or to collect in pit cells – the deepest points within leaf

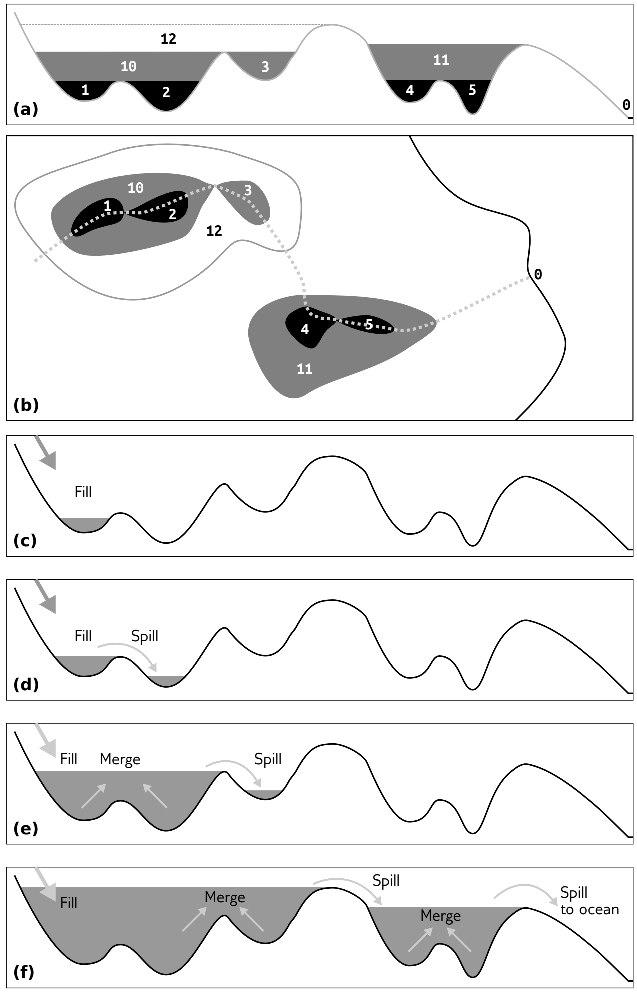

Figure 3. Fill-Spill-Merge process. Water moves through topographic depressions by filling them, spilling over sills, and merging to form meta-depressions. **(a)** Topographic cross section with labeled leaf depressions and their parents, following the left-hand side of the depression hierarchy in Figure 2. “0” represents the ocean; other numbers represent leaves and parents that together form depressions and meta-depressions. **(b)** Map showing this depression structure; the cross-section in (a) follows the dotted gray line. **(c)** A water source to the left begins to fill Depression 1. **(d)** Continued water input causes Depression 1 to overflow and spill into Depression 2. **(e)** Depression 2 fills, causing Depressions 1 and 2 to fill their parent (10) and merge to form a metadepression. This metadepression overflows into Depression 3. **(f)** Depression 3 fills and merges with Meta-Depression 10 (1 and 2 being implied members based on their position in the hierarchy) to flood their parent, 12. After Meta-Depression 12 overspills, it enters Depression 4, which then fills and spills into Depression 5. After Depression 5 floods, its waters join with those from Depression 4 to fill Meta-Depression 11, which then spills to the ocean. Figures 4 and 5 describe the algorithm in more specific detail.

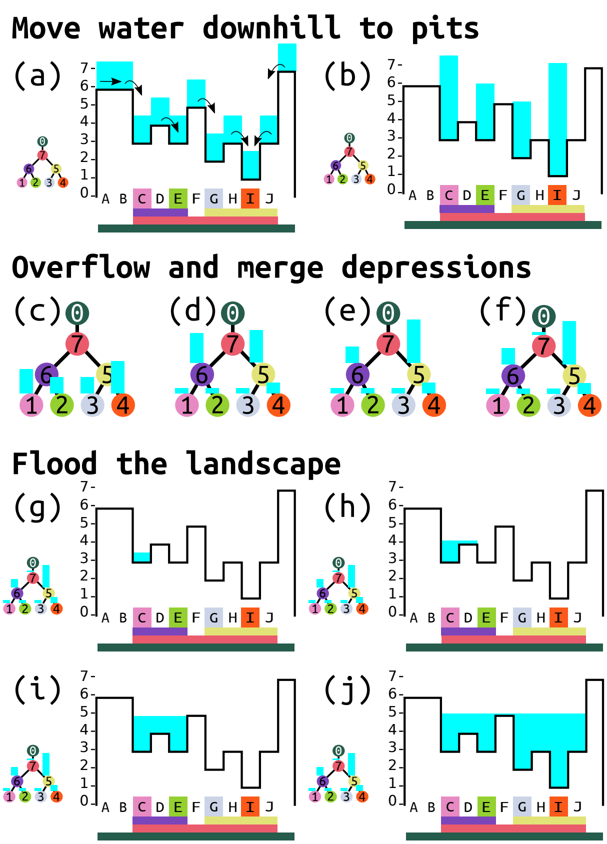

Figure 4. Visual Overview of the Algorithm. ~~In this figure~~ Black outlines represent the ~~heights~~ elevations of the ~~water bars~~ cells. Blue areas are ~~non-additive: only~~ the ~~changes between panels are important~~heights of water in each cell or depression within the depression hierarchy. Capital letters label cells, and numbers on colored dots label depressions. Colors at the base of each panel match the colored dots and indicate to which depression each cell belongs.

The algorithm consists of three major stages (Figure 5). From its initial distribution ~~(A)~~**(a)**, water is moved downhill ~~into~~ following flow directions in the steepest downslope direction from each cell, as indicated by the arrows. Water continues to move downslope until it reaches the pit cells (~~B~~**b**, §3.1). Water is then moved within the depression hierarchy (~~C–F~~**c–f**, §3.2)~~:~~. **(c)** shows the initial distribution of water within the depression hierarchy, based on how much water was in the pit cell of each depression. Water in depressions with insufficient volume ~~overflows~~ overflow first into their sibling depressions ~~(D)~~ and then – if the sibling depression becomes filled – passes to their parents. All of the leaf depressions in (~~E, F~~c) are completely filled, so no sibling depressions can accommodate more water. ~~Any~~ Therefore, depressions 1 and 2 pass their overflowing water up to their parent, depression 6, and depressions 3 and 4 pass their overflowing water up to their parent, depression 5. **(d)** Depression 6 is now overflowing, but its sibling, depression 5, is not full, so depression 6 passes as much of its overflowing water as it can to depression 5. **(e)** Once depression 5 is full, some overflowing water still remains, so this is passed to the parent, depression 7. **(f)** In this case, depression 7 is able to accommodate the remainder of the water. Had depression 7 also overflowed, the leftover water ~~overflows~~ would have overflowed into the ocean ~~(F)~~ and ~~is forgotten~~been disregarded. Depressions to be flooded are then identified and flooded (§3.3)~~starting from an arbitrarily-chosen~~. Since depression 7 contains water, we know that all of its descendants must be completely full. Therefore, we can flood these all at the same time, on the level of depression 7. Any one of the pit ~~cell (G–J)~~cells within depression 7 is arbitrarily selected as the starting point **(g)**. More cells are added until all of the water has been accommodated. **(h–j)** are a visual representation of this process, although the algorithm would first locate affected cells C–J, and then calculate the final height of water in all of these cells in a single step.

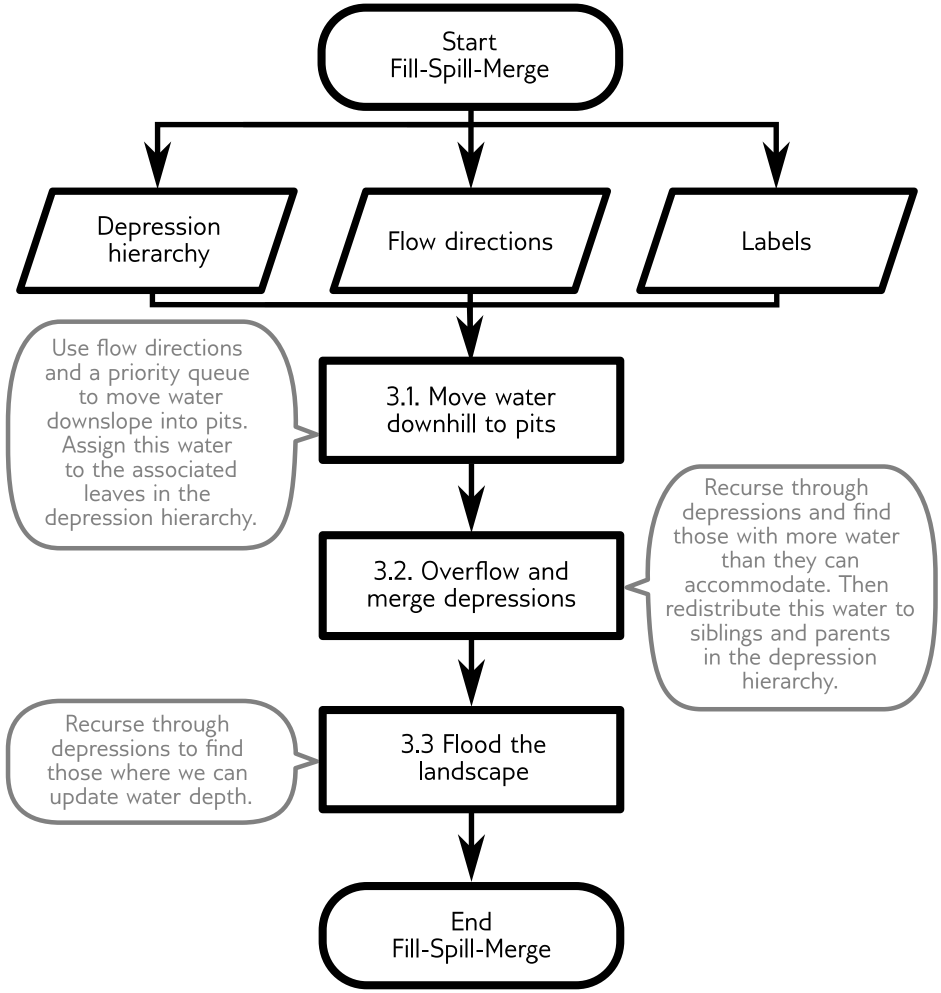

Figure 5. Flowchart showing the main steps taken by the algorithm. These steps are described in more detail in §3.1 to §3.3.

depressions. Note that the landscape may already have standing water at this stage. This operation takes place across all the cells of the DEM. Second (§3.2), water is redistributed across the depression hierarchy such that any depressions that have filled sufficiently ~~must~~ spill over into neighboring depressions and, if both depressions are full, flood their parent to merge into a single, larger body of water within a meta-depression. This operation is done without explicitly considering the cells of the DEM, which makes it very fast. Third and finally (§3.3), the water within the depression hierarchy is translated into an extent and depth of flooding across the topographic surface (DEM).

Computing a depression hierarchy (Barnes et al., 2020) is a necessary precursor to running Fill-Spill-Merge. The specific outputs from the depression hierarchy algorithm that are used in the Fill-Spill-Merge algorithm are:

– *DH*: the depression hierarchy itself.

- *Flowdirs*: a matrix of flow directions, indicating which of a cell's neighbors receives its flow. Because Priority-Flood (Barnes et al., 2014) is used to generate the depression hierarchy, flat areas are automatically resolved.

- *Labels*: a matrix indicating the leaf depression to which each cell belongs.

By routing water according to the DH, we significantly accelerate the compute speed and ensure that the full network of depressions is a topologically correct directed tree. Each of the following subsections details one of the numbered steps along the central path of the flowchart shown in Figure 5.

3.1 Move Water Downhill to Pits

We route water in a similar way to standard flow-accumulation algorithms (Mark, 1988; Wallis et al., 2009; Barnes, 2017), but for completeness summarize our approach here. Flow directions for each cell have already been identified by the ~~DH~~depression hierarchy algorithm. Each cell calculates how many of its neighbors flow into it. We call this value the cell's dependency count, as it describes the number of immediate upstream cells whose flow accumulation must be resolved before flow accumulation at the given cell can be computed. Local maxima in the DEM are identified as those cells that receive no flow from any neighbor. These local maxima are placed in a queue. Cells are then popped (i.e., noted while being removed) from this queue. The cells determine how much flow they generate locally (perhaps referring to a matrix of rainfall values, but also including existing stores of standing water) and add this to their flow accumulation value. They then add their flow accumulation to their downstream neighbor's and set their own flow accumulation value to zero. The neighbor's dependency count is then decremented. If the neighbor's dependency count has reached zero during this step, it is added to the end of the queue. This process of accumulating flow, passing it downstream, decrementing the dependency count, and adding cells to the queue continues until the queue is empty, at which point every cell on the map has been visited and any water has been moved downslope. Braun and Willett (2013) present an alternative formulation based on a depth-first traversal, but Barnes (2019) demonstrates that a breadth-first ordering, such as that presented here, is better suited to parallelism.

When the accumulated flow reaches the pit cell of a depression, the downhill-directed flow routing stops because there is no downhill neighbor to receive the flow. At this point, all of the flow-accumulated water in the pit cell is moved into the pit cell's associated leaf depression in the DH. That is, the water is moved out of the geographic space and into the topologic space. This then enables mass-conserving depression flooding via rapid Fill-Spill-Merge calculations, as detailed below.

3.2 Overflow and Merge Depressions

At this point, the Fill-Spill-Merge algorithm has routed all of the surface water into either the ocean or into the leaf nodes of the DH. The next step is to redistribute this water through the DH to nodes with enough volume to contain the water, and to send any excess water to the ocean. This set of operations can be performed entirely in the depression hierarchy without reference to the digital elevation model.

Intuitively, the process of filling, spilling, and merging can be visualized as occurring from leaf nodes to their parents (Figure 3). ~~Water must be redistributed such that leaf depressions containing~~ When a leaf depression initially contains more water than ~~they can holdspill over into their~~ it can hold, the water will be redistributed by spilling over into the neighboring depression. If this neighboring depression is already full, then the excess water must pass to the parent of both the depression and its neighbor. This process continues recursively until either the supplied water is exhausted or this water reaches the ultimate parent, the ocean. In this latter case, all excess water is dropped from the model and the ocean is unaffected.

To effect the intuition developed above, we need a well-defined way to visit all of the nodes in the depression hierarchy. A post-order traversal allows us to visit both of a node's children and their descendants before calculating any quantities on the node itself. The result is that leaves get processed before their parents. However, a single traversal is insufficient: we need one traversal (the "outer" traversal) to identify nodes that have excess water and another traversal (the "inner traversal") to distribute this water. The outer traversal may launch the inner traversal many times as it works its way up hierarchy. Pseudocode showing these travels is available in §6.1 and §6.2.

To efficiently redistribute water, the Fill-Spill-Merge algorithm performs nested depth-first traversals of the DH. The outer traversal (§6.1) is post-order and considers each meta-depression in turn, from the most deeply nested to the least. For each meta-depression, an inner traversal (§6.2) handles its overflows by moving water to its sibling (starting by filling the sibling's descendants) and, if there's any left, passing it to the depression's parent. In this way, the outer traversal maintains an invariant (a property which is true before and after each call a function): any meta-depression it has processed does not contain an overflow. Put another way, the outer traversal finds problems and the inner traversal fixes them.

The outer traversal of the DH (which is, after all, a forest of binary trees) begins with the ocean. For each depression, the algorithm first recurses into ~~the depression's left child and then into its~~ its oceanlinks, if any, and then into the left and then right child. ~~If any oceanlinks are found, the algorithm also recurses into them.~~ In the post-order portion of the traversal (which starts from the leaves and moves back up through the depression hierarchy), the algorithm identifies any depressions containing more water than they can accommodate. This process continues until the recursion returns to the ocean, at which point any additional water is assumed to be added to the ocean without impacting sea level, though this total discharge to the sea is recorded within the "ocean" depression.

When an overfilled depression is located ~~, the inner traversal redistributes this water. Let us call this overfilled depression A and note that it contains some amount of excess water — that is, water beyond its depression capacity. Our goal is to distribute this fixed amount of excess into~~ by the outer traversal above, its water needs to be redistributed to neighbouring depressions. ~~At each step below, the amount of this excess water remaining to be distributed will either remain the same or decrease.~~ If we call the overfilled depression D, then the water can be redistributed by starting a second, inner post-order traversal at D. This inner traversal carries Excess Water from one depression to another until it has found a home for all of it. When we pass water into a depression, it can go to one of three places: the depression itself, its sibling, or its parent. Distributing the water to any of these places may itself cause an overflow. Therefore, the inner (pre-order) traversal comprises the following steps:

1. Call the depression that we are currently considering B. This may be the depression we originally considered, depression ~~A~~D, or it may be some other depression reached during the steps detailed below. ~~We add water~~ If B is overflowing, we add the overflow to the Excess Water the inner traversal is carrying. If B has spare capacity we add water from the Excess to B until either it fills or all of the ~~water is used.~~ Excess Water the inner traversal is carrying is used.

2. At this point, ~~this part of the algorithm~~ the inner traversal can terminate if: (i) there is no water left, (ii) B is the parent of ~~A~~D, or (iii) B ~~acts as a parent of A by receiving its overflow via an oceanlink–geolink pair while not being a sibling or descendant, or (iv) B is the ocean~~was reached via an oceanlink.

3. Otherwise, if B has a sibling and the sibling's water volume is less than its depression volume, then start from Step 1 with the new B set as the depression pointed to by the current B's geolink.

4. Otherwise, if B has no sibling or the sibling's water volume is equal to its depression volume, then start from Step 1 with the new B set as the parent of the current B ~~. (Note that the parent may be the ocean ora node reached via an oceanlink).~~ or, if B has no parent, then use the depression to which B oceanlinks.

~~During each such pass through the inner traversal, water moves at most one step in the DH towards the ocean.~~

The next step of the outer traversal, which begins one level in the DH closer to the ocean, identifies a less nested metadepression for which the inner traversal might need to be run. If this step were not supplied with information about prior water redistribution, it could cause ~~multiple traversals of a subtree of the DH~~the inner traversal to cover the same nodes repeatedly, which would be computationally wasteful. To prevent this, the inner traversal returns the ID of the final node in which it placed water: this node is the only node in the traversal with spare capacity so future traversals can begin there. Therefore, on subsequent overflows, if such a cached value is available, then the recursion skips directly to that node. This ensures that all the calls to this part of the algorithm take no more than $O(N)$ time collectively.

The following examples uses the geometry from Figure 2 to describe a set of inner traversals, starting with an overflowing Depression #12. Step numbers mirror those above; numbers in parentheses indicate the number of recursions – that is, the number of times that the inner-traversal algorithm has returned to Step 1:

1 Depression #12 fills and overflows.

2 Depression #12's water overflows into Depression #4, which is not full, following its geolink.

  1(r1) Depression #4 acts as Depression #12's parent via ~~a geolink–oceanlink pair~~an oceanlink. The inner traversal terminates.

At this point, the outer traversal moves one level closer to the ocean, and the inner traversal repeats, this time starting at Depression #4.

1 Depression #4 fills and overflows.

2 Depression #4's water overflows into its sibling, Depression #5, which is not full and is a leaf depression. If Depression #5 had descendants, water overflowing from Depression #4 would have followed a geolink to one of these.

  1(r1) Depression #5s fills and overflows.

  2(r1) Depression #4 is full.

3(r1) Depression #5 overflows into its parent, Depression #11.

1(r2) Depression #11 overflows into the ocean; the inner traversal terminates.

Now the outer traversal moves yet another level closer to the ocean, and the new inner traversal starts at Depression #11.

1 Depression #11 fills and overflows.

2 Depression #11 has no sibling.

3 Depression #11 overflows into its parent, the ocean; all remaining excess water is absorbed into an infinite sink.

1(r1) The now-selected node is the ocean; the inner traversal terminates.

At this point, the outer traversal moves one level closer to the ocean, and arrives at the ocean. The outer traversal also terminates.

3.3 Flood the landscape

After water moves through the DH (Section 3.2, above), each node in the DH exists in one of the three following states:

1. **Empty:** The depression's water volume is equal to zero. In this case, nothing needs to be done. The depression's descendants might contain water, but the water never propagates to this level of the DH.
2. **Full:** The depression's water volume is equal to the volume of the depression itself. In this case, the depression is entirely full. If the depression's parent contains water, then the calculation of water depth is dealt with at a higher stage in the DH. If the depression's parent is empty, or if the depression's parent is the ocean, then the calculation is performed at this level as described below.
3. **Partially filled:** The depression's water volume is less than its depression volume. In this case, the depth of water across the depression and all its descendants' cells must be calculated at this level so that the depression fills to an appropriate level. This is described below and indicated as the *marginal volume* on Figure 2e.

The next step is to distribute this water across the DEM, appropriately flooding geographic depressions.

Given the three states described above, the algorithm locates the highest-level ~~node within each binary tree that contains~~ nodes which contain water. It does so ~~by first traversing from the ocean to each leaf depression by recursively traveling to each node's children in turn~~via a post-order traversal. Each time ~~it~~ the traversal reaches a leaf, the algorithm notes its label and pit cell. After identifying each of these, the algorithm reverses direction, moving from child to parent so long as the parent node contains water. ~~Therefore, this traversal towards the ocean ends at the highest-level node whose parent does not contain water. Call this node~~ Call the highest water-bearing node within a tree L. ~~The~~

In many cases, the water volume contained within the depression will ~~only very rarely be exactly enough to perfectly flood it~~be less than the total depression volume; therefore, we must ~~spread water across the depression to create a flat water surface.~~

~~To calculate water level within a depression, the algorithm begins by picking~~ calculate what the water level in the depression will be. To do this, we pick an arbitrary pit cell within ~~it~~L and its descendants, and then ~~uses~~ use this as a seed from which to start building a priority queue through the which will traverse the cells of the depression. The priority queue returns cells ordered from lowest to highest elevation. At each step through the priority queue, the algorithm checks whether a depression whose outlet is at this elevation would the cells visited so far collectively have enough volume to hold the water. If so, the algorithm exits, having successfully defined the flooded area. If not, it continues to build use the priority queue . to traverse the depression cell by cell. The filling procedure is shown in pseudocode in §6.3.

To expand this brief conceptual discussion into a more formal set of steps, let us begin by calling the active cell – that is, the one that is currently being considered by the algorithm – c_p. This cell is initially the arbitrary pit mentioned above, and is added to the priority queue. The algorithm marks c_p, which stands for "**c**ell of current highest **p**riority", as *visited*; all other cells remain unvisited. The algorithm then follows these steps:

1. Pop c_p from the priority queue, call it c, and use its elevation to calculate the volume of water that can be accommodated in the set of cells processed so far (Equation 3, below). If this volume is enough to accommodate the volume of water available, exit the loop and compute the final water level (Equation 6, below). Otherwise, proceed to Step 2.

2. Add the former c_p c (which was popped in Step 1) to a plain queue, which records all of the cells scanned so far; these cells will later be inundated.

3. Add the cells neighboring the former c_p c that are not marked as *visited* to one of two lists. If an unvisited neighboring cell shares a label with L or any of its descendants, then this neighboring cell is added to the priority queuethe priority queue if they belong to one of the descendant depressions of the one being filled. Each of these neighboring cells is then marked as *visited*.

4. Choose the lowest-elevation cell in the priority queue and label it as the new c_p and return to Step 1. If the priority queue is empty, then all cells in the same meta-depression as c_p or its descendants have been visited and we are now guaranteed to have sufficient depression volume to hold all of the water.

Step 1 in this approach requires an efficient way to determine the volume of a depression below any given elevation. To do so, we imagine a hypothetical outlet that drains the depression . If the depression is full enough that of its all cells receive water, then the elevation of this hypothetical outlet is simply that of the topographic outlet from the depression . If the depression is not yet completely filled, it can be visualized as a pipe in the side of the depression that is an infinite sink for any water entering it, thereby acting analogously to an overflow drain below the edge of a sink or bathtub. If we call the elevation of this hypothetical outlet is o and a depression contains cellsof elevations $\{a, b, c, d, \ldots\}$, then the capacity of the depression is If we call this elevation z_o and the depression below the outlet contains N cells with elevations $\{z_1, z_2, z_3, z_4, \ldots\}$ and unit cell area, the volume of water that the depression can accommodate simply equals the sum of the depth of water in each of its cells:

$$(o - a z_o - z_1) + (o - b z_o - z_2) + (o - c z_o - z_3) + (o - d z_o - z_4) + \ldots = No - a - b - c - d - No - z_1 - z_2 - z_3 - z_4 - \ldots \quad (1)$$

$$= No - \sum{}^{N} (\text{elevations})_{i=1}^{N} z_i \quad (2)$$

Now, consider cells $c_i = c_1, \ldots, c_N$ in the plain queue~~(i.e., those~~ ; that is, those cells that have been visited and popped from the priority queue~~), we~~ . We can calculate the volume of water that can be accommodated in the depression below ~~that~~ the elevation z_s of the last cell ~~popped from the priority queue, the sill z_s, as:~~ c_N (the sill) as:

$$V_{dep,z_s} = z_{\mathrm{s}} N_s \sum_{i=1}^{N} a_i - \sum_{i=1}^{N} z_i a_i \tag{3}$$

~~Here, $V_{\mathrm{dep,z_s}}$ is the volume of the depression below z_s, and~~ where z_i is the ~~elevation~~ elevation of cell c_i and a_i is the area of cell c_i. Thus, if we keep ~~track of the number of cells in a depressionand their total elevation~~running sums while traversing the depression, it is possible to directly calculate the volume of ~~a depression at any hypothetical outlet level~~water the depression can hold at each point in the traversal.

Once ~~$V_{\mathrm{dep,z_s}}$~~ $V_{\mathrm{dep,z_s}}$ is greater than or equal to the volume of water in the depression, ~~V_w~~ V_w, the plain queue contains all the cells to be flooded. At this point, the algorithm updates z_w, which is the water level within this depression. If ~~$V_w = V_{\mathrm{dep,z_s}}$~~ $V_w = V_{\mathrm{dep,z_s}}$, the algorithm sets $z_w = z_N$. If instead ~~$V_w < V_{\mathrm{dep,z_s}}$~~ $V_w < V_{\mathrm{dep,z_s}}$, the available volume in the depression is greater than the water volume, and the algorithm calculates z_w in the depression as follows:

$$V_w = z_w \sum_{i=1}^{N} a_i - \sum_{i=1}^{N} z_i a_i \tag{4}$$

$$z_w \sum_{i=1}^{N} a_i = \frac{1}{N} V_w + \sum_{i=1}^{N} z_i a_i \tag{5}$$

$$z_w = \left(\sum_{i=1}^{N} a_i \right)^{-1} \left(V_w + \sum_{i=1}^{N\,N} z_i a_i \right). \tag{6}$$

We call ~~this~~ Equation 6 the Lake-Level Equation (LLE). If all cells have a unit area, this simplifies to:

$$z_w = \frac{1}{N} \left(V_w + \sum_{i=1}^{N} z_i \right) \tag{7}$$

The conditional usage of the LLE described above is purely for computational efficiency: if ~~$V_w = V_{\mathrm{dep,z_s}}$~~ $V_w = V_{\mathrm{dep,z_s}}$, its solution is that $z_w = z_N$.

After solving for the water-surface elevation, the algorithm pops each cell in the plain queue ($c_i = c_1, \ldots, c_N$), corresponding to the flooded region, and sets its water elevation to the computed z_w. This is the final step of the Fill-Spill-Merge algorithm. At this point, it outputs a file representing the topography plus water thickness across the domain (i.e., topography with depressions filled or partially filled with water).

Because Fill-Spill-Merge routes water cell-by-cell to the pit cells of depressions and manages an array of water depths, it can be adapted for use with groundwater models, such as that described by Fan et al. (2013).

4 ~~Theoretical Analysis~~Algorithm performance

4.1 Theory

Here we use computational complexity as a means of contrasting the expected run-time of our algorithm against previous algorithms such as FlowFill (Callaghan and Wickert, 2019). To do so, we describe a simple iterative solver similar to FlowFill whose goal is to determine an appropriate water level for a depression. The solver operates on a one-dimensional domain of cells bounded by high cliffs on either side in which each cell may have a column of water. At each step, if the solver finds a discontinuity in water levels between two cells, it responds by averaging the heights of the cells' water columns. (The solver we describe is known as Jacobi's method.) The challenge we present to this solver is a direct analogue of routing flow along a stretch of river with negligible gradient and is very similar to routing flow across the surface of a lake or ocean.

For our analysis, we imagine that the system is initialized with a high column of water on the left and no water anywhere else. We call the cell with the water Cell 1. We call the cells to its right 2, 3, 4, and so on. During the solver's first step, Cell 1 is initialized. On its second step, Cell 1 averages its height with Cell 2. On the third step, Cell 2 averages with Cell 3 and Cell 1 then averages with Cell 2. On the fourth step, Cell 3 averages to 4, 2 averages to 3, and 1 averages with 2. Thus, the number of cells affected at each step are: 1, 2, 3, 4, and so on. Since there must be *at least* as many steps as there are cells, we can say that there are N steps. The total time, t_{compute}, is then

$$t_{\mathrm{compute}} = \sum_{i=1}^{N} i = \frac{N(N+1)}{2} \tag{8}$$

Thus, for any model (Callaghan and Wickert, 2019; Fan et al., 2013) that uses a scheme similar to our simple solver, the time required to solve the model is in $O(N^2)$.

In contrast, the new algorithm runs in $O(N \log N)$ time in the worst case. Moving water downhill (Section 3.1) is a flow-accumulation algorithm. This is known to take $O(N)$ time (Mark, 1988) and efficient variants exist for performing flow accumulation in parallel on large datasets (Barnes, 2017) and on GPUs (Barnes, 2019), though for simplicity we do not use these techniques here. Moving water within the depression hierarchy (Section 3.2) requires a depth-first post-order traversal of the entire hierarchy. This type of traversal is a foundational algorithm in computer science and takes $O(N)$ time. Each node in this traversal has the potential to overflow, which also results in a depth-first traversal, thereby requiring up to $O(N)$ time. However, by using a jump table that persists between calls to the overflow function, we ensure that it is able to identify the target of the overflow in amortized constant time; that is, the function is able to skip over fully-filled depressions. Finally, the algorithm floods the digital elevation model from the pit cells up. This requires a depth-first post-order traversal, which calls a flooding function (Section 3.3) on select subtrees of the DH. The depth-first traversal takes $O(N)$ time, as described above. The priority queue used for flooding nominally takes $O(N \log N)$ time in the worst case for floating-point data and $O(N)$ time in the worst case for integer data (Barnes et al., 2014). However, with specialized data structures the time can be reduced to $O(N)$ for both floating-point and integer data (Barnes et al., 2014). Most real datasets consist of many small depressions whose cell counts $N_{\mathrm{cells-in-dep}}$ are much smaller than the total number of cells in the digital elevation model. Therefore, the actual time is for this step is $O(N_{\mathrm{dep}} N_{\mathrm{cells-in-dep}})$, where N_{dep} is the total number of depressions and $N_{\mathrm{dep}} N_{\mathrm{cells-in-dep}}$ can

Dataset	Dimensions	Cells	FSM Time (s)	Total Time (s)
Madagascar	2000×1000	$2.0 \cdot 10^6$	0.1	0.4
U.S. Great Basin	1920×2400	$4.6 \cdot 10^6$	0.2	8.7
Australia	5640×4200	$2.3 \cdot 10^7$	9.1	15.6
Africa	9480×9000	$8.5 \cdot 10^7$	65.3	118.0
N&S America	18720×17400	$3.2 \cdot 10^8$	53.2	231.6
Minnesota 30m topobathy	34742×23831	$8.2 \cdot 10^8$	307.8	792.6

Table 1. Datasets used, their dimensions, and algorithm wall-times. Tests were performed on the Comet cluster run by XSEDE (see main text for full specifications). Times for Fill-Spill-Merge ("FSM Time") alone and this time plus the depression hierarchy construction time ("Total Time") are shown. Topographic data for Madagascar, the U.S. Great Basin, Australia, Africa, and North & South America, were clipped from the global GEBCO_08 30-arcsecond global combined topographic and bathymetric elevation data set (GEBCO, 2010). The Minnesota 30m topobathy data is the merged result of two data sources. The topography is resampled from the Minnesota Geospatial Information Office's 1m LiDAR Elevation Dataset (MNGEO - Minnesota Geospatial Information Office, 2019). Bathymetric data were provided by the Minnesota Department of Natural Resources (MNDNR - Minnesota Department of Natural Resources, 2014). Richard Lively of the Minnesota Geological Survey merged and combined these data sets.

be much less than N. Because the worst-case time complexity of any operation is $O(N)$, this bounds the time of the algorithm as a whole. However, to reduce the potential for bugs, we use the C++ standard library's $O(N \log N)$ priority queue in our implementation, at the cost of reduced performance.

To put this in more concrete terms, consider a long stretch of low-gradient river. Such a feature poses a lower bound on the time of our simple solver. North America's Red River of the North runs for 885 km with a gradient that is often on the order of 0.03 m km^{-1}. On a 90 m grid of floating-point data, the river would be 9,833 cells long. Our simple (Jacobi) solver would then take an estimated 97 million time units to reach a solution, whereas the new solver that we describe in this paper would take 9,833 time units, a 10,000× speed-up. Our empirical results, below, support both the theory and this expected value.

5 ~~Empirical Tests~~

4.1 Computational Performance

We have implemented the algorithm described above in `C++`~~11~~17 using the Geospatial Data Abstraction Library (GDAL) (GDAL Development Team, 2016) to read and write data. There are ~~981~~ 924 lines of code of which 50% are or contain comments. The code can be acquired from https://github.com/r-barnes/Barnes2020-FillSpillMerge and Zenodo (Barnes and Callaghan, 2020). The code contains extensive unit and end-to-end tests, which leverage both deterministic and random testing; the code passes a total of 214,990 test assertions and achieve 97% test coverage. The missed lines flag emergency situations which can only arise if there is a logic error, so they (in theory) cannot be reached.

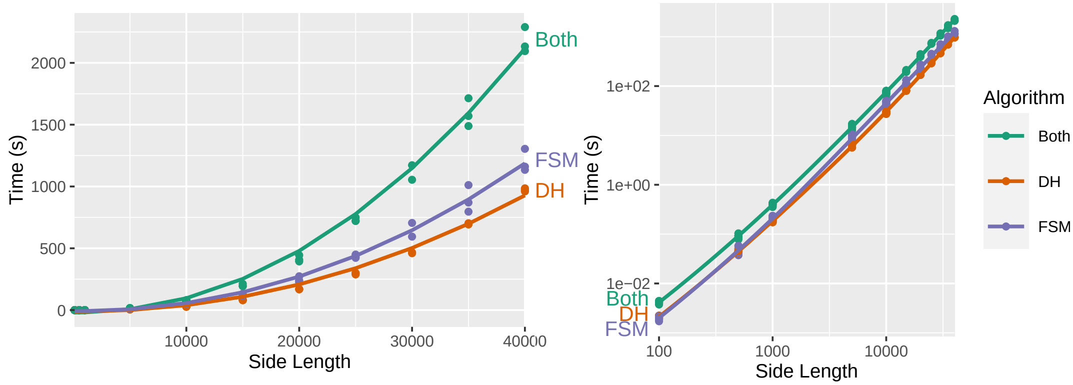

Figure 6. Performance on synthetic data. The left-hand plot shows the data on linear axes and the right-hand plot on log-log axes. The number of cells in each dataset is the square of the side length. The lines show $N \log N$ fits to each algorithm's time ($R^2 \approx 0.99$ for each). “DH" shows the performance of the Depression Hierarchy algorithm while “FSM" shows that of the Fill-Spill-Merge algorithm; “Both" shows the addition of these two values.

Tests were run on the Comet machine of the Extreme Science and Engineering Discovery Environment (XSEDE) (Towns et al., 2014). Each node of the machine has 2.5 GHz Intel Xeon E5-2680v3 processors with 24 cores per node and 128 GB of DDR4 DRAM. Code was compiled using GNU g++ 7.2.0 with full optimizations enabled. Scaling testson

We ran two sets of scaling tests, one on actual data and one on synthetic data. On actual data, our scaling tests cover datasets spanning three orders of magnitude in terms of their number of cellsare , as shown in Table 1. The GuessCompx package written in the R programming language by R package GuessCompx Agenis-Nevers et al. (2019) shows that an $O(N \log N)$ scaling relationship gives the best fit to the data, which agrees with the theory. Further tests are described in our Applications section (§4.1), below

To more precisely demonstrate performance, we run Fill-Spill-Merge on synthetic landscapes of various sizes generated using RichDEM's Perlin noise random terrain generator (Barnes, 2018). Multiple landscapes are generated and timed at each size to smooth timing variation due to both the data and fluctuations in the testing environment. This results in Figure 6, which again shows that the performance data gives a good fit to an $N \log N$ function.

5 Applications

4.1 Model intercomparison

Given a depression hierarchy data structure, Fill-Spill-Merge provides an efficient method to route water across any surface while taking depressions into account. Furthermore, Fill-Spill-Merge can be used to assess which depressions are most important in day-to-day or seasonal changes to the hydrologic system. For example, small depressions will become flooded and spill over even with relatively small amounts of water reaching them, while larger depressions may not be completely filled. These depressions impact the hydrologic connectivity of the landscape (Callaghan and Wickert, 2019). If standing water is retained between invocations of Fill-Spill-Merge, and new water added at each invocation, the algorithm can be used to simulate the movement of water across landscapes; we will explore this further in future work.

4.2 Field applications

We have compared Fill-Spill-Merge with a prior algorithm, FlowFill, at the same two sites used by Callaghan and Wickert (2019): a reach of the Sangamon River in Illinois (Figure 7) and the Río Toro basin in Argentina (Figure 8). Like Fill-Spill-Merge, FlowFill can be used to route water across a landscape while preserving real depressions, but the algorithm is significantly slower (Table 2). The two selected study sites provide very different landscapes for testing the performance of the algorithm. The Sangamon River site is located at 39.97°N, 88.72°W, in Illinois, USA. It is a low-relief, post-glacial landscape containing many closed depressions, which may impact hydrologic connectivity as a function of runoff (Lai and Anders, 2018). It furthermore contains a grid of roads and associated embankments whose elevations are significant when compared to regional relief and impact water flow paths and storage. Callaghan and Wickert (2019) resampled the 2.5 ft (0.76 m) resolution LiDAR DEM Illinois Geospatial Data Clearinghouse (2020) to 15 m resolution for analysis and manually removed several road bridges using GRASS GIS (Neteler et al., 2012) to prevent artificial pooling behind these; here we use the same modified DEM to enable a direct comparison between the algorithms. The Río Toro site is located mainly in Salta Province, Argentina, around 24.5°S, 65.8°W. This site exhibits more rugged fluvially sculpted topography (Hilley and Strecker, 2005). Callaghan and Wickert (2019) resampled the 12-m TanDEM-X DEM of this region (Krieger et al., 2013; Rizzoli et al., 2017) to 120 m resolution. Here we use this same resampled DEM for comparison.

As shown in Table 2, wall-times using Fill-Spill-Merge ranged from 0.227–0.243 s for the Sangamon River site and 0.300–0.319 s for the Río Toro site. This compares with times ranging from 20–643 s and 31-155 s, respectively, for FlowFill. These times for both sites correspond to a 86–2,645× reduction in wall-time. Since FlowFill was run with 24 processors, this translates to a 2,064–63,480× reduction in compute time. Considering that each of these example DEMs is quite small relative to modern full-resolution LiDAR-derived elevation data sets or continental-scale 30-meter DEMs (Table 1), this speed-up and its associated $O(N \log N)$ scaling provides a significant advantage for topographic analysis and solving associated problems in hydrology and geomorphology.

Although both FlowFill and Fill-Spill-Merge route water downslope, flooding depressions based on the quantity of available water, our results differ in some ways from those from those of FlowFill (Callaghan and Wickert, 2019). In both Figures 7 and 8, Fill-Spill-Merge flooded some depressions more deeply than FlowFill did , and , to a lesser extent, flooded a few and flooded some depressions with less water. One possible cause for this discrepancy is FlowFill's asymptotic approach to an equilibrium water level, which may prevent small volumes of water from reaching the depression to which they belong. On the other hand, depressions with a narrow outlet could be especially prone to being overfilled by FlowFill because its cell-by-cell algorithm could dynamically dam this outlet, routing additional water into the depression. Both of these possibilities are further linked to the fact that FlowFill dynamically evolves a land-plus-water flow-routing surface, whereas Fill-Spill-Merge routes flow

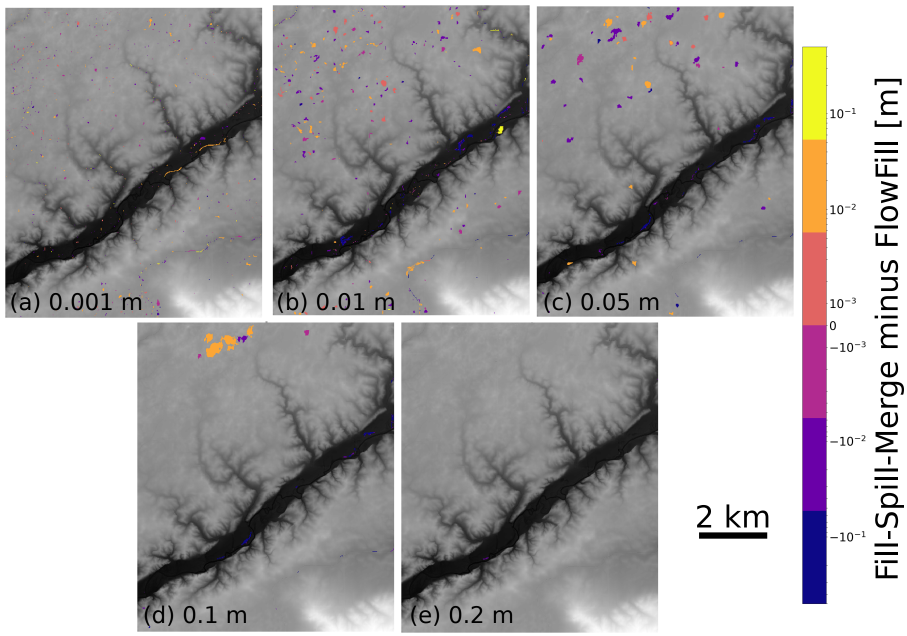

Figure 7. The difference between results of Fill-Spill-Merge and FlowFill at the Sangamon River site. The values for panels (a) to (e) indicate the depth of uniform runoff applied across the landscape for both algorithms. For example, in (a), each cell across the domain starts with ~~0.2~~ 0.001 m of surface water. ~~Green~~ Orange to yellow colors indicate locations where Fill-Spill-Merge had more water, and ~~blue to~~ purple to blue colors indicate locations where FlowFill had more water. Differences of less than 3 mm have been masked out. ~~Commonly~~Differences are generally small, and are likely a result of the iterative nature of the FlowFill algorithm which causes it to asymptotically approach the correct values. In some locations, Fill-Spill-Merge retains slightly more water in depressions ~~than~~ that FlowFill does. This could be due to water which has not yet finished moving downslope and into these depressions in the ~~iterative nature of the~~ FlowFill algorithm~~, which causes it to asymptotically approach the correct values~~. In ~~some~~ other locations, FlowFill has retained more water. One possible reason for this is that some depressions have a narrow outlet, through which Fill-Spill-Merge is able to move all water as appropriate but the cell-by-cell movement of water with FlowFill can produce transient dams that reroute additional water towards these subcatchments. This DEM was prepared by Lai and Anders (2018) and Callaghan and Wickert (2019) from LiDAR topographic data provided by the Illinois State Geological Survey (Illinois Geospatial Data Clearinghouse, 2020).

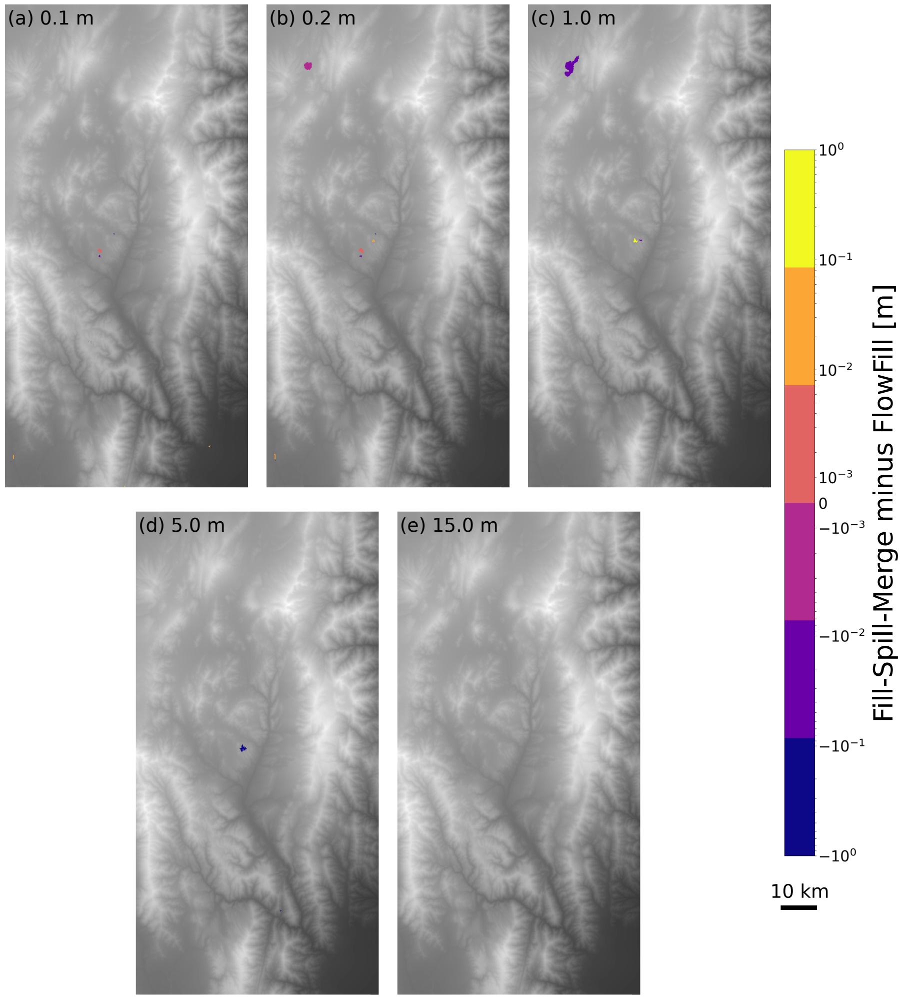

Figure 8. The difference between results of Fill-Spill-Merge and FlowFill at the Río Toro site. The values for panels (a) to (~~d~~e) indicate the depth of uniform runoff applied across the landscape for both algorithms. For example, in (a), each cell across the domain starts with ~~15~~ 0.1 m of surface water. ~~Green~~ Orange to yellow colors indicate locations where Fill-Spill-Merge had more water, and ~~blue to~~ purple to blue colors indicate locations where FlowFill had more water. Differences of less than 3 mm have been masked out. In panel (~~a~~e), 15 m of water was enough to fill all depressions with both algorithms, so there are no differences between the two. The most significant difference is seen in panel (c), where ~~Fill-Spill-Merge~~ FlowFill retained additional water in a large depression. This is likely due to transient damming of its narrow inlet in FlowFill's cell-by-cell method of moving water, which may have prevented the full volume of water from ~~flowing into~~ leaving the depression. This DEM was generated with data acquired from the TanDEM-X mission (Krieger et al., 2013; Rizzoli et al., 2017).

	Sangamon			Río Toro		
Runoff depth [m]	FlowFill	FSM	Speed-up	FlowFill	FSM	Speed-Up
15	642.65	0.243	2645	154.70	0.317	488
10	626.59	0.241	2600	124.37	0.309	402
5	570.02	0.241	2365	93.56	0.300	312
1	472.33	0.241	1960	53.09	0.316	168
0.2	508.87	0.235	2165	38.30	0.316	121
0.1	464.15	0.230	2018	35.75	0.301	119
0.05	418.71	0.243	1723	33.62	0.316	106
0.01	200.81	0.227	885	32.06	0.315	102
0.001	20.12	0.235	86	30.99	0.319	97

Table 2. Time comparison of Fill-Spill-Merge vs FlowFill. Wall-times are in seconds comparing FlowFill (Callaghan and Wickert, 2019) parallelized across 24 cores versus Fill-Spill-Merge on a single core. Using FlowFill, wall-times increased with the depth of applied runoff and on flatter landscapes. Using FSM, wall-time is independent of depth of applied runoff and ruggedness of landscape, but increases for larger domains. FSM's wall-times were 86–2,645 times faster than FlowFill for these examples; compute times were 2,064–63,480 times faster.

just over the land surface. These differences make FlowFill more useful for understanding temporal changes in surface water distribution, while Fill-Spill-Merge provides a more accurate snapshot of surface hydrology under equilibrium conditions.

5 Conclusions

Here we leverage the depression hierarchy data structure (Barnes et al., 2020) to route flow through surface depressions in a realistic, yet efficient, manner. In comparison to previous approaches, such as Jacobi iteration, the new algorithm runs in log-linear time in the input size and is accompanied by ~~extensively-commented~~ extensively commented source code. This computationally efficient algorithm may help us to better understand hydrologic connectivity and water storage across ~~hummocky land surfaces~~the land surface, and is an important step forwards in recognising the importance of depressions as real-world features in digital elevation models.

Code availability. Complete, well-commented source code, an associated makefile, and correctness tests are available from https://github.com/r-barnes/Barnes2020-FillSpillMerge and Zenodo (Barnes and Callaghan, 2020).

Author contributions. KC and AW conceived the problem. RB conceived the algorithm and developed initial implementations. KC and RB completed, debugged and tested the algorithm. All authors contributed to the preparation of the manuscript.

Competing interests. The authors declare that they have no conflict of interest.

Acknowledgements. RB was supported by the Department of Energy's Computational Science Graduate Fellowship (Grant No. DE-FG02-97ER25308) and, through the Berkeley Institute for Data Science's PhD Fellowship, by the Gordon and Betty Moore Foundation (Grant GBMF3834) and by the Alfred P. Sloan Foundation (Grant 2013-10-27).

KLC was supported by the National Science Foundation under grant no. EAR-1903606, the University of Minnesota Department of Earth Sciences Junior F Hayden Fellowship, the University of Minnesota Department of Earth Sciences H.E. Wright Footsteps Award, and start-up funds awarded to AW by the University of Minnesota.

Empirical tests and results were performed on XSEDE's Comet supercomputer (Towns et al., 2014), which is supported by the National Science Foundation (Grant No. ACI-1053575). Portability and debugging tests were performed on the Mesabi machine at the Minnesota Supercomputing Institute (MSI) at the University of Minnesota (http://www.msi.umn.edu).

The Deutsches Zentrum für Luft- und Raumfahrt (DLR) provided 12 m TanDEM-X DEM coverage of the Río Toro catchment via proposal DEM_GEOL1915 awarded to Taylor Schildgen, Andrew Wickert, Stefanie Tofelde, and Mitch D'Arcy. Jingtao Lai and Alison Anders provided a copy of their Sangamon River DEM.

This collaboration resulted from a serendipitous meeting at the Community Surface Dynamics Modeling System (CSDMS) annual meeting, which RB had attended on a CSDMS travel grant.

6 Pseudocode

6.1 MoveWaterInDepHier

```
function MoveWaterInDepHier(root, DH, JumpTable)
Let root be the id of the depression we're currently considering
Let DH be a Depression Hierarchy
Let JumpTable be a hash table mapping DH labels to DH labels

▷ For "children" of leaves
if root=NoValue then return

▷ The traversal
for each ocean-linked child c of root do
    Call MoveWaterInDepHier(c, DH, JumpTable)
end for
Call MoveWaterInDepHier(c.left_child, DH, JumpTable)
Call MoveWaterInDepHier(c.right_child, DH, JumpTable)

if root=OCEAN then return

if root has children and both their depression volumes equal their water volumes and root's water volume is zero then
    root.water_vol += root.left_child.water_vol
    root.water_vol += root.right_child.water_vol
end if

if root.water_vol>root.dep_vol then
    Call OverflowInto(root, root.parent, DH, JumpTable, 0)
end if
```

6.2 OverflowInto

```
function OverflowInto(root, StopNode, DH, JumpTable, ExtraWater)
Let root be the id of the depression we're currently considering
Let StopNode be the id of the depression that ends the traversal. It is the parent of the depression that first called this function.
Let DH be a Depression Hierarchy
Let JumpTable be a hash table mapping DH labels to DH labels
Let ExtraWater be the water that needs to be distributed in DH

▷ If depression is too full, get its excess so we can find a home for it
if root.water_vol>root.dep_vol then
    ExtraWater += root.water_vol - root.dep_vol
    root.water_vol = root.dep_vol
end if

if root=StopNode or root=OCEAN then
    root.water_vol += ExtraWater
    return root
end if

▷ 1st place to stash water: in this depression
if root.water_vol<root.dep_vol then
    Let C=root.dep_vol - root.water_vol
    if ExtraWater< C then
        root.water_vol = root.water_vol+ExtraWater
        ExtraWater = 0
    else
        root.water_vol = root.dep_vol
```

        ExtraWater -= C
    end if
end if

if *ExtraWater=0* **then**
    return root
end if

if *root*∈*JumpTable* **then**
    return *JumpTable*(root) = OverflowInto(*JumpTable(root), StopNode, DH, JumpTable, ExtraWater*)
end if

▷ 2nd place to stash water: in the depression's sibling
if *root.sib*≠NoValue **then**
    if *root.sib.water_vol*<*root.sib.dep_vol* **then**
        return *JumpTable(root)* = OverflowInto(*root.geolink, StopNode, DH, JumpTable, ExtraWater*)
    else if *root.sib.water_vol*>*root.sib.dep_vol* **then**
        e=root.sib.water_vol-root.sib.dep_vol
        ExtraWater += e
        root.sib.water_vol = root.sib.dep_vol
    end if
end if

▷ 3rd place to stash water: in the depression's parent
if *root.parent.water_vol=0* **and** *root* is not oceanlinked to *root.parent* **then**
    root.parent.water_vol += root.water_vol
    if root.sib≠NoValue **then**
        root.parent.water_vol += root.sib.water_vol
    end if
end if
return *JumpTable(root)* = OverflowInto(*root.parent, StopNode, DH, JumpTable, ExtraWater*)

6.3 FillDepressions

function FillDepressions(*PitCell, OutCell, DepLabels, WaterVol, dem, labels, wtd*)

Let *PitCell* be the cell to start filling from
Let *OutCell* be the outlet/spill cell
Let *DepLabels* be the labels contained within the metadepression we are trying to fill
Let *WaterVol* be the amount of water that needs to be spread throughout the depression
Let *dem* be the topography.
Let *labels* be the labels from the Depression Hierarchy
Let *wtd* be the depth of water in each cell.
Let *visited* be a hash set of cell ids
Let *PQ* be a priority queue sorted by increasing elevation
Let *affected* be a plain queue
Let T_e be the total elevation; initially 0

if *WaterVol=0* **then return**

Place *PitCell* into *PQ* and mark it visited
while PQ is not empty **do**
    Let *c=pop(PQ)*
    Let $V = |affected| \cdot c.elev - T_e$

    if *WaterVol*$< V$ **then**
        $W_L = (WaterVol + T_e)/|affected|$
        Set *wtd* for all cells in *affected* to W_L
        return
    end if

    if $c \neq$ *OutCell* **then**
        Place c into *affected*
        T_e += $c.elev$
    end if

Add all of *c*'s neighbours that belong to depressions in *DepLabels* and are not the outlet cell to *PQ* and mark them visited
if *PQ* is empty **then**
    Add *OutCell* to *PQ* and mark it visited
end if

---

end while

References

Agenis-Nevers, M., Bokde, N. D., Yaseen, Z. M., and Shende, M.: GuessCompx: An empirical complexity estimation in R, arXiv:1911.01420v1, 2019.

Arnold, N.: A new approach for dealing with depressions in digital elevation models when calculating flow accumulation values, Progress in Physical Geography: Earth and Environment, 34, 781–809, https://doi.org/10.1177/0309133310384542, https://doi.org/10.1177/0309133310384542, 2010.

Barnes, R.: Parallel non-divergent flow accumulation for trillion cell digital elevation models on desktops or clusters, Environmental Modelling & Software, 92, 202–212, https://doi.org/10.1016/j.envsoft.2017.02.022, 2017.

Barnes, R.: r-barnes/richdem: Zenodo DOI Release, Software, https://doi.org/10.5281/zenodo.1295618, https://doi.org/10.5281/zenodo.1295618, 2018.

Barnes, R.: Accelerating a fluvial incision and landscape evolution model with parallelism, Geomorphology, 330, 28–39, https://doi.org/10.1016/j.geomorph.2019.01.002, 2019.

Barnes, R. and Callaghan, K.: Depression Hierarchy Source Code, https://doi.org/10.5281/zenodo.3238558, 2019.

Barnes, R. and Callaghan, K.: Fill-Spill-Merge Source Code, https://doi.org/10.5281/zenodo.3755142, 2020.

Barnes, R., Lehman, C., and Mulla, D.: Priority-flood: An optimal depression-filling and watershed-labeling algorithm for digital elevation models, Computers & Geosciences, 62, 117 – 127, https://doi.org/10.1016/j.cageo.2013.04.024, 2014.

Barnes, R., Callaghan, K., and Wickert, A.: Computing water flow through complex landscapes, Part 2: Finding hierarchies in depressions and morphological segmentations, Earth Surface Dynamics, https://doi.org/10.5194/esurf-2019-34, 2020.

Beucher, S.: Watershed, Hierarchical Segmentation and Waterfall Algorithm, in: Mathematical Morphology and Its Applications to Image Processing, edited by Viergever, M. A., Serra, J., and Soille, P., vol. 2, pp. 69–76, Springer Netherlands, Dordrecht, https://doi.org/10.1007/978-94-011-1040-2_10, 1994.

Braun, J. and Willett, S. D.: A very efficient O(n), implicit and parallel method to solve the stream power equation governing fluvial incision and landscape evolution, Geomorphology, 180-181, 170–179, https://doi.org/10.1016/j.geomorph.2012.10.008, 2013.

Breckenridge, A. and Johnson, T. C.: Paleohydrology of the upper Laurentian Great Lakes from the late glacial to early Holocene, Quaternary Research, 71, 397–408, https://doi.org/10.1016/j.yqres.2009.01.003, 2009.

Cabrol, N. A. and Grin, E. A.: Distribution, classification, and ages of Martian impact crater lakes, Icarus, 142, 160–172, 1999.

Callaghan, K. L. and Wickert, A. D.: Computing water flow through complex landscapes – Part 1: Incorporating depressions in flow routing using FlowFill, Earth Surface Dynamics, 7, 737–753, https://doi.org/10.5194/esurf-7-737-2019, 2019.

Cordonnier, G., Bovy, B., and Braun, J.: A Versatile, Linear Complexity Algorithm for Flow Routing in Topographies with Depressions, Earth Surface Dynamics Discussions, 2018, 1–18, https://doi.org/10.5194/esurf-2018-81, 2018.

Fan, Y., Li, H., and Miguez-Macho, G.: Global Patterns of Groundwater Table Depth, Science, 339, 940–943, https://doi.org/10.1126/science.1229881, 2013.

Fenner, T. I. and Loizou, G.: Loop-free Algorithms for Traversing Binary Trees, BIT, 24, 33–44, https://doi.org/10.1007/BF01934513, 1984.

GDAL Development Team: GDAL - Geospatial Data Abstraction Library, Open Source Geospatial Foundation, available at http://www.gdal.org., 2016.

GEBCO: General Bathymetric Chart of the Oceans (GEBCO), GEBCO_08 grid, version 20100927, http://www.gebco.net, 2010.

Hilley, G. E. and Strecker, M. R.: Processes of oscillatory basin filling and excavation in a tectonically active orogen: Quebrada del Toro Basin, NW Argentina, GSA Bulletin, 117, 887–901, https://doi.org/10.1130/B25602.1, 2005.

Illinois Geospatial Data Clearinghouse: Illinois Height Modernization (ILHMP), https://clearinghouse.isgs.illinois.edu/data/elevation/illinois-height-modernization-ilhmp-lidar-data, 2020.

Jenson, S. and Domingue, J.: Extracting Topographic Structure from Digital Elevation Data for Geographic Information System Analysis, Photogrammetric Engineering and Remote Sensing, 54, 1–5, https://doi.org/0099-1112/88/5411-1593$02.25/0, 1988.

Krieger, G., Zink, M., Bachmann, M., Bräutigam, B., Schulze, D., Martone, M., Rizzoli, P., Steinbrecher, U., Antony, J. W., De Zan, F., et al.: TanDEM-X: A radar interferometer with two formation-flying satellites, Acta Astronautica, 89, 83–98, 2013.

Lai, J. and Anders, A. M.: Modeled Postglacial Landscape Evolution at the Southern Margin of the Laurentide Ice Sheet: Hydrological Connection of Uplands Controls the Pace and Style of Fluvial Network Expansion, Journal of Geophysical Research: Earth Surface, 123, 967–984, https://doi.org/10.1029/2017JF004509, 2018.

Li, S., MacMillan, R., Lobb, D. A., McConkey, B. G., Moulin, A., and Fraser, W. R.: Lidar DEM error analyses and topographic depression identification in a hummocky landscape in the prairie region of Canada, Geomorphology, 129, 263–275, https://doi.org/10.1016/j.geomorph.2011.02.020, 2011.

Lindsay, J. and Creed, I.: Removal of artifact depressions from digital elevation models: towards a minimum impact approach, Hydrological processes, 19, 3113–3126, https://doi.org/10.1002/hyp.5835, 2005a.

Lindsay, J. B.: Efficient hybrid breaching-filling sink removal methods for flow path enforcement in digital elevation models: Efficient Hybrid Sink Removal Methods for Flow Path Enforcement, Hydrological Processes, https://doi.org/10.1002/hyp.10648, 2015.

Lindsay, J. B.: Efficient hybrid breaching-filling sink removal methods for flow path enforcement in digital elevation models: Efficient Hybrid Sink Removal Methods for Flow Path Enforcement, Hydrological Processes, 30, 846–857, https://doi.org/10.1002/hyp.10648, http://doi.wiley.com/10.1002/hyp.10648, 2016.

Lindsay, J. B. and Creed, I. F.: Removal of artifact depressions from digital elevation models: Towards a minimum impact approach, Hydrological Processes, 19, 3113–3126, https://doi.org/10.1002/hyp.5835, 2005b.

Mark, D.: Modelling in Geomorphological Systems, chap. Network models in geomorphology, pp. 73–97, John Wiley & Sons, 1988.

Martz, L. W. and Garbrecht, J.: The treatment of flat areas and depressions in automated drainage analysis of raster digital elevation models, Hydrological Processes, 12, 843–855, https://doi.org/10.1002/(SICI)1099-1085(199805)12:6<843::AID-HYP658>3.0.CO;2-R, 1998.

Martz, L. W. and Jong, E. d.: CATCH: A FORTRAN program for measuring catchment area from digital elevation models, Computers and Geosciences, 14, 627–640, https://doi.org/10.1016/0098-3004(88)90018-0, 1988.

MNDNR - Minnesota Department of Natural Resources: Lake Bathymetric Outlines, Contours, Vegetation, and DEM, https://gisdata.mn.gov/dataset/water-lake-bathymetry, 2014.

MNGEO - Minnesota Geospatial Information Office: LiDAR Elevation Data for Minnesota, http://www.mngeo.state.mn.us/chouse/elevation/lidar.html, 2019.

Neteler, M., Bowman, M. H., Landa, M., and Metz, M.: GRASS GIS: A multi-purpose open source GIS, Environmental Modelling and Software, 31, 124–130, https://doi.org/10.1016/j.envsoft.2011.11.014, http://dx.doi.org/10.1016/j.envsoft.2011.11.014, 2012.

O'Callaghan, J. and Mark, D.: The extraction of drainage networks from digital elevation data, Computer Vision, Graphics, and Image Processing, 28, 323–344, https://doi.org/10.1016/S0734-189X(84)80011-0, 1984.

Reheis, M.: Highest Pluvial-Lake Shorelines and Pleistocene Climate of the Western Great Basin, Quaternary Research, 52, 196–205, https://doi.org/10.1006/qres.1999.2064, 1999.

Riddick, T., Brovkin, V., Hagemann, S., and Mikolajewicz, U.: Dynamic hydrological discharge modelling for coupled climate model simulations of the last glacial cycle: the MPI-DynamicHD model version 3.0, Geoscientific Model Development, 11, 4291–4316, https://doi.org/10.5194/gmd-11-4291-2018, 2018.

Rizzoli, P., Martone, M., Gonzalez, C., Wecklich, C., Tridon, D. B., Bräutigam, B., Bachmann, M., Schulze, D., Fritz, T., Huber, M., et al.: Generation and performance assessment of the global TanDEM-X digital elevation model, ISPRS Journal of Photogrammetry and Remote Sensing, 132, 119–139, 2017.

Salembier, P. and Pardas, M.: Hierarchical morphological segmentation for image sequence coding, IEEE Transactions on Image Processing, 3, 639–651, https://doi.org/10.1109/83.334980, 1994.

Schwanghart, W. and Scherler, D.: Bumps in river profiles: uncertainty assessment and smoothing using quantile regression techniques, Earth Surface Dynamics, 5, 821–839, https://doi.org/10.5194/esurf-5-821-2017, 2017.

Sedgewick, R. and Wayne, K.: Algorithms, Addison-Wesley, 4 edn., 2011.

Skiena, S. S.: The Algorithm Design Manual, Springer, 2008.

Soille, P.: Optimal removal of spurious pits in grid digital elevation models, Water Resources Research, 40, 1–9, https://doi.org/10.1029/2004WR003060, 2004.

Soille, P., Vogt, J., and Colombo, R.: Carving and adaptive drainage enforcement of grid digital elevation models, Water Resources Research, 39, 1366, https://doi.org/10.1029/2002WR001879, 2003.

Towns, J., Cockerill, T., Dahan, M., Foster, I., Gaither, K., Grimshaw, A., Hazlewood, V., Lathrop, S., Lifka, D., Peterson, G. D., et al.: XSEDE: accelerating scientific discovery, Computing in Science & Engineering, 16, 62–74, 2014.

Wallis, C., Watson, D., Tarboton, D., and Wallace, R.: Parallel flow-direction and contributing area calculation for hydrology analysis in digital elevation models, in: Preceedings of the 2009 International Conference on Parallel and Distributed Processing Techniques and Applications, Las Vegas, Nevada, http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.158.2864, 2009.

Wickert, A. D.: Reconstruction of North American drainage basins and river discharge since the Last Glacial Maximum, Earth Surface Dynamics, 4, 831–869, https://doi.org/10.5194/esurf-4-831-2016, 2016.

Wu, Q. and Lane, C. R.: Delineation and quantification of wetland depressions in the Prairie Pothole Region of North Dakota, Wetlands, 36, 215–227, 2016.

Wu, Q., Liu, H., Wang, S., Yu, B., Beck, R., and Hinkel, K.: A localized contour tree method for deriving geometric and topological properties of complex surface depressions based on high-resolution topographical data, International Journal of Geographical Information Science, 29, 2041–2060, https://doi.org/10.1080/13658816.2015.1038719, https://doi.org/10.1080/13658816.2015.1038719, 2015.

Wu, Q., Lane, C. R., Wang, L., Vanderhoof, M. K., Christensen, J. R., and Liu, H.: Efficient Delineation of Nested Depression Hierarchy in Digital Elevation Models for Hydrological Analysis Using Level-Set Method, Journal of the American Water Resources Association, 0, https://doi.org/10.1111/1752-1688.12689, 2018.