A recursive algorithm for calculating the longest flow path and its iterative implementation*

Huidae Cho[a,*]

[a]*Institute for Environmental and Spatial Analysis, University of North Georgia, Oakwood, GA 30566, USA*

ARTICLE INFO

Keywords:
Longest flow path
Watershed
Hydrology
Open source
GIS
GRASS GIS

ABSTRACT

The longest flow path is widely used for studying hydrology. Traditionally, both or either of upstream and downstream flow length rasters are required to calculate the longest flow path. When processing multiple subwatersheds, this approach requires separate calculations of the downstream flow length raster for all the subwatersheds. However, raster computation involves a lot of disk input/output and can be slow. By defining the longest flow path recursively and introducing a branching strategy based on Hack's law, this study proposes a new longest flow path algorithm that computes as few rasters as possible to reduce computational time and improve efficiency. To avoid stack overflows by excessive recursion, its iterative counterpart algorithm was also proposed. The proposed algorithms were implemented as a GRASS GIS module. Benchmark experiments proved that the new module outperforms an existing tool for a commercial GIS.

Software availability

- GRASS GIS software: Free under the GNU GPL license
  - GRASS GIS: https://grass.osgeo.org/
  - r.accumulate: https://github.com/OSGeo/grass-addons/tree/master/grass7/raster/r.accumulate
    - Recursive version with the `-r` flag (r.accumulate-recursive)

*NOTICE: This is the author's version of a work that was accepted for publication in Environmental Modelling & Software. Changes resulting from the publishing process, such as peer review, editing, corrections, structural formatting, and other quality control mechanisms may not be reflected in this document. Changes may have been made to this work since it was submitted for publication. A definitive version was subsequently published in Environmental Modelling & Software 131C, 104774 (July 2020), doi:10.1016/j.envsoft.2020.104774.

CITATION: Cho, H., 2020. A recursive algorithm for calculating the longest flow path and its iterative implementation. Environmental Modelling & Software 131C, 104774.

*Corresponding author
hcho@isnew.info (H. Cho)
https://hcho.isnew.info (H. Cho)
ORCID(s): 0000-0003-1878-1274 (H. Cho)
https://twitter.com/HuidaeCho (H. Cho)
https://www.linkedin.com/profile/view?id=HuidaeCho (H. Cho)

    * Iterative version by default without the `-r` flag (r.accumulate-iterative)

- Operating system requirements
  - Microsoft Windows XP or newer, macOS 10.4.10 or newer, recent GNU/Linux or a UNIX variant

1. Introduction

This paper proposes a new recursive algorithm and its iterative implementation for calculating the longest flow path. The longest flow path is one of major watershed parameters (Huang and Lee, 2016) used by many hydrologic models including the Hydrologic Engineering Center's Hydrologic Modeling System (HEC-HMS) (Feldman, 2000), the Soil and Water Assessment Tool (SWAT) (Arnold et al., 1998), the Storm Water Management Model (SWMM) (Rossman and Huber, 2016), and the Topography Model (TOPMODEL) (Beven and Kirkby, 1979) to name a few. It is mainly used to calculate the time of concentration and the lag time for hydrologic analysis (Maidment and Djokic, 2000; Feldman, 2000; Olivera, 2001). The United States Geological Survey (USGS) uses it to analyze annual peak-flow data and create regression equations that estimate the magnitude and frequency of floods (Gotvald et al., 2009; Feaster et al., 2014; Williams-Sether, 2015).

A flow path is the hydrologic path or watercourse from one point to another in the watershed. The longest flow path represents the flow path from a headwater (typically the watershed divide) to the outlet that is longer than all other flow paths in the watershed. It can set-theoretically be defined as

$$\overrightarrow{\mathrm{LFP}} \in \left\{ \overrightarrow{\mathrm{FP}}_i : \left| \overrightarrow{\mathrm{FP}}_i \right| \geq \left| \overrightarrow{\mathrm{FP}}_j \right| \ \forall i \neq j \right\} \tag{1}$$

where $\overrightarrow{\mathrm{LFP}}$ is a longest flow path, and i and j are indices for flow paths ($\overrightarrow{\mathrm{FP}}$) in the watershed. There can be more than one longest flow path in some case depending on the topography.

A typical procedure to determine the longest flow path requires the Digital Elevation Model (DEM) and involves a Geographic Information System (GIS) (Smith, 1995). To the best of the author's knowledge and based on an extensive literature review, Smith (1995) introduced the original longest flow path algorithm for GIS. This algorithm calculates two flow length rasters for an outlet cell using the DEM and adds both rasters to determine the longest flow path raster (Smith, 1995; Olivera and Maidment, 1998). The Arc Hydro toolbox (Maidment, 2002) for ArcGIS Pro (Esri, 2020a) provides a longest flow path tool that does not require the calculation of an upstream flow length raster. However, in spite of its important role in hydrologic studies, not much attention has been paid to the efficiency of the existing approaches.

This study improves upon the decades-old algorithm and introduces a new recursive algorithm and its iterative implementation that is more memory efficient. Section 2 reviews Smith's (1995) work, open source implementations of his algorithm for the Geographic Resources Analysis Support System (GRASS) GIS (Neteler et al., 2012), and the algorithm of the Arc Hydro Longest Flow Path tool for ArcGIS Pro. Section 3 elaborates on the new algorithms and designs benchmark experiments

for comparing the performance of the new module and the Arc Hydro tool. Benchmark results are analyzed and discussed in Sections 4 and 5, respectively, and Section 6 summarizes findings.

2. Background

2.1. Smith's flow-length-based approach

The length of the longest flow path for an outlet cell can be obtained by calculating the upstream flow length raster (Smith, 1995). However, finding the longest flow path itself also requires the calculation of the downstream flow length raster and the summation of both flow length rasters (Smith, 1995). The downstream flow length (DFL) is the flow length starting from the outlet. It can be written as

$$\mathrm{DFL}_i = \begin{cases} 0 & i = 0 \text{ at the outlet} \\ \mathrm{FL}_{i-1,i} + \mathrm{DFL}_{i-1} & i \geq 1 \end{cases} \tag{2}$$

where DFL_i is the downstream flow length from the outlet to cell i, $\mathrm{FL}_{i-1,i}$ is the flow length between cells $i-1$ and i, and i is 0 at the outlet and increases as we traverse in the upstream direction. The upstream flow length (UFL) is the flow length starting from a headwater. Similar to the DFL, the UFL can be written as

$$\mathrm{UFL}_i = \begin{cases} 0 & i = 0 \text{ at a headwater} \\ \mathrm{FL}_{i-1,i} + \max(\mathrm{UFL}_{i-1}) & i \geq 1 \end{cases} \tag{3}$$

where UFL_i is the upstream flow length from a headwater to cell i, $\mathrm{FL}_{i-1,i}$ is the flow length between cells $i-1$ and i, and i is 0 at a headwater and increases as we traverse in the downstream direction.

The DFL and UFL are maximum at the headwater on the longest flow path and the outlet, respectively, and both maximums are the same and referred to as the longest flow length (LFL). The longest flow path ($\overrightarrow{\mathrm{LFP}}$) is the path defined by all cells with a value equal to the LFL defined by

$$\mathrm{LFL} = \max(\mathbf{DFL} + \mathbf{UFL}) \tag{4}$$

where **DFL** and **UFL** represent the downstream and upstream flow length rasters, respectively (boldface notations for sets of cells and features such as raster and vector maps). However, it would not be possible to simply find cells with the exact LFL value because of rounding errors and, in practice, a small error tolerance is needed to find all cells as follows:

$$\mathrm{LFL} - \varepsilon \leq c_i \leq \mathrm{LFL} + \varepsilon \tag{5}$$

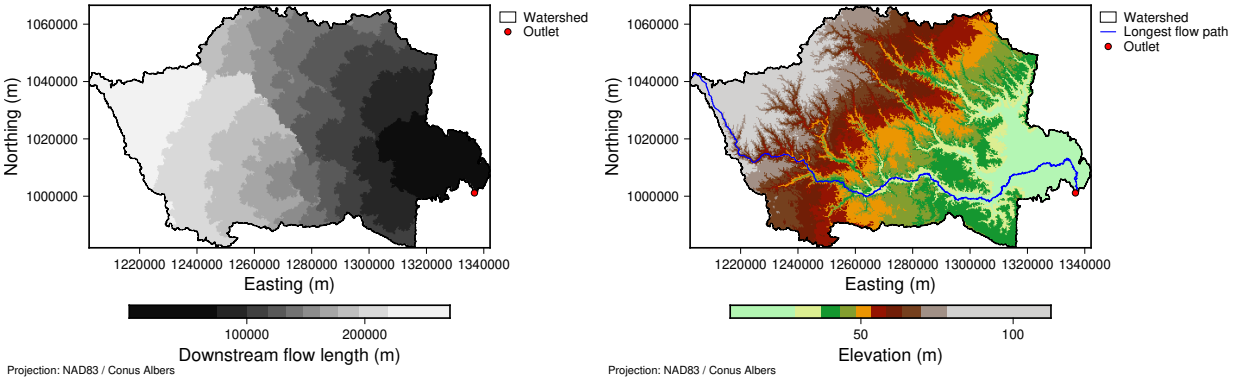
where c_i is the value of cell i and ε is a positive error tolerance which should be close to 0. Finally, the longest flow path can be determined by connecting these cells.

Figure 1 shows an example of the DFL and the longest flow path. In Figure 1a, the DFL is 0 (black) at the outlet and high (bright) at headwater cells. The UFL raster is not shown because

Table 1
GRASS GIS raster-based modules. All modules internally invoke other modules written in the C language.

Module	Release date	Language	Description
r.lfp/v.lfp	September 5, 2014	Python	Single outlet; Raster-to-vector issue; Deprecated by new r.lfp
lfp.sh	February 20, 2017	Bash	Multiple outlets; Raster-to-vector issue; Not part of GRASS GIS
r.lfp	May 15, 2018	Python	Multiple outlets; No raster-to-vector issue
lfp2.sh	February 9, 2019	Bash	Multiple outlets; Simplified vector handling; Raster-to-vector issue; Not part of GRASS GIS

most cells outside the longest flow path have a low value (dark) and the cells along the longest flow path (bright) are hard to see because of the cell size. Adding the **DFL** and **UFL** rasters yields a summation raster (**DFL**+**UFL**), which is used to derive the longest flow path shown in Figure 1b.



(a) Downstream flow length. (b) Longest flow path.

Figure 1: Downstream flow length raster and longest flow path for the Satilla River watershed in Georgia.

2.2. GRASS GIS modules

Historically, GRASS GIS (Neteler et al., 2012) has supported a raster-based approach for calculating the longest flow path since the release of the r.lfp and v.lfp modules in 2014. Table 1 summarizes raster-based modules that the author developed for GRASS GIS. A critical problem in the raster-based approach can occur during a raster-to-vector conversion because the conversion process does not understand hydrology and simply reshapes a series of cells into a linear feature without considering flow directions. Depending on the conversion algorithm, a cluster of cells may be simplified and some cells can be missing from the vector output. This conversion issue is noted as a raster-to-vector issue in Table 1. Figure 2 shows an example of this raster-to-vector conversion issue. The vector longest flow path initially follows flow directions faithfully, but it takes a short cut skipping one cell in the middle. This conversion issue violates the hydrologic validity of the vector result.

The deprecated r.lfp was responsible for computing the raster longest flow path while v.lfp

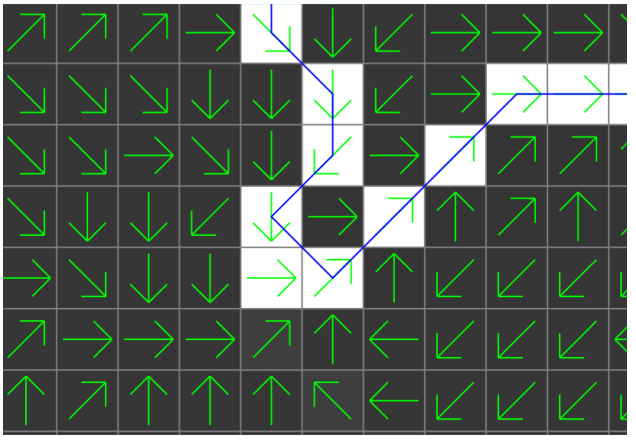


Figure 2: Hydrologically valid raster (white cells), but invalid vector (blue line) longest flow paths in Satilla River, Georgia. The green arrows indicate flow directions.

would convert it to vector. Only one outlet was supported at a time, which made these modules inefficient for processing multiple outlets because of heavy disk input/output (I/O). These modules were deprecated by the new r.lfp. The new r.lfp module addressed the raster-to-vector conversion issue by tracing the flow direction raster from headwater cells. This module directly generates the vector longest flow path and eliminated need for v.lfp. It also supports multiple outlets in one run and can save substantial computational time. The major difference between this module and the other modules is a lack of the upstream flow length raster (**UFL**) in the algorithm. Only the downstream flow length raster (**DFL**) is computed for each outlet and the longest flow path is traced following the flow direction raster (**FDR**) starting from headwater cells.

The lfp.sh module was developed to calculate longest flow paths for a lot of watersheds. This module tries to save computational time by calculating the upstream and downstream flow length rasters only once for the entire computational region encompassing all watersheds. The module can handle multiple outlets on the same stream network, but its algorithm is rather complicated because of heavy vector handling. However, it still converts raster to vector, which means that it inherits the same hydrologic issue from its deprecated siblings. The lfp2.sh module was an attempt to get rid of complicated vector handling from lfp.sh. These two modules were not released as part of GRASS GIS because they were written in Bash (a non-official scripting language for GRASS GIS) (Free Software Foundation, 2014). Both modules calculate and sum the downstream and upstream flow length rasters at different stages and extract cells whose value is equal to the longest flow length. The lfp.sh module creates the summation raster (**DUFL**) only once outside the main loop to minimize disk I/O while the lfp2.sh module creates the upstream flow length raster (**UFL**) once outside the loop, but it calculates the downstream flow length raster (**DFL**) for each outlet to reduce vector edits.

All of these existing modules only support accumulated watersheds. For example, if there is one outlet upstream of another, the longest flow path for the downstream outlet will be delineated

for the entire watershed, not for the downstream subwatershed, leaving an overlap between the two longest flow paths. This way of calculating multiple longest flow paths is not typically used for hydrologic analysis, so these modules were not used for this study. However, based on an independent test, these modules underperformed the new module introduced in Section 3 and were found to be inefficient in terms of computational time.

2.3. Arc Hydro tool

The Arc Hydro toolbox for ArcGIS Pro provides the Longest Flow Path tool written in Python (van Rossum and Drake, 2009). Algorithm 1 shows its pseudocode. This tool only calculates the downstream flow length raster (**DFL**) for each subwatershed using the built-in flow length tool and finds headwater cells with the longest flow length from the boundary cells. It then traces down the flow direction raster starting from the headwater cells. The tool implicitly assumes that longest flow paths always start from the subwatershed divide because it searches only the boundary zone for headwater cells.

Require: W ▷ Subwatershed polygons
Require: FDR ▷ Flow direction raster
$n \leftarrow |\mathbf{W}|$ ▷ Number of subwatersheds in **W**
for $i \leftarrow 1$ **to** n **do**
  $W_i \leftarrow$ Subwatershed i from **W**
  $\mathbf{FDR}_i \leftarrow$ Clip **FDR** to W_i
  Z $\leftarrow 1$ **if** $\mathbf{FDR}_i$ is not NoData **else** NoData
  S $\leftarrow$ Shrink **Z** by one cell
  N $\leftarrow 1$ **if S** is NoData **else** 0
  B $\leftarrow$ Bitwise AND of **N** and **Z**
  B $\leftarrow$ NoData **if B** is 0 **else Z** ▷ Subwatershed boundary cells
  $\mathbf{DFL}_i \leftarrow$ Calculate the downstream flow length raster for subwatershed W_i using **FDR**
  M $\leftarrow$ Calculate the zonal statistics maximum of $\mathbf{DFL}_i$ by zone **B** ▷ Boundary cells only
  MN $\leftarrow 0$ **if** $\mathbf{DFL}_i$ is **M else** 1 ▷ Boundary cells only
  C $\leftarrow$ **M if MN** is 0 else NoData ▷ Boundary cells only
  O $\leftarrow$ Convert the **C** raster to points
  $\overrightarrow{L} \leftarrow$ Trace **FDR** starting from the headwater points in **O**
  if $\left|\overrightarrow{L}\right|$ is equal to the length of intersection of $\overrightarrow{L}$ and W_i **then**
    $\overrightarrow{L} \leftarrow$ Extend the downstream end of $\overrightarrow{L}$ to the boundary of W_i
      ▷ Regardless of the flow direction at O_i? What about the upstream end?
  end if
  $\overrightarrow{\mathrm{LFP}}_i \leftarrow \overrightarrow{L}$
end for

Algorithm 1: Pseudocode for the Longest Flow Path tool in Arc Hydro for ArcGIS Pro. Created based on Python code analysis.

3. Methods

3.1. Recursive definition of the longest flow path

The I/O of raster data often takes relatively a long computational time compared to actual raster manipulation and becomes a bottleneck (Qin et al., 2014). It would be ideal to not use raster data at all, but the most basic input is the flow direction raster map and it is not feasible to implement an algorithm that is purely vector-based. In this section, we approach the problem of calculating the longest flow path from a different perspective and try to avoid calculating the upstream and downstream flow length rasters and summing these two rasters all together.

In fact, the longest flow path defined by Eq. (1) can recursively be redefined as

$$\overrightarrow{\mathrm{LFP}}_i \in \begin{cases} \left\{ \overrightarrow{\mathrm{LFP}}_j + \overrightarrow{L}_{ji} : \left| \overrightarrow{\mathrm{LFP}}_j + \overrightarrow{L}_{ji} \right| \geq \left| \overrightarrow{\mathrm{LFP}}_k + \overrightarrow{L}_{ki} \right| \ \forall j,k \in \mathbf{UP}, j \neq k \right\} & \text{if } \mathbf{UP} \neq \emptyset \\ \left\{ \vec{0} \right\} & \text{otherwise} \end{cases} \tag{6}$$

where $\overrightarrow{\mathrm{LFP}}_i$ is a longest flow path for cell i, $\overrightarrow{L}_{ji}$ is the flow path from cells j to i, and $\mathbf{UP}$ is the set of up to eight upstream neighbor cells flowing into cell i. In other words, if there are no upstream neighbor cells, we have reached a headwater cell and the longest flow path at the headwater cell will be a zero vector ($\vec{0}$). If there are upstream neighbor cells, a longest flow path for the current cell can be determined by connecting its upstream neighbor cell's longest flow path and the path from the neighbor to the current cell. Since there can be maximum eight upstream neighbor cells, one condition has to be satisfied that the combined path of the upstream cell's longest flow path and the path from the neighbor to the current cell has to be longer than or equal to the length of the combined path of any other neighbor cells.

This recursive definition of the longest flow path looks simple, but it is not straightforward to implement it in a computationally efficient way because the number of possible upstream cells can grow exponentially depending on the size, topography, and shape of the watershed. Assuming that each cell receives flow from $n \leq 8$ upstream neighbor cells on average, we can estimate the number of cells to visit v starting from one outlet towards upstream in s steps using the following equation:

$$v = \sum_{i=0}^{s} n^i \tag{7}$$

where $s = 0$ means that we are at the outlet. By plugging the number of cells on an $M \times N$ map into v and backcalculating s for $v = M \times N$, we can estimate how many steps we can traverse the stream upwards before we have to visit all the cells on the map. For example, on a 1000×1000 map with an average upstream neighbor cells $n = 2$, we would have to visit all the cells (a million) while being able to traverse the stream only 19 steps upwards. For $n = 3$, it would take 13 steps upwards until we have to visit all the million cells. In the worse case, when $n = 8$, it would take only seven steps. Of course, not all cells have the same number of upstream neighbors and calculating the actual number of steps would be more complicated. However, from these examples, we learned that

it would not be any efficient computationally compared to the flow-length-based approach.

To address the problem of an exponentially growing number of cells to visit in each step, we need to devise some constraints that help us filter out non-candidate neighbor cells as early as possible. In the following sections, we will discuss the relationship between the longest flow path and area of watersheds, and develop a strategy that can filter out neighbor cells whose potential longest flow length is shorter than others.

3.2. Branching strategy

Hack (1957) proposed a power-law relationship between the main channel length and area of watersheds. This empirical relationship is usually referred to as Hack's law (Rigon et al., 1996) and can be written as

$$L = cA^h \tag{8}$$

where L is the distance from the outlet to the drainage divide along the stream channel, A is the watershed area, and c and h are Hack's coefficient and exponent, respectively.

Many researchers have studied the significance of h (Sassolas-Serrayet et al., 2018) and it is generally believed to be slightly below 0.6 and greater than 0.5 (Rigon et al., 1996). Mesa and Gupta (1987) derived a theoretical equation of h as a function of the watershed's magnitude n as follows:

$$h(n) = \frac{1}{2}\left[\frac{\pi + (\pi/n)^{1/2}}{\pi - 1/n}\right]. \tag{9}$$

For $n = 1$ and 1000, h is 1.147 and 0.509. Eq. (9) implies that h tends to converge to 0.5 as n approaches infinity and the minimum h value is 0.5. Sassolas-Serrayet et al. (2018) analyzed Hack's coefficient c using around 22,000 watersheds in Bhutan and found c between 1.5 and 2.5 for $h \approx 0.5$, which is consistent with previous studies (Sassolas-Serrayet et al., 2018). They proposed an equation for c using the Gravelius compactness coefficient (GC) (Gravelius, 1914) as follows:

$$c = \frac{1}{2}\mathrm{GC}\sqrt{\pi} + \frac{1}{4}\sqrt{\mathrm{GC}^2\pi - 4} \tag{10}$$

where GC is the ratio of the watershed perimeter to the circumference of the circle whose area is equivalent to the watershed area. GC can be written as

$$\mathrm{GC} = \frac{P}{2\sqrt{\pi A}} \tag{11}$$

where P and A are the watershed perimeter and area, respectively. GC becomes 1 for perfectly circular watersheds, imaginary of course, and starts growing as a watershed is being elongated (Sassolas-Serrayet et al., 2018). The minimum GC allowed in Eq. (10) is $\sqrt{4/\pi}$ yielding 1 as the minimum c. The coefficient c is not unitless and has a dimension of L^{1-2h} where L denotes a length unit. However, as h approaches 0.5, $[c]$ tends to be $\mathsf{L}^{1-2\times0.5} = 1$ and c becomes practically unitless.

One consideration we have to take into account for this research is that the longest flow path

does not always follow the main channel. For upstream-most subwatersheds, the definition of the longest flow path may agree with that of the main channel in Hack's law in some cases. However, for downstream subwatersheds, the longest flow path starts from non-stream land surface as shallow surface runoff and enters the stream later. By definition, the longest flow path is almost always longer than the stream channel flowing through a subwatershed. For this reason, we cannot directly use Eq. (8) with typical c and h values. Since the recursive definition in Eq. (6) traverses the flow path from downstream towards upstream, we want to eliminate inflowing neighbor cells that cannot possibly generate a longest flow path by comparing them with their peers at each branching cell.

This paper proposes that, just like the main channel length (L) in Hack's law, the longest flow length (LFL) be also estimated using Eq. (8) with a slightly different interpretation. Since Hack's law involves the watershed area, the flow direction raster cannot be used alone in the new recursive algorithm. Instead, flow accumulation is computed using the flow direction raster and is used to estimate the subwatershed area at each cell. The basic idea for branching and eliminating neighbor cells is simple. Let's say there are two inflowing neighbor cells. We take their flow accumulation values and calculate their potential minimum and maximum LFLs. If the maximum LFL of one cell is shorter than the minimum LFL of the other cell, the former cannot generate a longest flow path because its theoretically longest LFL is still shorter than the other cell's theoretically shortest LFL. The former is discarded from further recursion, and the latter survives and is further being traversed and evaluated.

3.3. Longest and shortest longest flow lengths

We will address the longest longest flow length ($\mathrm{LFL}_{\max}$) first because estimating this distance is more intuitive and easier. Given the flow accumulation value (the number of upstream cells) at a cell, in what configuration would these upstream cells yield $\mathrm{LFL}_{\max}$? Travelling all these cells diagonally only without any horizontal or vertical moves will result in $\mathrm{LFL}_{\max}$. Therefore, $\mathrm{LFL}_{\max}$ can be written as

$$\mathrm{LFL}_{\max} = s \cdot \mathrm{FAC}\sqrt{2} \tag{12}$$

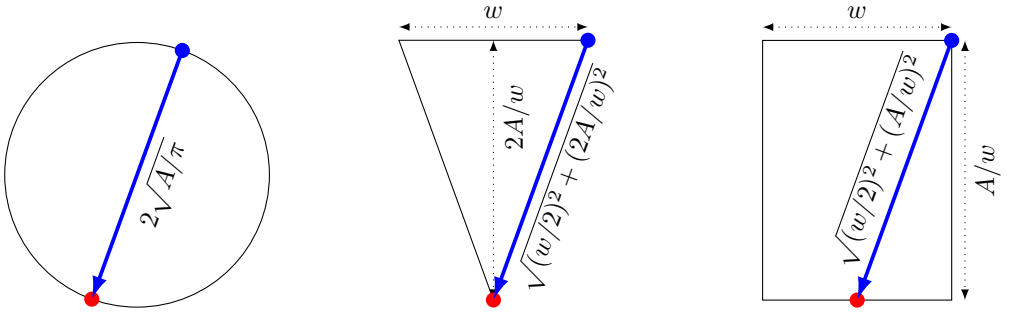
where s is the width or height (mostly the same in any GIS) of one cell and FAC is the flow accumulation value at a neighbor cell.

Now, we will use Eq. (8) to estimate the shortest longest flow length ($\mathrm{LFL}_{\min}$). Since we compare one cell's $\mathrm{LFL}_{\max}$ with another cell's $\mathrm{LFL}_{\min}$ and decide whether or not to eliminate one of these cells, it is important to be conservative in estimating $\mathrm{LFL}_{\min}$ because we do not want to discard those cells that would have yielded a longest flow path if they were not discarded accidentally. If $\mathrm{LFL}_{\min}$ is estimated to be too short, the recursive algorithm will not be able to filter out many non-candidate cells or not at all in the worse case as if there were no branching strategy in place. In contrast, if it is estimated to be too long, the algorithm will start getting rid of good candidate cells and can generate non-longest flow paths. Considering that the longest flow path is usually longer than the main channel and it is safer to underestimate $\mathrm{LFL}_{\min}$ not to miss cells that are perfectly

fine, the paper proposes the following equation for $\mathrm{LFL}_{\mathrm{min}}$:

$$\mathrm{LFL}_{\mathrm{min}} = s\sqrt{\mathrm{FAC}}. \tag{13}$$

Eq. (13) can be obtained by plugging $c = 1$, $h = 0.5$, and $A = s^2 \cdot \mathrm{FAC}$ into Eq. (8). A value of 1 for c is less than its minimum value $\sqrt{4/\pi}$ for Eq. (10) and a value of 0.5 for h is its minimum value for Eq. (9). It means that Eq. (13) is always shorter than Eq. (8) with the minimum c and h for Eqs. (10) and (9), respectively, and being conservative. Also, the unit of c is unitless because h is 0.5.



(a) Circular. (b) Isosceles Triangular. (c) Rectangular.

Figure 3: $\mathrm{LFL}_{\mathrm{min}}$ for different simplified watersheds. The solid black and thick blue lines represent the watershed boundary and shortest longest flow path, respectively. The blue and red dots indicate the furthest headwater and outlet points, respectively. The width and area of the watershed are referred to as w and A, respectively.

Figure 3 shows three different simplified shapes of watersheds and their shortest longest flow path (LFP) with its LFL, assuming that headwater points are located on the watershed boundary. In these simplified watersheds, the shortest LFP must be a straight line connecting the remotest point (blue dots) and the outlet (red dots). Any other straight paths will not be able to drain the furthest point and any non-straight paths will be longer than the shortest LFP. For hypothetical circular watersheds, $\mathrm{LFL} = 2\sqrt{A/\pi}$. For isosceles triangular and rectangular watersheds, by taking the derivative of the LFL with respect to w, we can obtain

$$\frac{\mathrm{dLFL}}{\mathrm{d}w} = \mathrm{LFL}^{-1}\left(\frac{w}{4} - 4A^2w^{-3}\right) \tag{14}$$

and

$$\frac{\mathrm{dLFL}}{\mathrm{d}w} = \mathrm{LFL}^{-1}\left(\frac{w}{4} - A^2w^{-3}\right), \tag{15}$$

respectively. By setting Eqs. (14) and (15) to 0, we found that $w = 2\sqrt{A}$ and $w = \sqrt{2A}$ minimize the LFLs of isosceles triangular and rectangular watersheds, respectively. Plugging these w values into the LFL equations in Figure 3 yields the shortest LFLs of $\sqrt{2A}$ and $\sqrt{A}$, respectively, for both types of watersheds. Since we use the FAC raster to calculate the watershed area, we can substitute $s^2 \cdot \mathrm{FAC}$ for A in the final LFL equations. Therefore, all three LFLs for the simplified watersheds

can be rewritten as $2s\sqrt{\mathrm{FAC}/\pi}$, $s\sqrt{2 \cdot \mathrm{FAC}}$, and $s\sqrt{\mathrm{FAC}}$, and we confirmed that all these lengths are longer than or equal to the shortest LFL defined in Eq. (13).

3.4. New GRASS GIS module

The recursive algorithm and its iterative variant take a flow direction raster and outlet points as input parameters. It was a design decision not to provide subwatershed polygons because delineating them is an additional step. However, since subwatershed polygons are not provided, these algorithms have no knowledge about the shape of subwatersheds. In addition, FAC values read from the flow accumulation raster are only accumulated. Without flow "subaccumulation" introduced in Algorithm 2, these algorithms would only generate longest flow paths for accumulated watersheds. The flow subaccumulation process takes each of input outlet points and subtracts its FAC value from the FAC value of its downstream cells. The final result of this process is a raster where individual subwatershed regions have their own flow accumulation computed independent of their upstream flow contribution. If the flow direction raster is clipped to each subwatershed, flow accumulation is computed, and resulting flow accumulation rasters are patched together for all subwatersheds, the patched raster will be identical to the flow subaccumulation raster. This process is computationally very light because it only traces down, not up, along the downstream portion of LFPs. Both recursive and iterative algorithms perform flow subaccumulation before starting the up-tracing process.

```
1:  procedure SUBACCUMULATE(FDR, FAC, O)
2:      n ← |O|                                          ▷ Number of outlets in O
3:      for i ← 1 to n do
4:          O_i ← Outlet i from O
5:          FAC_i ← FAC value at O_i
6:          for j ∈ {All downstream cell indices of O_i} do      ▷ Use FDR for down tracing
7:              O_j ← Outlet j from O
8:              FAC_j ← FAC value at O_j
9:              if FAC_j > FAC_i then                ▷ If this cell still needs to be processed
10:                 FAC_j ← FAC_j − FAC_i                        ▷ Replace FAC_j in FAC
11:             end if
12:         end for
13:     end for
14: end procedure
```

Algorithm 2: Pseudocode for the SUBACCUMULATE function for both recursive and iterative algorithms.

Algorithm 3 is used to find inflowing neighbor cells and calculate their attributes including the accumulation value, downstream length, and theoretical minimum and maximum LFLs, which will be used later in the main logic for up-tracing in both recursive and iterative algorithms.

Algorithms 4 and 5 present pseudocode for the main procedure and its subroutine, respectively, for implementing recursive LFP tracing. Algorithms 6 and 7 present pseudocode for the main pro-

procedure FINDUP(**FDR**, **FAC**, x, y, l, **UP**, **HL**)
  $s \leftarrow$ Horizontal/vertical cell size
  $d \leftarrow s\sqrt{2}$ ▷ Diagonal cell size
  UP $\leftarrow$ Find up to 8 inflowing neighbor cells using **FDR**
    ▷ Count those cells only whose accumulation is less than the **FAC** value at (x, y)
  $n \leftarrow |\mathbf{UP}|$ ▷ Number of inflowing neighbor cells in **UP**
  for $i \leftarrow 1$ **to** n **do**
    Store the coordinates of UP_i as its coordinates attribute ▷ $\mathrm{UP}_{i,\mathrm{coordinates}}$
    Read and store the **FAC** value at UP_i as its accumulation attribute ▷ $\mathrm{UP}_{i,\mathrm{accumulation}}$
    if UP_i is a diagonal neighbor **then**
      $L \leftarrow l + d$
    else
      $L \leftarrow l + s$
    end if
    Store L as its downstream length attribute ▷ $\mathrm{UP}_{i,\mathrm{downlength}}$
    Store $L + s \times \sqrt{\mathrm{UP}_{i,\mathrm{accumulation}}}$ as its minimum length attribute ▷ $\mathrm{UP}_{i,\mathrm{minlength}}$
    Store $L + d \times \mathrm{UP}_{i,\mathrm{accumulation}}$ as its maximum length attribute ▷ $\mathrm{UP}_{i,\mathrm{maxlength}}$
  end for
  if $n > 1$ **then** ▷ If there are multiple inflowing neighbor cells
    UP $\leftarrow$ Sort cells in **UP** by their maximum length in descending order
    UP $\leftarrow \left\{\mathrm{UP}_i : \sqrt{2} \times \mathrm{UP}_{i,\mathrm{accumulation}} \geq \sqrt{\mathrm{UP}_{1,\mathrm{accumulation}}} \text{ for } 1 \leq i \leq n\right\}$
    ▷ Discard UP_i whose theoretical max LFL is shorter than the theoretical min LFL of UP_1
  end if
  $m \leftarrow |\mathbf{HL}|$ ▷ Number of headwater cells found so far in **HL**
  if $m > 0$ **then** ▷ If any headwater cells were found before
    $n \leftarrow |\mathbf{UP}|$ ▷ Number of inflowing neighbor cells in **UP**
    UP $\leftarrow \{\mathrm{UP}_i : |\mathrm{UP}_{i,\mathrm{maxlength}}| \geq |\mathrm{HL}_j| \text{ for } 1 \leq i \leq n, 1 \leq j \leq m\}$
    ▷ Discard UP_i whose theoretical max LFL is shorter than the LFLs of all headwater cells
  end if
end procedure

Algorithm 3: Pseudocode for the FINDUP function for both recursive and iterative algorithms.

cedure and its subroutine, respectively, for implementing LFP tracing iteratively using a stack as its primary data structure for managing neighbor cells. The r.accumulate GRASS GIS module implements both recursive (with the `-r` flag) and iterative (default without the `-r` flag) LFP algorithms. This module is written in the C language (Kernighan and Ritchie, 1988). To differentiate between the two algorithms in the same GRASS GIS module, the recursive and iterative implementations will be referred to as r.accumulate-recursive and r.accumulate-iterative, respectively. The reason why the author implemented the iterative version is for stack safety in deep recursion. Any function or procedure calls use the stack memory to store data about the calls until those routines return to the caller (Reese, 2013). However, the size of this memory is limited and invoking a function recursively too many times can cause a stack overflow when the system runs out of stack memory (Reese, 2013). In fact, this stack overflow condition occurred in one of the experiments that will be

discussed in Section 4. By converting recursive calls into a last in first out (LIFO) data structure using the heap memory (a stack in the heap memory), the iterative version avoids stack overflows that can be caused by deep recursion. In the recursive version, one LFP is searched for first all the way to the upstream-most headwater cell of one neighbor cell and move to the next branch (depth-first search) while, in the iterative version, all neighbor cells are pushed to the stack first, move upstream by one cell, and repeat these two steps until we find an upstream-most headwater cell (breadth-first search). This difference in search strategies becomes important when the size of a subwatershed is large because depth-first search would require a lot of stack memory for excessive recursion.

Require: FDR ▷ Flow direction raster
Require: O ▷ Outlets
FAC ← Accumulate flow using **FDR** ▷ Accumulated cell counts
SUBACCUMULATE(**FDR**, **FAC**, **O**) ▷ Consider the subwatershed heierarchy
LFP ← ∅ ▷ LFP vector map
$n \leftarrow |\mathbf{O}|$ ▷ Number of outlets in **O**
for $i \leftarrow 1$ **to** n **do**
    (x, y) ← Coordinates of O_i
    HL ← ∅ ▷ Headwater set
    TRACEUP(**FDR**, **FAC**, x, y, 0, **HL**) ▷ Discard the return value
    if no headwater cells in **HL then**
        fatal error ▷ Failed to calculate $\overrightarrow{\mathrm{LFP}}_i$
    end if
    $\mathbf{LFP}_i \leftarrow \emptyset$ ▷ $\overrightarrow{\mathrm{LFP}}_i$ set
    $n \leftarrow |\mathbf{HL}|$ ▷ Number of headwater cells
    for $j \leftarrow 1$ **to** n **do**
        $\overrightarrow{\mathrm{LL}}_j$ ← Trace **FDR** starting from HL_j and create a vector line
        $\mathbf{LFP}_i \leftarrow \mathbf{LFP}_i \cup \left\{\overrightarrow{\mathrm{LL}}_j\right\}$ ▷ Store $\overrightarrow{\mathrm{LL}}_j$ as an $\overrightarrow{\mathrm{LFP}}_i$
    end for
    LFP ← **LFP** ∪ $\mathbf{LFP}_i$ ▷ Add $\mathbf{LFP}_i$ to **LFP**
end for

Algorithm 4: Pseudocode for r.accumulate-recursive.

3.5. Benchmark experiments

The states of Georgia and Texas in the United States were selected for benchmark experiments. The 1 arcsecond (approximately 30 m) National Elevation Dataset (NED) tiles were downloaded from USGS (2020), patched, and clipped to the state boundaries of Georgia and Texas. A hundred random outlet points were generated for each state once and the same set of outlets was used for all the experiments for the same state.

There are many different factors affecting the performance of software including system architectures, central processing units (CPUs), clock speeds, the sizes of cache and random-access memory (RAM), drive types, operating systems (OSs), compilers, software implementations, etc. (Lee et al.,

```
1: function TraceUp(FDR, FAC, x, y, l, HL)
2:     if FAC value at (x, y) = 1 then                          ▷ If a headwater cell is found
3:         return true                                          ▷ Tracing was successful; stop recursion
4:     end if
5:     UP ← ∅                                                   ▷ Placeholder for inflowing neighbor cells' attributes
                                                                ▷ Attributes include coordinates, accumulation, downlength, and maxlength
6:     FindUp(FDR, FAC, x, y, 0, UP, HL)                        ▷ Find inflowing neighbor cells and attributes
7:     n ← |UP|                                                 ▷ Number of inflowing neighbor cells in UP
8:     if n = 0 then
9:         return true                                          ▷ Tracing was successful; stop recursion
10:    end if
11:    for i ← 1 to n do
12:        (x_i, y_i) ← Coordinates of upstream cell i
13:        if TraceUp(FDR, FAC, x_i, y_i, UP_{i,downlength}, HL) = true then
                                                                ▷ If tracing was successful
14:            if |HL| = 0 ∨ UP_{i,downlength} = HL_{1,downlength} then
                                                                ▷ If no headwater cells in HL or ties are found
15:                HL ← HL ∪ {UP_i}                             ▷ Add UP_i to HL
16:            else if UP_{i,downlength} > HL_{1,downlength} then
                                                                ▷ If UP_{i,downlength} is longer than existing downstream lengths
17:                HL ← {UP_i}                                  ▷ Leave UP_i only whose downstream length is the longest
18:            end if
19:        end if
20:    end for                                                  ▷ Repeast tracing from the next upstream cell
21:    return false                                             ▷ Move one cell upstream; keep tracing recursively
22: end function
```

Algorithm 5: Pseudocode for the TraceUp function for r.accumulate-recursive.

2016; Micheloni and Olivo, 2017; Wang et al., 2019). However, because of limited computational resources, the main purpose of the benchmark experiments in this study is to see how the disk type and memory size impact the performance of the recursive (r.accumulate-recursive) and iterative (r.accumulate-iterative) versions of the GRASS GIS module. The author used GRASS GIS on two Linux systems with different configurations including one with a lower clock speed, a hard disk drive (HDD), and more memory, and the other with a higher clock speed, a solid-state drive (SSD), and less memory. He also used the Longest Flow Path tool from the Arc Hydro toolbox for ArcGIS Pro on a Windows system for benchmarking with commercial software. The Windows system is equipped with the most recent CPU with the highest clock speed among the three systems and an SSD. Since the Windows system did not have access to GRASS GIS (administrative rights are required to run the GRASS GIS installer), only the ArcGIS Pro tool was run on this system. Similarly, ArcGIS Pro is not available for Linux, so only GRASS GIS was used on the Linux systems.

Table 2 shows the system specifications used for experiments including OSs, CPUs, the amounts of RAM, the sizes of swap partitions, and software versions. Since the Longest Flow Path tool in Arc

Require: FDR ▷ Flow direction raster
Require: O ▷ Outlets
1: **FAC** ← Accumulate flow using **FDR** ▷ Accumulated cell counts
2: SUBACCUMULATE(**FDR**, **FAC**, **O**) ▷ Consider the subwatershed heierarchy
3: **LFP** ← ∅ ▷ LFP vector map
4: $n \leftarrow |\mathbf{O}|$ ▷ Number of outlets in **O**
5: **for** $i \leftarrow 1$ **to** n **do**
6: $(x, y) \leftarrow$ Coordinates of $\mathbf{O}_i$
7: **LL** ← ∅ ▷ Line set
8: TRACEUP(**FDR**, **FAC**, x, y, **LL**)
9: **if** no lines in **LL then**
10: **fatal error** ▷ Failed to calculate $\overrightarrow{\mathrm{LFP}}_i$
11: **end if**
12: $\mathbf{LFP}_i \leftarrow \emptyset$ ▷ $\overrightarrow{\mathrm{LFP}}_i$ set
13: $m \leftarrow |\mathbf{LL}|$ ▷ Number of lines in **LL**
14: **for** $j \leftarrow 1$ **to** m **do**
15: $\mathbf{LFP}_i \leftarrow \mathbf{LFP}_i \cup \{\overrightarrow{\mathrm{LL}}_j\}$ ▷ Store $\overrightarrow{\mathrm{LL}}_j$ as an $\overrightarrow{\mathrm{LFP}}_i$
16: **end for**
17: $\mathbf{LFP} \leftarrow \mathbf{LFP} \cup \mathbf{LFP}_i$ ▷ Add $\mathbf{LFP}_i$ to **LFP**
18: **end for**

Algorithm 6: Pseudocode for r.accumulate-iterative.

Table 2
System specifications for experiments. MB and GB are 1,000,000 B and 1,000,000,000 B, respectively. The system architecture is all x86_64 (64 bit architecture). Systems 1, 2, and 3 are referred to as Linux SSD, Linux HDD, and Windows, respectively. The Linux SSD and HDD systems used the default stack size of 8 MiB, which is $8 \times 1024 \times 1024$ B. *: 3200 MB/s sequential read, 1800 MB/s sequential write. †: 7200 rev/min. ‡: 3200 MB/s sequential read, 2400 MB/s sequential write.

OS	CPU	Disk type	RAM	Swap	Software version
[1]Linux 4.14.39	Intel® Core™ i5-7300U @ 2.60GHz	SSD*	16 GB	20 GB	GRASS GIS 7.9.dev
[2]Linux 4.4.172	Intel® Xeon® E5620 @ 2.40GHz	HDD†	48 GB	30 GB	GRASS GIS 7.9.dev
[3]Windows 10	Intel® Core™ i7-8700 @ 3.20GHz	SSD‡	32 GB	4864 MB	ArcGIS Pro 2.5.1
					Arc Hydro Pro 2.0.165

Hydro Pro accepts subwatersheds instead of outlets, the Watershed tool built in ArcGIS Pro was used to delineate 100 subwatershed raster maps first, which were then converted to polygon vector maps using the Raster To Polygon tool before these experiments were conducted. The Longest Flow Path tool in Arc Hydro Pro will be referred to as the Arc Hydro tool or simply Arc Hydro hereafter because its name "Longest Flow Path" is too generic.

Two different experiments were conducted including batch and non-batch mode experiments. The batch mode experiment assumes a scenario where we want to compute longest flow paths for multiple outlets at once in one run of each program. It is expected to increase disk I/O compared to the non-batch mode experiment explained later because a program needs to read in a flow direction

```
1: procedure TraceUp(FDR, FAC, x, y, LL)
2:     if FAC value at (x, y) = 1 then                          ▷ If a headwater cell is found
3:         return                                               ▷ No LFP exists for the headwater cell
4:     end if
5:     UP ← ∅                                                   ▷ Placeholder for inflowing neighbor cells' attributes
                                                                ▷ Attributes include coordinates, accumulation, downlength, and maxlength
6:     HL ← ∅                                                   ▷ Headwater set
7:     FindUp(FDR, FAC, x, y, 0, UP, HL)                        ▷ Find inflowing neighbor cells and attributes
8:     if no cells in UP then
9:         return                                               ▷ Tracing was successful; stop tracing
10:    end if
11:    US ← Push all inflowing neighbor cells in UP into US     ▷ |US| ← |UP|
                                                                ▷ Last in first out (LIFO) stack space for upstream cells
12:    while upstream cells in US do
13:        u ← Pop one inflowing neighbor cell from US          ▷ |US| ← |US| − 1
14:        (x_u, y_u) ← u_coordinates
15:        l_u ← u_downlength
16:        UP ← ∅
17:        FindUp(FDR, FAC, x_u, y_u, l_u, UP, HL)
18:        if upstream cells in UP then
19:            US ← Push all inflowing neighbor cells in UP into US   ▷ |US| ← |US| + |UP|
20:        else                                                 ▷ No inflowing cells; found a headwater cell
21:            if |HL| = 0 ∨ l_u = HL_{1,downlength} then        ▷ If no headwater cells in HL or ties are found
22:                HL ← HL ∪ {u}                                ▷ Add u to HL
23:            else if l_u > HL_{1,downlength} then             ▷ If l_u is longer than existing downstream lengths
24:                HL ← {u}                                     ▷ Leave u only whose downstream length is the longest
25:            end if
26:        end if
27:    end while
28:    n ← |HL|                                                 ▷ Number of headwater cells
29:    for i ← 1 to n do
30:        LL_i ← Trace FDR starting from HL_i and create a vector line
31:        LL ← LL ∪ {LL_i}                                     ▷ Add LL_i to LL
32:    end for
33: end procedure
```

Algorithm 7: Pseudocode for the TraceUp function for r.accumulate-iterative.

raster and calculate flow accumulation and subaccumulation for the entire state, not just for the area of interest. This experiment tests how fast and efficient each program is at computing multiple longest flow paths once after reading input data. Each program is run 30 times independently for each state. The results are used to calculate the average and standard deviation of run times. In contrast, the non-batch mode experiment runs programs for each outlet (GRASS GIS module) or subwatershed (Arc Hydro tool) at a time independently in a separate process. Since the Arc Hydro tool uses subwatershed polygons as a mask, the GRASS GIS module also used the same polygons to

restrict the computational region to be fair. In this experiment, the burden of disk I/O is expected to be lower than in the batch mode experiment because programs do not have to read in any data outside the area of interest. This experiment allows to analyze the performance sensitivity of the program to the subwatershed area. Each program is run once for each outlet independently; hence there are 100 runs for each state because there are 100 outlets (or subwatersheds for the Arc Hydro tool).

4. Results

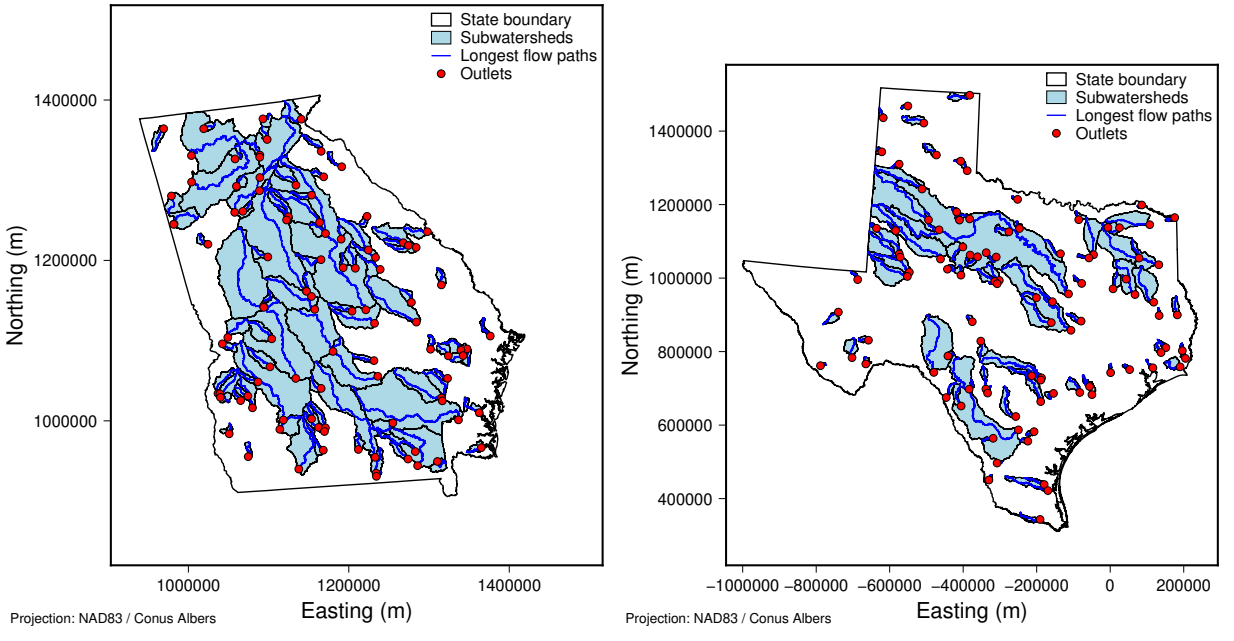
4.1. Validation of longest flow paths

First, we need to validate Eqs. (12) and (13) proposed for the longest and shortest longest flow lengths, respectively. For both states, the lengths of all calculated longest flow paths were strictly between $\mathrm{LFL}_{\min}$ and $\mathrm{LFL}_{\max}$ estimated using the equations. The Gravelius compactness coefficient (GC) calculated using Eq. (11) varied from 1.64 to 5.49. Plugging the GC into Eq. (11) yielded Hack's coefficient c of 1.99 to 7.25, which is greater than the assumed $c = 1$ in Eq. (13) for the shortest longest flow length.

For the state of Georgia, all the three programs have successfully completed 30 batch runs and 100 non-batch runs. However, for the state of Texas, r.accumulate-recursive failed to complete all 30 batch runs and one non-batch run on both Linux SSD and HDD systems. In fact, the 30 batch runs failed because of the same subwatershed for which the one failed non-batch run was not successful at calculating the longest flow path with a segmentation fault error. Except for these failed cases, r.accumulate-recursive and r.accumulate-iterative generated identical longest flow paths with a maximum error tolerance of 1×10^{-8} m due to round-off errors in both experiments. Figure 4 shows the longest flow paths generated by r.accumulate-iterative.

Figures 5 and 6 highlight major differences between the results from r.accumulate-iterative and Arc Hydro. Because of the assumption made in the Arc Hydro tool where any longest flow paths must start from the drainage divide of a subwatershed and cannot start from its interior, it is clear from Figure 5 that the Arc Hydro tool can generate inferior longest flow paths that are not truly the longest in some cases. Thirteen and six longest flow paths in Georgia and Texas, respectively, started from the interior of their subwatersheds and the Arc Hydro tool generated not-so longest flow paths that are shorter than those generated by r.accumulate-iterative. The probabilities of generating invalid longest flow paths are 13 % and 6 % out of 100 longest flow paths for Georgia and Texas, respectively. Figure 5a shows a relative error in length of 1.3 %, but Figure 5b presents a more extreme case where the error is 27.4 %.

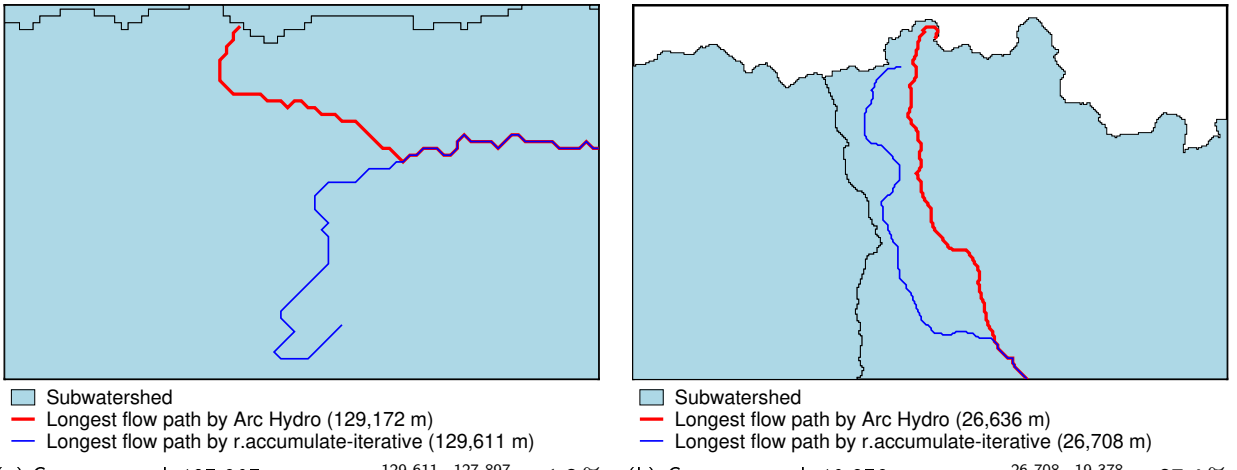
Another feature of the Arc Hydro tool is an extension of the end node of the longest flow path, but it simply extends the last node into the same direction as the last segment, not in the flow direction at the outlet cell as shown in Figure 6a. Last, from an independent run using a different dataset (Neteler and Mitasova, 2008), the Arc Hydro tool generated a shorter longest flow path for a subwatershed where the true longest flow path actually starts from a boundary cell, which means that this subwatershed satisfies the tool's drainage divide assumption, but the tool still failed to



(a) Georgia. (b) Texas.

Figure 4: Randomly generated 100 subwatershed outlets and longest flow paths generated by r.accumulate-iterative.

find the longest flow path.

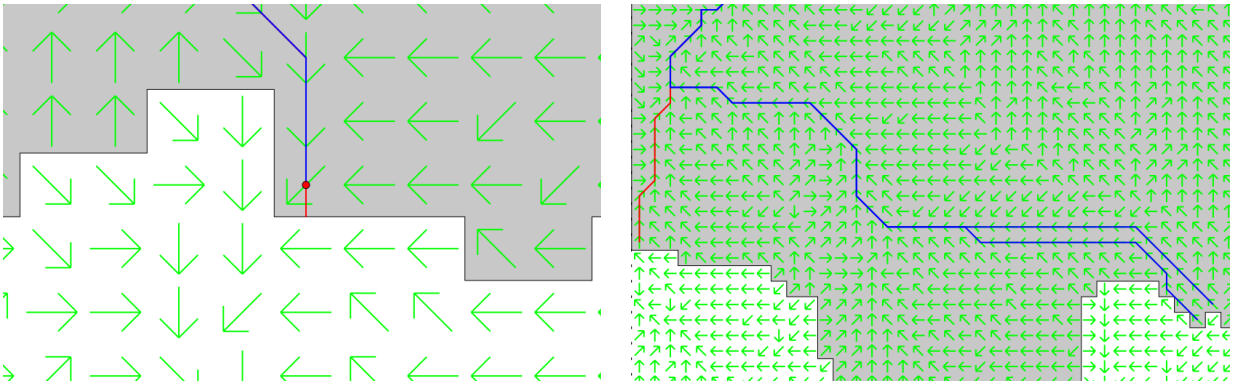


(a) Common path 127,897 m, error $= \frac{129{,}611-127{,}897}{129{,}611} = 1.3\,\%$. (b) Common path 19,378 m, error $= \frac{26{,}708-19{,}378}{26{,}708} = 27.4\,\%$.

Figure 5: Longest flow paths starting from the interior of subwatersheds.

4.2. Comparisons of elapsed times

Table 3 summarizes 30 elapsed times for processing all the 100 outlets in each state in the batch mode experiment. Overall, the Arc Hydro tool was about 22 to 49 times slower than the GRASS GIS module. For Georgia, the Linux SSD system was about two times faster than the Linux HDD



(a) Incorrectly extended (red line). The GRASS GIS module does not currently extend the end node of a longest flow path, but it is trivial to implement this feature correctly.

(b) Not the longest flow path (red) generated by an independent run using the GRASS GIS North Carolina sample dataset (Neteler and Mitasova, 2008).

Figure 6: Invalid longest flow paths generated by the Arc Hydro tool. The red and blue lines show the longest flow paths calculated by the Arc Hydro tool and r.accumulate-recursive, respectively. The gray polygon and green arrows represent the subwatershed and flow directions, respectively.

Table 3
Elapsed times with standard deviations in seconds from the batch mode experiment. *: r.accumulate-recursive failed to complete the Texas runs while processing the largest 51st subwatershed on both Linux systems.

State	Total area	System	r.accumulate-recursive	r.accumulate-iterative	Arc Hydro
Georgia	80,081 km^2	Linux SSD	19.845 ± 0.029	19.995 ± 0.028	
		Linux HDD	40.544 ± 0.901	40.813 ± 1.053	
		Windows			978.781 ± 4.744
Texas	187,198 km^2	Linux SSD	228.212 ± 9.527*	238.292 ± 14.475	
		Linux HDD	163.178 ± 2.563*	164.761 ± 2.493	
		Windows			5273.845 ± 33.113

system. On the same Linux SSD system, the worst run of r.accumulate-recursive outperformed the best run of r.accumulate-iterative. However, on the Linux HDD system, one r.accumulate-recursive run was the worst of both versions and the range of r.accumulate-recursive except the worst outlier was within the range of r.accumulate-iterative. For Texas, the Linux SSD system was about 1.4 times slower than the Linux HDD system. However, r.accumulate-recursive failed to complete on both systems, so comparisons between the two versions were not made.

Table 4 summarizes 100 elapsed times for each state in the non-batch mode experiment. The GRASS GIS module on both Linux SSD and HDD systems was about 34 to 97 times faster than the Arc Hydro tool on the Windows system with an SSD drive. Figure 7 shows the performance sensitivity of each program to the subwatershed area. For both states, the elapsed time of the Arc Hydro tool grew exponentially with increasing subwatershed areas. Its log-linear regression analysis yielded adjusted R^2 values (R^2_{adj}'s) of 0.9194 and 0.8681 for Georgia and Texas (one outlier excluded from Texas), respectively, with a p-value less than 2.2×10^{-16}. In contrast, the GRASS

Table 4
Elapsed times in seconds from the non-batch mode experiment. *: average elapsed time per km^2. †: r.accumulate-recursive failed to generate the longest flow path for the largest 51st subwatershed in Texas.

State	Total area	System	Statistic	r.accumulate-recursive	r.accumulate-iterative	Arc Hydro
Georgia	80,081 km^2	Linux SSD	Total	30.831	31.304	
			Average*	0.000,38	0.000,39	
		Linux HDD	Total	56.610	60.600	
			Average*	0.000,71	0.000,76	
		Windows	Total			2038.314
			Average*			0.025,45
Texas	187,198 km^2	Linux SSD	Total	64.130 †	64.629	
			Average*	0.000,34†	0.000,35	
		Linux HDD	Total	112.618 †	115.343	
			Average*	0.000,60†	0.000,62	
		Windows	Total			6191.002
			Average*			0.033,07

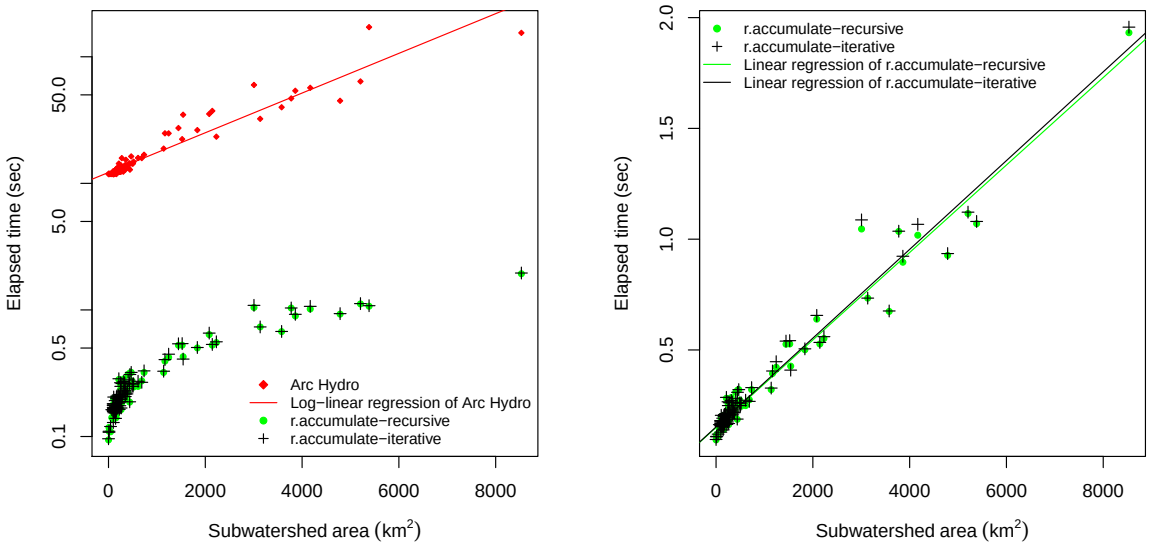
GIS module exhibited a linear growth ($R^2_{\text{adj}} \geq 0.9503$ with a p-value $< 2.2 \times 10^{-16}$) for both states.

5. Discussion

The GRASS GIS module on the Linux systems outperformed the Arc Hydro tool on Windows in both batch and non-batch mode experiments even though the Windows system has the best hardware specifications except the size of RAM (less than Linux HDD, but more than Linux SSD). The performance of the Arc Hydro tool deteriorated exponentially as the subwatershed size grew as opposed to the linear performance deterioration of the GRASS GIS module. This inefficiency of the Arc Hydro tool may be attributed to Esri's decision to implement the Arc Hydro toolbox using their Python application programming interface (API) instead of the .NET (Microsoft Corporation, 2020) API which is a more advanced approach to extending the capabilities of ArcGIS Pro (Esri, 2020b). However, it is still to be investigated if the lower performance was actually due to the additional Python layer between the frontend and its core engine or simply the tool's algorithm. Without a .NET implementation of the Arc Hydro tool, it would not be feasible to determine which was the case. The Arc Hydro tool not only was slower but also had three issues as follows:

1. longest flow paths starting from the interior of subwatersheds were not correctly calculated because of its boundary assumption where the tool assumes that longest flow paths only start from boundary cells (Figure 5),
2. for one non-benchmark run, it was not able to calculate the correct longest flow path even though the boundary assumption was not violated (Figure 6b), and
3. its end-node extension feature does not consider the flow direction at the outlet cell (Figure 6a).

In the batch mode experiment, the Linux SSD system was faster than its HDD variant except for



(a) Log-scale elapsed times versus subwatershed areas for Georgia from Linux SSD.

(b) Elapsed times versus subwatershed areas for Georgia from Linux SSD.

Figure 7: Elapsed times versus subwatershed areas from the non-batch mode experiment. The Arc Hydro results from the Windows system are plotted as a reference on (a). The ranges of R^2_{adj} for the Arc Hydro tool (log-linear regression) and GRASS GIS module (linear regression) are 0.8681 to 0.9194 and 0.9503 to 0.9691, respectively. The largest 51st subwatershed that failed in Texas (an outlier) was excluded from the regression analysis of the Arc Hydro results, but it was included in the regression analysis of the GRASS GIS results. The p-value for all regression lines is less than 2.2×10^{-16}.

the case of Texas. Since the Linux HDD system has a slower CPU and HDD with more RAM, we can see that the amount of memory played a more important role in processing the Texas subwatersheds. Texas has about total 2 million integer (4 bytes) cells in the flow direction raster which, when loaded into memory, occupies 8 GB. The GRASS GIS module additionally allocates 8 GB to calculate a flow accumulation raster. Holding these two rasters in memory alone requires 16 GB leaving only on-disk swap space for other tasks such as recursion and storing outputs on the Linux SSD system, which has 16 GB system memory. The faster CPU did not help when memory ran out and the slower swap space started kicking in. On the other hand, the Linux HDD system still has 32 GB of RAM for calculating longest flow paths. However, even with the remaining 32 GB memory, recursively traversing the largest 51st subwatershed in Texas consumed up the 8 MiB stack memory allocated by the OS kernel to r.accumulate-recursive. The area of this subwatershed is 47,077.62 km^2. Given the 30 m resolution of the elevation data, this area translates to total 52 million cells. Assuming that only 5 %—which is conservative compared to the size of the subwatershed—of the 52 million cells need to be recursively visited to find the longest flow path, the average stack size per visit (or per recursive call) would be only 3 bytes ($8 \times 1024^2/(0.05 \times 52 \times 1000^2)$), which is not enough to store state information about each recursive function call including variables. This lack of stack space caused a stack overflow. Since all the 100 subwatersheds were being processed together

in one process, the stack overflow resulted in no outputs at all. In contrast, with no recursive calls, r.accumulate-iterative was able to leverage the swap space on the Linux SSD system or the remaining 32 GB memory on the Linux HDD system. Still, the Linux SSD system was slower because a disk-based swap partition is slower than real memory.

In the non-batch mode experiment, the Linux SSD system consistently outperformed the Linux HDD system. The stack overflow issue did occur again with the largest Texas subwatershed, but unlike in the batch mode experiment, r.accumulate-recursive calculated the other 99 longest flow paths successfully because each subwatershed was processed separately. Also, for the same reason, the Linux SSD system ran out of memory only with this subwatershed and handled the other subwatersheds smoothly with enough memory. Therefore, no performance hit was observed on this system. Both Linux systems showed consistent performance per subwatershed area across the states regardless of the total area (0.000,34 s/km^2 to 0.000,39 s/km^2 for Linux SSD and 0.000,60 s/km^2 to 0.000,76 s/km^2 for Linux HDD as shown in Table 4). We can use this consistency in performance to predict computational times when running the GRASS GIS module.

In terms of the total computing time of the GRASS GIS module between the batch and non-batch mode experiments, the former was faster than the latter for Georgia and vice versa for Texas. For Georgia, it took more time to process individual subwatersheds separately. This added computational time in the non-batch mode experiment can be explained by an overhead expense of creating 100 separate output vector maps and repeating basic managerial tasks such as reading header files, adjusting computational regions, etc. However, for Texas, the non-batch mode experiment was faster because processing one subwatershed at a time is much more memory efficient than processing all the subwatersheds in one process especially when there are many large subwatersheds in the state.

In both experiments, r.accumulate-iterative (r.accumulate without the `-r` flag) turned out to be more robust with no failures and only slightly slower than its recursive sibling (r.accumulate with the `-r` flag). For this reason, it is recommended to use the r.accumulate module without the `-r` flag, which is the default. Currently, administrative rights are required to install GRASS GIS on Microsoft Windows, which is why the r.accumulate module was only tested on the Linux systems. However, GRASS GIS and the new module also support Microsoft Windows XP or newer and macOS 10.4.10 or newer. Future research includes implementation of the proposed algorithms for ArcGIS Pro and comparison of their performance with that of the GRASS GIS module.

6. Conclusions

In this paper, we studied the original longest flow path approach and briefly reviewed existing longest flow path programs. It was shown how hydrologically invalid longest flow paths could be generated by naively converting raster results to vector and a recursive definition of the longest flow path was introduced. Naturally, a new recursive algorithm was proposed based on the recursive definition for calculating the longest flow path using depth-first search and its iterative counterpart

algorithm was developed using breadth-first search. These algorithms use equations for the longest and shorted longest flow lengths that were derived from Hack's law. A branching strategy was devised to help the proposed algorithms filter out inferior neighbor cells and speed up traversal. Using randomly generated 100 outlets and subwatersheds each in Georgia and Texas, performance comparisons were made between the recursive and iterative versions of the r.accumulate GRASS GIS module that implement the proposed algorithms, and the Longest Flow Path tool in the Arc Hydro toolbox for ArcGIS Pro. An implicit assumption in the Arc Hydro tool was tested and shown to not work well in some case. With increasing subwatershed areas, the performance of the Arc Hydro tool exponentially deteriorated while that of the GRASS GIS module linearly declined. The GRASS GIS module outperformed the Arc Hydro tool, but the recursive implementation failed to process the largest subwatershed in Texas because of a stack overflow resulting from deep recursion. The iterative implementation was more robust in terms of memory usage and successfully calculated all the longest flow paths for both states. Since the iterative implementation was only slightly slower than its recursive counterpart, the iterative version is highly recommended over the other version. GRASS GIS and the new module support Microsoft Windows XP or newer with administrative rights, macOS 10.4.10 or newer, recent GNU/Linux or a UNIX variant. Future work includes implementation of the proposed algorithms for ArcGIS Pro.

Acknowledgements

The author thanks Dr. Daniel P. Ames for handling this manuscript and an anonymous reviewer for providing constructive comments. He also thanks the GRASS GIS community for their continued support for improving the historical modules and stimulating the development of the new software.

References

Arnold, J.G., Srinivasan, R., Muttiah, R.S., Williams, J.R., 1998. Large area hydrologic modelling and assessment, Part I: Model development. Journal of the American Water Resources Association 34, 73–89.

Beven, K.J., Kirkby, M.J., 1979. A physically based, variable contributing area model of basin hydrology. Hydrological Sciences Bulletin 24, 43–69.

Esri, 2020a. ArcGIS Pro 2.5. URL: https://www.esri.com/en-us/arcgis/products/arcgis-pro/overview.

Esri, 2020b. Proconcepts migrating to ArcGIS Pro. URL: https://github.com/esri/arcgis-pro-sdk/wiki/ProConcepts-Migrating-to-ArcGIS-Pro.

Feaster, T.D., Gotvald, A.J., Weaver, J.C., 2014. Methods for Estimating the Magnitude and Frequency of Floods for Urban and Small, Rural Streams in Georgia, South Carolina, and North Carolina, 2011. US Geological Survey Scientific Investigations Report 2014-5030. U.S. Department of the Interior, U.S. Geological Survey.

Feldman, A.D., 2000. Hydrologic Modeling System HEC-HMS Technical Reference Manual. U.S. Army Corps of Engineers, Institute for Water Resources, Hydrologic Engineering Center. Davis, CA. URL: http://www.hec.usace.army.mil/software/hec-hms/documentation.aspx.

Free Software Foundation, 2014. Bash 4.3. https://ftp.gnu.org/gnu/bash/bash-4.3.tar.gz. URL: https://ftp.gnu.org/gnu/bash/bash-4.3.tar.gz.

Gotvald, A.J., Feaster, T.D., Weaver, J.C., 2009. Magnitude and Frequency of Rural Floods in the Southeastern United States, 2006: Volume 1, Georgia. US Geological Survey Scientific Investigations Report 2009-5043. U.S. Department of the Interior, U.S. Geological Survey.

Gravelius, H., 1914. Grundriß der gesamten gewässerkunde, band 1: Flußkunde. Compendium of Hydrology I, 265–278.

Hack, J.T., 1957. Studies of longitudinal stream profiles in Virginia and Maryland. Geological Survey Professional Paper 294-B, 45–97. URL: https://pubs.usgs.gov/pp/0294b/report.pdf.

Huang, P.C., Lee, K.T., 2016. Distinctions of geomorphological properties caused by different flow-direction predictions from digital elevation models. International Journal of Geographical Information Science 30, 168–185. doi:doi:10.1080/13658816.2015.1079913.

Kernighan, B.W., Ritchie, D.M., 1988. The C Programming Language. 2nd ed., Prentice Hall, Inc. URL: https://www.cs.princeton.edu/~bwk/cbook.html.

Lee, S., Min, H., Yoon, S., 2016. Will solid-state drives accelerate your bioinformatics? In-depth profileing, performance analysis and beyond. Briefings in Bioinformaticss 17, 713–727. doi:doi:10.1093/bib/bbv073.

Maidment, D., Djokic, D. (Eds.), 2000. Hydrologic and Hydraulic Modeling Support with Geographic Information Systems. Esri Press.

Maidment, D.R. (Ed.), 2002. Arc Hydro: GIS for Water Resources. 3rd ed., Esri Press.

Mesa, O.J., Gupta, V.K., 1987. On the main channel length-area relationship for channel networks. Water Resources Research 23, 2119–2122. doi:doi:10.1029/WR023i011p02119.

Micheloni, R., Olivo, P., 2017. Solid-state drives (SSDs), in: Proceedings of the IEEE, IEEE. pp. 1586–1588. doi:doi:10.1109/JPROC.2017.2727228.

Microsoft Corporation, 2020. Introduction to .NET core. URL: https://docs.microsoft.com/en-us/dotnet/core/introduction.

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

Neteler, M., Mitasova, H., 2008. Open Source GIS: A GRASS GIS Approach. 3rd ed., Springer, New York. URL: https://grassbook.org/.

Olivera, F., 2001. Extracting hydrologic information from spatial data for HMS modeling. Journal of Hydrologic Engineering 6, 524–530. doi:doi:10.1061/(ASCE)1084-0699(2001)6:6(524).

Olivera, F., Maidment, D., 1998. Geographic information system use for hydrologic data development for design of highway drainage facilities. Transportation Research Record: Journal of the Transportation Research Board 1625, 131–138. doi:doi:10.3141/1625-17.

Qin, C.Z., Zhan, L.J., Zhu, A.X., 2014. How to apply the Geospatial Data Abstraction Library (GDAL) properly to parallel geospatial raster I/O? Transactions in GIS 18, 950–957.

Reese, R., 2013. Understanding and Using C Pointers. O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

Rigon, R., Rodriguez-Iturbe, I., Maritan, A., Giacometti, A., Tarboton, D.G., Rinaldo, A., 1996. On Hack's law. Water Resources Research 32, 3367–3374. doi:doi:10.1029/96WR02397.

Rossman, L.A., Huber, W.C., 2016. Storm Water Management Tool Reference Manual Volume I—Hydrology (Revised). National Risk Management Laboratory, Office of Research and Development, U.S. Environmental Protection Agency. Cincinnati, OH.

van Rossum, G., Drake, F.L., 2009. Python 3 Reference Manual. CreateSpace, Scotts Valley, CA.

Sassolas-Serrayet, T., Cattin, R., Ferry, M., 2018. The shape of watersheds. Nature Communications 9, 3791. doi:doi:10.1038/s41467-018-06210-4.

Smith, P.N.H., 1995. Hydrologic data development system. Transportation Research Record: Journal of the Transportation Research Board 1599, 118–127. doi:doi:10.3141/1599-15.

USGS, 2020. U.S. Geological Survey. One arc-second National Elevation Dataset (NED). ftp://rockyftp.cr.usgs.gov/vdelivery/Datasets/Staged/NED/1/IMG. Accessed in May 2020.

Wang, Y., Lee, V., Wei, G.Y., Brooks, D., 2019. Predicting new workload or CPU performance by analyzing public datasets. ACM Transactions on Architecture and Code Optimization 15, 53. doi:doi:10.1145/3284127.

Williams-Sether, T., 2015. Regional Regression Equations to Estimate Peak-Flow Frequency at Sites in North Dakota Using Data through 2009. US Geological Survey Scientific Investigations Report 2015-5096. U.S. Department of the Interior, U.S. Geological Survey.