Earth Surf. Dynam. Discuss.,
https://doi.org/10.5194/esurf-2018-81-RC3, 2019

Interactive comment on "A Versatile, Linear Complexity Algorithm for Flow Routing in Topographies with Depressions" *by* Guillaume Cordonnier et al.

Anonymous Referee #2

Received and published: 17 February 2019

The authors present a new depression-filling and breaching algorithm. The work seems technically sound and I look forward to seeing this in publication soon.

The only flaw in the work is that the empirical comparisons versus existing work are weak. I discuss this in greater detail below.

I've recommended the paper for "Major Revision" due to this, but, were it not for this point, I think the rest of the revisions would be minor.

Major Comments ====================== 1. This is a really cool idea and a well-written paper - nice work!

2. There are a number of minor grammatical issues in the paper that need sorting. I've identified most of the ones I found, but others may be lurking.

3. The only major flaw with this paper is that the benchmarks seem ill-constructed. Figure 5a is nice, though I think it would be improved by showing Zhou (2017). (If I recall correctly, RichDEM also has an implementation of this, so it should be easy to do.) I think it would also be improved by considering larger datasets, perhaps up to 10,000x10,000. I think Figure 5c needs some real work. Wei (2018) should be included here, no doubt. Figure 5a shows that it has completely different performance characteristics than Barnes (2014a), so this is critical to contextualizing your work. Surely modifying the algorithm to include surface offsets or flow directions isn't too difficult? Otherwise, perhaps Wei et al. or Barnes et al. have such an implementation? Please find a way to address this.

4. Similarly, as discussed below, the Wei, Barnes, and Zhou algorithms are all adapted to large datasets and, when I've looked at the source, written in a very literate style (minimal bit-hacking). This raises the question of whether versions of these algorithms optimized for use on smaller datasets would perform better in your benchmark. I think you've done a good job of showing that your algorithm ranks with the state of the art, but the comparison isn't strong enough to conclude that the design itself is superior. You may or may not consider this worth addressing.

5. Please make the source code available online. It seems to only be available as a pull request at the moment and this is not suitable for publication. Ideally, the source code and associated tests would be available on Github and archived on Zenodo or Figshare.

Detailed Comments ======================

p. 2, Line 3: "have been optimized so that they can deliver acceptable performance when used with large datasets."

p. 3, Line 8: Check grammar. "using of fast algorithms of linear complexity at each step of the procedure, which now makes the whole computation very efficient."

p. 3, Line 9: Check grammar. "Not only this method allows using" [I feel like I've made an abrupt transition between authors; additional proofreading prior to submission would have been good.]

p. 3, Line 15: "modelling". On p. 1, Line 15 you used "modeling". Please choose a single convention.

p. 3, Line 22: "zk>zn, and rcv(k)=n". Could it be that "zk>zn and not rcv(k)=n" or "not zk>zn and rcv(k)=n"? That is, are both conditions necessary, or does one imply the other? You should clarify this.

p. 3, Line 25: Lindsay (2016) also defines a "flat-bottomed depression". In such a depression, no cell is a proper local minimum. Presumably such cells would be labeled as singular nodes due to the manner in which you construct the rcv array, though you don't make this special case explicit in your paper. You should clarify this.

p. 4, Pt 1: This is the inevitable result of using convergent/unidirectional flow routing.

p. 4, Pt 2: Check grammar.

p. 4, Pt 3: I have difficulty following this point. A minimum can only be defined with respect to a set of possible alternatives and it is not clear at this point what the set of alternatives is. Since you haven't introduced it previously, I would appreciate a forward reference to the point in your paper where you do introduce it.

p. 4, Line 8: "whether the singular node is a base level node". I feel like the term "base level node" is unintuitive, since many nodes could be at, or below, a given base level, but be interior to a landscape. The term "boundary node" seems more intuitive.

p. 4, Line 9: "that all have an elevation below a given water level". I find this definition counterintuitive. A basin includes regions outside of a depression (up to the peaks of mountains). We are now calling a depression the set of cells below a chosen water-level, but nothing about this paragraph constrains a water level to be contained with a Lindsay (2016) typology-style depression. It seems possible to set the spill as a mountain peak and declare the entire DEM a depression. I wonder if there's a more intuitive term that can be used here? Or perhaps a constraint can be added?

p. 4, Line 12: "precipitations" -> "precipitation"

p. 4, Line 15: "respective spill and ensure" -> "respective spill, and ensure". I think the use of an Oxford comma here is useful in alerting the reader that the third item has been reached.

p. 4, Lines 19-30: I think these steps are performed iteratively, but don't see mention of this. Could you please clarify?

p. 5, Line 3: "visiting the donors recursively". I followed the reference to the Appendix, but did not find it to be enlightening since it requires knowledge of the stack construction from Braun and Willet (2013). I think you are doing a depth-first traversal of all of the cells in a basin (those cells whose flow ultimately terminates at a given singular node). If so, stating this directly would make things clearer. The word recursion also implies a particular computational motif—functions calling themselves—which isn't implied by the Appendix. If you are using recursion, the Appendix should be modified. However, this is a risky strategy for larger datasets since some operating systems impose relatively low limits on the depth of recursion.

Algorithm 6: It might be useful to note somewhere that the link is undirected such that Links.contains((n1,n2))==Links.contains((n2,n1)). The current implementation seems to rely on the ordering of the stack to ensure this is true, but in another context it may be useful to relax that ordering.

Algorithm 6: 'if elevation of the pass of Links(link) < zpass then' finds the highest link between two basins. I think this is the opposite of what you want.

p. 6, Line 2: "To illustrate the problem" Which problem?

p. 6, Line 3: "Hence, routing the flow across the basins consists in connecting all outflows such that the resulting flow paths, from inner basins to the boundary basins [...]" As I describe elsewhere in this review, this isn't strictly true and represents a serious restriction on the use of this algorithm. (TODO)

p. 6, Line 18: "Kriskal's algorithm" -> "Kruskal's algorithm" // "classical algorithm" -> "classical algorithms"

p. 6, Line 19: "majored by". I'm not familiar with this terminology.

p. 7, Line 1: "using this algorithm come a global upper bound". Grammar.

p. 7, First paragraph: "Union Find" is usually written as "Union-Find".

p. 7, Line 11: "lower that O(log N)" -> "lower than"

p. 7, Line 15: "in a plan" -> "in a plane"

Algorithm 2: This is super cool! What a good find. The pseudocode here seems as though it won't get you to O(N) time since each step must be implemented carefully to make that guarantee. Perhaps some statement to this effect with a reference to Mareš (2002) is in order?

p. 8, Line 1: "rather unlikely". This approach to correctness seems a bit cavalier. I expect that a good algorithm works in all cases, not merely when it's convenient. The rest of this paragraph seems to imply that, in theory, this problem can be handled. But I'm left with the feeling that your implementation may contain subtle bugs that you've thought are too unlikely to pay attention to! I would like assurance that you check for such exceptional situations and either handle them gracefully and correctly, or flag them for user intervention. It's also unclear why the assumption you state solves the problem and why it isn't possible to build adversarially nested structures.

p. 8, 2.3.1: Are all of the depressions carved to make a continuous flow network? If so, the situation shown in Figure 3 doesn't seem as though it can happen: Flow will progress along the carved network before water ever pools. Similarly, if depression-filling is applied, pooling won't occur.

p. 9, Line 4: "minimal distance" The minimal euclidean distance ignoring obstacles? I think this could be clearer.

p. 9: Please number equations.

Algorithms 3 and 4: On Page 3, a Donors array was listed as one of the data structures used. Presumably that needs to be updated as well, though it doesn't appear here. Some explanation of this is needed.

Algorithm 4: I struggled somewhat to follow the logic here. Why is a cost function used instead of a simple breadth-first ordering? Perhaps a figure showing what the different flow paths look like would be helpful. The algorithm also seems as though it will terminate at the depression's pit cell. However, the flow pattern to the outlet will then have half of the depression exhibiting convergent flow to the pit cell and half of the depression showing convergent flow to the outlet. This flow pattern is unrealistic if the entire depression is treated as having been filled with flow superimposed on the resultant flat surface.

Figure 3c: The depiction of standing water shown here is at odds with my understanding of what the flow network looks like following carving or depression-filling. Or perhaps erosion fills in the channels connecting depressions, resulting in standing water? But then one wonders what the point of all of this was?

Figure 4: Since the updates presented in the paper so far only affect the receiver network, it wasn't immediately clear to me how the topography had been generated.

p. 11: If only the receiver array is modified, how is the Equation From Page 9 implemented when water flows uphill? Campforts and Grovers (2015, "Keeping the Edge") show that the Stream Power Law as it's used in Fastscape doesn't preserve knickpoints or other topography discontinuities well. The channels carved here seem as though they are representative of the sorts of terrain this critique applies to. Some discussion of modeling philosophy might be useful.

Figure 5: What I find most interesting about this figure is that Wei 2018 is competitive with your results in (a), but gets dropped in (c). Why, in (c), do you choose to compare your work against a slower algorithm?

Figure 5: Could you put (a), (b), and (c) in bold in the caption to help readers find the relevant text?

p. 12 and 13: Given this description, I don't see how your benchmarks will be meaningful. Using the Wei (2018) algorithm in conjunction with the Barnes (2014b) algorithm for the benchmarks of Figure 5a makes a certain sense, but why not simply modify the Wei (2018) algorithm using the PF+e strategy of Barnes (2014a)? Wouldn't this comparison be more fair?

Why does Figure 5c uses Barnes (2014a)? I can't find an explanation anywhere. It seems like comparing against Barnes (2014a) is a poor choice. Barnes (2014a) suggests a path for accelerating Priority-Flood by reducing the number of cells passed through the Priority-Queue. Barnes (2014a), Zhou (2016), and Wei (2018) all present progressively better algorithms for doing this. It's better to think of all of these algorithms as being O(N + M log M), where M is the number of cells which pass through the priority-queue. M decreases dramatically between Barnes (2014a) and Wei (2018), as shown in Wei (2018, Figure 8). By the time Wei (2018), the O(N) term might even dominate the O(M log M) term, meaning that Wei (2018) is a linear-time algorithm! Indeed, your Figure 5a suggests this. In any case, Figure 5c seems to overstate your results since it's not a proper performance comparison.

p. 13, Line 8: "we reuse the implementations available in the RichDEM" This seems reasonable, but it should be done with caveats. As you note on p. 2, this set of algorithms has "been optimized so that they can deliver acceptable performance when used with large datasets." It seems as though your algorithm uses significantly more space than the B, Z, and W algorithms you cite in your paper. The Barnes (2014a) algorithms work in-place using only the priority-queue and a boolean array as extra memory; that is, it requires about O(sqrt(N)) additional space. In comparison, your code has a Receivers array, Donors array (which takes O(8N) space), and Union-Find array, leading to a O(10N) space requirement. Trading space for time is a well-known optimization technique. This raises the question of how the B,Z,W algorithms would perform had they been optimized for small datasets. It also raises the question of how your algorithm would perform on larger datasets. The average size of the test dataset used by Wei (2018), for instance, was 2*10ˆ8 cells - two orders of magnitude larger than the 1500x1500 grid you use and three orders of magnitude larger than the Figure 5a benchmark. I don't think additional tests are necessary, but it should be clarified somewhere that the algorithms you're comparing against are being used for dataset sizes they weren't designed for.

p. 14, Lines 10-15: This is a really nice analysis. However, I'm not sure I agree with it entirely. This line: "Because those variants are more complex, k4 and k5 have higher values." Why is the cost of using a priority-queue higher for the Zhou and Wei algorithms? I would think that k2=k5. I agree that k4>k1. However, one of the reasons priority queues are expensive is because they have poor cache coherence. (Search online for "Latency numbers every computer programmer should know". See also, Luengo-Hendriks (2010)'s discussion of priority queue performance variation.) This means I expect k1<k2 and k4<k5. Wei trades computation (which is fast) for memory management (which is slow); the question then is whether nl is small enough to offset the higher cost of k4.

p. 14, Lines 10-15: Some analysis of the space used by the various algorithms would improve this discussion.

p. 15, Line 17: The Barnes (2014a) algorithm also includes a variant (+FlowDirs) that could be used to construct a receivers array similar to yours. Presumably Zhou and

Wei could be modified in this way as well.

p. 15, Line 20: Barnes (2019, "Accelerating a fluvial incision") maps Fastscape to multi-core architectures and GPUs, showing a 43x speed-up versus serial performance, but also fails to find a depression-filling/breaching algorithm for that environment.

p. 15, Line 24: Please add a link to a Zenodo archive or Figshare copy of your code. This has the advantage of giving it a doi number and assuring readers can find the version used in your publication.

Bibliography: doi numbering is inconsistent. Please include doi numbers whenever possible.

Flat bottomed-depressions: It would be helpful to make a note of the special cases where many cells are singular and physically adjacent. I think your algorithm handles this gracefully, but it would be nice to make that explicit.

For basins with large collection zones and small depressions, overflow might need to go upstream as well as downstream with respect to what's specified in the receiver array. This limits the utility of this algorithm in situations where dynamic flow is being modeled.

Interactive comment on Earth Surf. Dynam. Discuss., https://doi.org/10.5194/esurf-2018-81, 2018.