Cite as Barnes, Lehman, Mulla. "Priority-Flood: An Optimal Depression-Filling and Watershed-Labeling Algorithm for Digital Elevation Models". Computers & Geosciences. Vol 62, Jan 2014, pp 117–127. doi: "10.1016/j.cageo.2013.04.024".

# Priority-Flood: An Optimal Depression-Filling and Watershed-Labeling Algorithm for Digital Elevation Models

Richard Barnes[a,*], Clarence Lehman[b], David Mulla[c]

[a] *Ecology, Evolution, & Behavior, University of Minnesota, USA*
[b] *College of Biological Sciences, University of Minnesota, USA*
[c] *Soil, Water, and Climate, University of Minnesota, USA*

**Abstract**

Depressions (or pits) are low areas within a digital elevation model that are surrounded by higher terrain, with no outlet to lower areas. Filling them so they are level, as fluid would fill them if the terrain were impermeable, is often necessary in preprocessing DEMs. The depression-filling algorithm presented here—called Priority-Flood—unifies and improves on the work of a number of previous authors who have published similar algorithms. The algorithm operates by flooding DEMs inwards from their edges using a priority queue to determine the next cell to be flooded. The resultant DEM has no depressions or digital dams: every cell is guaranteed to drain. The algorithm is optimal for both integer and floating-point data, working in $O(n)$ and $O(n \log_2 n)$ time, respectively. It is shown that by using a plain queue to fill depressions once they have been found, an $O(m \log_2 m)$ time-complexity can be achieved, where $m$ does not exceed the number of cells $n$. This is the lowest time complexity of any known floating-point depression-filling algorithm. In testing, this improved variation of the algorithm performed up to 37% faster than the original. Additionally, a parallel version of an older, but widely-used depression-filling algorithm required six parallel processors to achieve a run-time on par with what the newer algorithm's improved variation took on a single processor. The Priority-Flood Algorithm is simple to understand and implement: the included pseudocode is only 20 lines and the included `C++` reference implementation is under a hundred lines. The algorithm can work on irregular meshes as well as 4-, 6-, 8-, and $n$-connected grids. It can also be adapted to label watersheds and determine flow directions through either incremental elevation changes or depression carving. In the case of incremental elevation changes, the algorithm includes safety checks not present in prior works.

*Keywords:* pit filling; terrain analysis; hydrology; drainage network; modeling; GIS

## 1. Background

A digital elevation model (DEM) is a representation of terrain elevations above some common base level, usually stored as a rectangular array of floating-point or integer values. DEMs may be used to estimate a region's hydrologic and geomorphic properties, including soil moisture, terrain stability, erosive potential, rainfall retention, and stream power. Many algorithms for extracting these properties require (1) that every cell within a DEM must have a defined flow direction and (2) that by following flow directions from one cell

*Corresponding author, 321-222-7637. ORCID: 0000-0002-0204-6040
*Email addresses:* `rbarnes@umn.edu` (Richard Barnes), `lehman@umn.edu` (Clarence Lehman), `mulla003@umn.edu` (David Mulla)

to another, it is always possible to reach the edge of the DEM. These requirements are confounded by the presence of depressions and flats within the DEM.

Depressions (also known as pits) are inwardly-draining regions of the DEM which have no outlet. Sometimes representative of natural terrain, they may also result from technical issues in the DEM's collection and processing, such as from biased terrain reflectance or conversions from floating-point to integer precision.[Nardi et al., 2008] A depression may be resolved either by breaching its wall (e.g. Martz and Garbrecht [1998]), thus allowing it to drain to a nearby area of lower elevation, or by filling it.

DEMs have increased in resolution from thirty-plus meters in the recent past to the sub-meter resolutions becoming available today. Increasing resolution has led to increased data sizes: current data sets are on the order of gigabytes and increasing, with billions of data points. While computer processing and memory performance have increased appreciably during this time, legacy equipment and algorithms suited to manipulating smaller DEMs with coarser resolutions make processing these improved data sources costly, if not impossible. Therefore, improved algorithms are needed.

This paper presents an algorithm to resolve depressions by unifying and extending the work of several previous authors. Also presented are variants of this algorithm which can label watersheds and determine flow directions.

The general definition of the depression-filling problem was stated by Planchon and Darboux [2002]. Given a DEM $Z$, its depression-filled counterpart $W$ is defined by the following criteria:

1. Each cell of $W$ is greater than or equal to its corresponding cell in $Z$.
2. For each cell $c$ of $W$, there is a path that leads from $c$ to the boundary by moving downwards by an amount of at least $\epsilon$ between any two cells on the path, where $\epsilon$ may be zero. Such a path is referred to as an $\epsilon$-descending path.
3. $W$ is the lowest surface allowed by properties (1) and (2).

If $\epsilon = 0$, then the third criterion is easy to achieve; however, if $\epsilon \neq 0$, then special precautions must be taken, as described below.

The algorithm presented here is one of only two time-efficient algorithms for solving the depression-filling problem. Special cases of this algorithm have been described many times. These cases, their relations, and alternative algorithms are detailed below.

## 2. Alternative Algorithms

Arge et al. [2003] describes a specialized $O(n \log_2 n)$ procedure to perform watershed labeling, determine flow directions, and calculate flow accumulation on massive grids in situations where I/O must be minimized. Although parts of this procedure share sufficient similarities with Priority-Flood to be considered related, the combinations of algorithms used and the way in which they are specialized place it outside the scope of this paper. It is expected that the algorithm by Arge will run slower than that described here due to the greater overhead involved in explicit data management; however, this efficiency of memory may allow the algorithm by Arge to run better in situations where memory is limited.

There is also a widely-used algorithm for 8-connected grids by Planchon and Darboux [2002]. The algorithm works by flooding the entire DEM and then draining the edge cells. The entirety of the DEM is then repeatedly scanned to find the border of the drained and undrained regions; undrained cells adjacent to this border are then drained and increased in elevation by a small amount. The algorithm terminates when all cells have been drained.

Planchon and Darboux [2002] present two different implementations of their algorithm. The first is simple to implement, but inefficient, running in $O(n^{1.5})$ time. The second implementation runs in $O(n^{1.2})$ time during testing, but is much more complex, using 48 constants to define an iterative scan from multiple directions, a recursive upstream search with stack limiting to prevent overflows, and a quadruply-nested loop. The algorithm's design permits it to run in fixed memory. Unfortunately, the DEMs which require this are typically so large as to make running the Planchon–Darboux algorithm onerous.

| Year | Authors | Operation | Data Type | Connectedness | Time Complexity |
|---|---|---|---|---|---|
| 1989 | C. Ehlschlaeger | Flow Directions, Accumulation | Integer/Float* | 8, Any | $O(n^2)$* |
| 1991 | Vincent & Soille | Watershed Labels | Integer | Gridded, $n$-dim | $O(n)$ |
| 1992 | S. Beucher & F. Meyer | Watershed Labels | Integer | 4, 6, 8 | $O(n)$* |
| 1994 | F. Meyer | Watershed Labels | Integer* | 8* | $O(n)$* |
| 1994 | Soille & Gratin | Filling | Integer | 4, 6, 8 | $O(n)$* |
| 2006 | Wang & H. Liu | Filling | Floating* | 8* | $O(n \log_2 n)$ |
| 2009 | Y.H. Liu, Zhang, & Xu | Filling | Floating* | 8 | $O(n \log_2 k)^\dagger$ |
| 2010 | Metz, Mitasova, & Harmon | Flow Directions | Floating* | 8* | $O(n \log_2 n)$* |
| 2011 | Metz, Mitasova, & Harmon | Flow Directions | Floating* | "Gridded" | $O(n \log_2 n)$* |
| 2011 | N. Beucher & S. Beucher | Watershed Labels | Integer | 4, 6, 8 | $O(n)$* |
| 2012 | Magalhães et. al | Filling, Flow Directions, Accumulation | Integer | 8* | $O(n)$ |
| 2012 | Gomes et. al | Flow Directions, Accumulation | Integer | 8* | $O(n)$ |

**Table 1:** Summary of previous Priority-Flood variants. The table lists claims the authors have made. The (*) symbol indicates that the authors have not made a direct claim, so one has been inferred from their design choices. All the floating-point variants will also work on integer data, though the specified time complexities are then suboptimal. $^\dagger$Liu et al. [2009] claim a $O(8n \log_2 n)$ time complexity, but implement an $O(n \log_2 k)$ algorithm.

Fortunately, a fast, simple, and versatile alternative algorithm is available. Here, it is referred to as the Priority-Flood Algorithm. Later, it will be shown that this alternative algorithm runs significantly faster than that by Planchon and Darboux.

## 3. The Priority-Flood Algorithm

### *3.1. History*

In its most general form, the Priority-Flood Algorithm works by inserting the edge cells of a DEM into a priority-queue where they are ordered by increasing elevation. The cell with the lowest elevation is popped from the queue and manipulated. Following this, each neighbor which has not already been considered by the algorithm is manipulated and then added to the priority queue. The algorithm continues until the priority queue is empty.

The Priority-Flood Algorithm may be applied to either integer or floating-point DEMs and is optimal for both; the general algorithm is also indifferent as to the underlying connectedness of the DEM and works equally well on 4-, 6-, or 8-connected grids, as well as meshes. As detailed below, special cases of the Priority-Flood Algorithm have been independently described and improved by many authors. Table 1 summarizes the work of these authors. The following is a historic overview of the Priority-Flood Algorithm followed by a description of the algorithm, an important improvement, and details of some of the algorithm's many variants.

Ehlschlaeger [1989] was the first to suggest the Priority-Flood Algorithm, noting that

> The most accurate method for determining watershed boundaries involves placing a person familiar with the nuances of contour maps at a drafting table to manually interpret drainage basins.

He goes on to disparage the use of local 3 x 3 neighbourhoods in determining flow directions, pointing out that a manual interpreter would instead utilize a high-level view of the general flow of water and the location of drainage basins. His variation of the algorithm uses insertion sort and so has a sub-optimal average time complexity of $O(n^2)$ or more. Of all the authors mentioned here, Ehlschlaeger is the only one to mention modeling algorithms on human behavior.

Vincent and Soille [1991] describe an $O(n)$ algorithm for labeling watersheds in integer digital images by sorting all the pixels into an ascending order. The image is then flooded upwards from its lowest pixels. A distance function is used to determine which cells form the boundaries between two watersheds.

Beucher and Meyer [1992] present a similar $O(n)$ algorithm. Each local minima is assigned a unique label. The terrain is then flooded upwards from its lowest points using a hierarchical queue, with the result

being a DEM wherein all cells which ultimately drain to a given minima bear that minima's label. Beucher and Meyer provide a thorough discussion of hierarchical queues work and examples of their algorithm's application to image analysis.

Meyer [1994] gives an overview of many existing methods to derive watershed boundaries. He identifies plateaus and regions of equal elevation as being potentially problematic, but solves the problem by using first-in, first-out (FIFO) queues. This is analogous to the total order priority queues discussed later in this paper.

Soille and Gratin [1994] were the first authors to apply Priority-Flood specifically to depression filling. Their $O(n)$ algorithm is described using the mathematics of morphological filters for use on 4-, 6-, or 8-connected grids with integer values. Their variation floods the DEM inwards from the edges and does not consider local minima.

Wang and Liu [2006] describe the first floating-point variant of the Priority-Flood Algorithm, identifying the time complexity in such a situation as being $O(n \log_2 n)$. They present a lucid comparison of their algorithm versus older, less-efficient depression-filling algorithms by Jenson and Domingue [1988], O'Callaghan and Mark [1984], and Marks et al. [1984].

Liu, Zhang, and Xu [2009] describe a variant of the algorithm by Wang and Liu [2006]; whereas Wang and Liu [2006] use a cell's presence in the priority queue as an indication of whether it should be processed, Liu et al. [2009] recognize that for an 8-connected grid a two-dimensional boolean array offers a clearer correlation between the pseudocode and an efficient implementation.

Liu et al. claim a time complexity of $O(8n \log_2 n)$, but this is too high. Their implementation uses a sorted dictionary (i.e. a map) which bins all cells of equal elevation. As a result, their algorithm actually has a time of $O(n \log_2 k)$, where $k$ is the number of unique elevation levels, which may be less than $n$. As a result of this implementation detail and the small sizes of their test DEMs, Liu et al. incorrectly conclude that Priority-Flood is slower than the Planchon–Darboux Algorithm.

Metz et al. [2010] and Metz et al. [2011] describe a variant of the algorithm using a totally-ordered priority queue (this will be explained below) to determine flow directions in watersheds. In the resulting DEM, every cell has a defined D8 flow direction which is guaranteed to drain to the edge of the DEM. Metz et al. evaluate their method against a "traditional sink filling method" and an impact reduction approach by Lindsay and Creed [2005]; they find that their method yields flow accumulation streams which are closer to GPS field control points than the traditional method for all tested DEMs at all tested resolutions. Similarly, their algorithm out-performed the impact reduction approach in most cases. A detailed account of the method by Metz et al. is given below.

Beucher and Beucher [2011] describe an algorithm similar to that of Beucher and Meyer [1992], but extend it to finding watershed boundaries and image reconstructions. The authors point out that for 32-bit integers a hierarchical queue may have to allocate impossibly large numbers of child queues. To counter this, they present space-efficient implementations of hierarchical queues and methods of overcoming this kind of over-allocation.

Magalhães et al. [2012] describe a Priority-Flood variant using hierarchical queues to achieve $O(n)$ processing on an integer data set, and note that some floating-point data can be discretized. The algorithm they present is essentially the same as that described by Soille and Gratin [1994], but flow directions are generated in the manner of Metz et al. [2010] and Metz et al. [2011]. Although using integer data limits the scope of their method, they claim it is a significant improvement over the floating-point algorithms presented by Wang and Liu [2006] and Liu et al. [2009]. Gomes et al. [2012] extends the work of Magalhães et al. [2012] to increase the efficiency of the Priority-Flood Algorithm in situations where memory access must be limited.

### *3.2. The Algorithm*

Generalizing the work of these authors yields the Priority-Flood Algorithm as described by Alg. 1 and Fig. 1. To initialize the algorithm, all of the edge cells of the DEM (e.g. cells $A$ and $I$ in Fig. 1a) are pushed (i.e. added) onto a priority queue. The priority queue is ordered with cells of lower elevation having greater priority, so that the cell with the lowest elevation in the queue is always the cell popped (i.e. removed from the queue for processing) first.

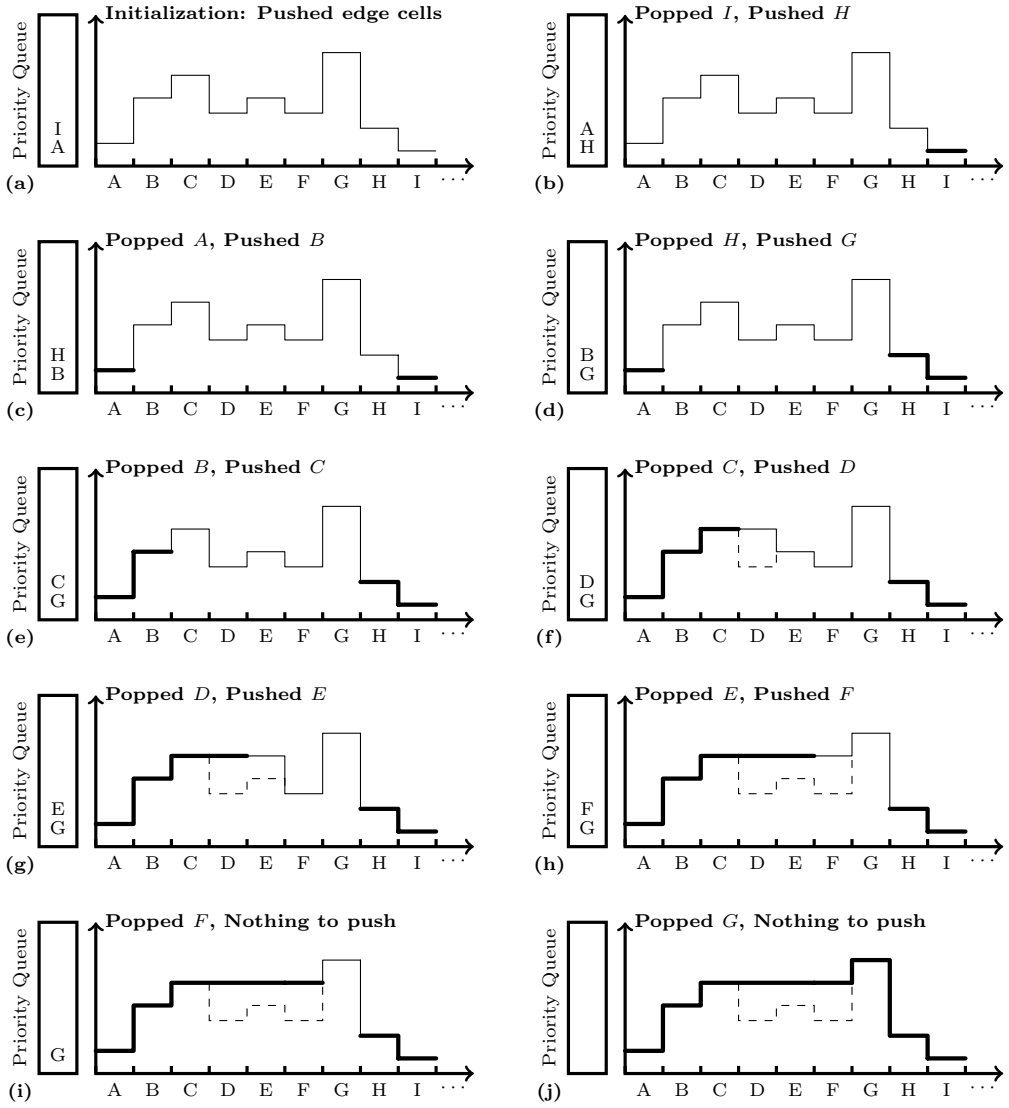


**Figure 1:** A conceptual cross-section of a DEM demonstrating the Priority-Flood Algorithm. The algorithm is initialized in (a), with all the edge cells having been pushed into a priority queue from which the cell with the lowest elevation is popped. In (b)–(e), cells are popped from the priority queue, marked as processed (dark lines on the landscape), and their neighbors pushed. In (f), cell $C$ is the lowest cell which is guaranteed to drain, therefore its neighbor $D$ must be in a depression. To resolve the depression, cell $D$ is raised to the elevation of $C$ before being pushed into the priority queue. $D$'s old elevation is no longer needed, but is shown with a dotted line for clarity. In (g), $E$ must also, by the same logic, be in a depression; therefore, it is raised to the new elevation of $D$. This continues through (i). In (j), the final cells are popped, though there is nothing to do. (k) shows the final elevations of the cells. Since the cells are raised as the algorithm progresses, no extra work is necessary between (j) and (k); the algorithm is complete when the priority queue is empty.

**Algorithm 1** PRIORITY-FLOOD: A generalization of the hierarchical-queue and priority-queue methods described by previous authors. The algorithm is described using pictures in Fig. 1. **Upon entry, (1)** *DEM* contains the elevations of every cell or the value NODATA for cells not part of the DEM. **(2)** The value NODATA is less than the elevation of any cell. **At exit, (1)** *DEM* contains the elevations of every cell or the value NODATA for cells not part of the DEM. **(2)** The elevations of *DEM* are such that there are no digital dams and no undrainable depressions in the landscape, though there may be flats.

**Require:** *DEM*
Let *Open* be a priority queue
Let *Closed* have the same dimensions as *DEM*
Let *Closed* be initialized to FALSE
**for all** $c$ on the edges of *DEM* **do**
  Push $c$ onto *Open* with priority $DEM(c)$
  $Closed(c) \leftarrow$ TRUE
**while** *Open* is not empty **do**
  $c \leftarrow$ POP(*Open*)
  **for all** neighbors $n$ of $c$ **do**
    **if** $Closed(n)$ **then repeat loop**
    $DEM(n) \leftarrow \max(DEM(n), DEM(c))$
    $Closed(n) \leftarrow$ TRUE
    Push $n$ onto *Open* with priority $DEM(n)$

Furthermore, all of the edge cells are marked as resolved in a special boolean array. By definition, edge cells have an $\epsilon$-descending path to the DEM's edge and they are already at the correct elevation, so the depression-filling criteria are preserved.

DEMs—especially gridded DEMs—may be used to represent irregularly-shaped patches of landscape. These patches are akin to islands of data floating in a sea of cells which must be ignored. These ignorable cells are denoted with a special NODATA value which is assumed to be some extremely negative number. Because this makes the NODATA cells lower than any data cell, they have no impact on terrain flooding and can be treated as normal data cells.

Next, cells are popped off of the priority queue. Each cell $c$ which is popped is guaranteed to be the lowest cell with an established drainage path to the edge of the DEM; therefore, if $c$ is at the brink of a depression (such as cell $C$ is in Fig. 1e), then there can be no lower cell which will drain the depression.

As each cell $c$ is popped from the priority queue, its neighbors are added to the priority queue, provided they have not already been resolved. If an unresolved neighbor $n$ is at a lower elevation than $c$ (such as cells $D, E, F$ in Fig. 1), it is raised to the elevation of $c$ before it is placed on the queue. In this way, an $\epsilon$-descending path is constructed for each cell, leading to the fulfillment of the second depression-filling criterion. Because $n$ is always brought up only to the elevation of the lowest cell which still drains to the edge of the DEM, the third depression-filling criterion is fulfilled.

The algorithm terminates when the priority queue is empty (Fig. 1k).

Since all cells are given elevations greater or equal to their original values in the DEM, the first property of the depression-filling criteria is fulfilled. Because all three criteria are fulfilled at each step, it follows that the result of the Priority-Flood Algorithm is a solution to the depression-filling problem.

Because the algorithm does not rely on the structure or connectedness of the underlying DEM, it can be applied to 4-, 6-, 8-, or, indeed, $n$-connected grids. The underlying grid need not even be rectangular—for instance, it may be a mesh based on a Delaunay triangulation.

### *3.3. An Important Improvement*

In the Priority-Flood Algorithm, cells are only raised when they are neighbors of the lowest cell which drains to the edge of the DEM. However, if a cell $c$ is raised, then it must have the same priority as the cell which raised it (this cell having had the highest priority in the priority queue); therefore, there is no need to push $c$ onto the priority queue and incur the associated cost. Rather, $c$ can be pushed onto a plain (first-in, first-out) queue at cost $O(1)$.

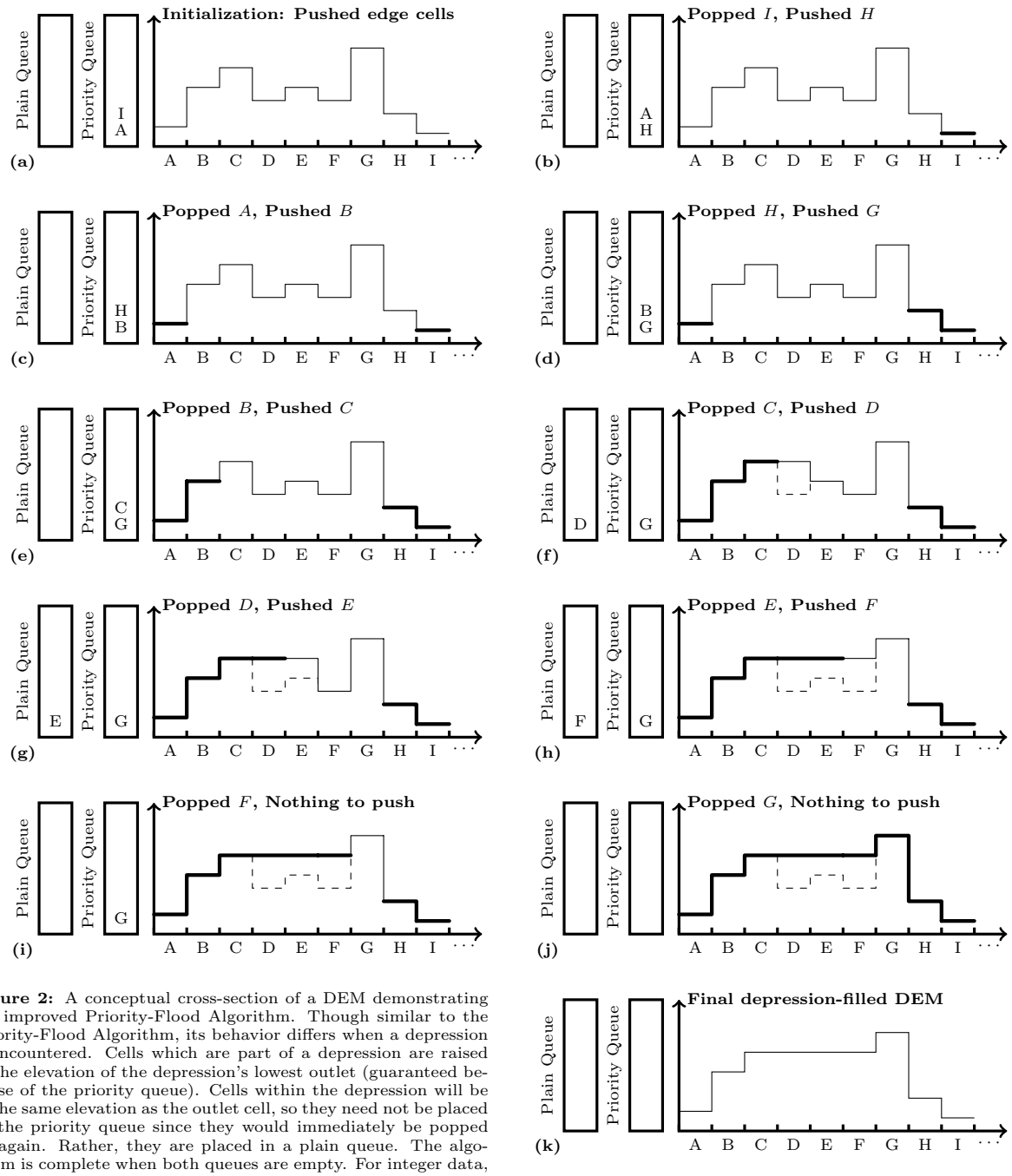


**Figure 2:** A conceptual cross-section of a DEM demonstrating the improved Priority-Flood Algorithm. Though similar to the Priority-Flood Algorithm, its behavior differs when a depression is encountered. Cells which are part of a depression are raised to the elevation of the depression's lowest outlet (guaranteed because of the priority queue). Cells within the depression will be at the same elevation as the outlet cell, so they need not be placed on the priority queue since they would immediately be popped off again. Rather, they are placed in a plain queue. The algorithm is complete when both queues are empty. For integer data, this eliminates overhead; for floating-point data, this eliminates $O(\log_2 n)$ work per cell in the depression.

This improvement is described by Alg. 2 and Fig. 2. As before, the algorithm is initialized by pushing all the edge cells of the DEM onto a priority queue (Fig. 2a). Cells are popped from this queue and their neighbors pushed onto it (Fig. 2b–e). However, if a neighbor is lower than the cell which is pushing it, the neighbor is raised to the elevation of that cell and added to a plain queue (Fig. 2f–i); if there are cells in the plain queue, they are always popped before cells in the priority queue. The algorithm terminates when there are no cells left in either queue (Fig. 2k).

**Algorithm 2** IMPROVED PRIORITY-FLOOD: This improvement to the PRIORITY-FLOOD uses a plain queue to speed-up the filling of depressions, once they are found. **Upon entry, (1)** *DEM* contains the elevations of every cell or the value NODATA for cells not part of the DEM. **(2)** The value NODATA is less than the elevation of any cell. **At exit, (1)** *DEM* contains the elevations of every cell or the value NODATA for cells not part of the DEM. **(2)** The elevations of *DEM* are such that there are no digital dams and no undrainable depressions in the landscape, though there may be flats.

**Require:** *DEM*
Let *Open* be a priority queue
Let *Pit* be a plain queue
Let *Closed* have the same dimensions as *DEM*
Let *Closed* be initialized to FALSE
**for all** $c$ on the edges of *DEM* **do**
  Push $c$ onto *Open* with priority $DEM(c)$
  $Closed(c) \leftarrow$ TRUE
**while** either *Open* or *Pit* is not empty **do**
  **if** *Pit* is not empty **then**
    $c \leftarrow \text{POP}(Pit)$
  **else**
    $c \leftarrow \text{POP}(Open)$
  **for all** neighbors $n$ of $c$ **do**
    **if** $Closed(n)$ **then repeat loop**
    $Closed(n) \leftarrow$ TRUE
    **if** $DEM(n) \leq DEM(c)$ **then**
      $DEM(n) \leftarrow DEM(c)$
      Push $n$ onto *Pit*
    **else**
      Push $n$ onto *Open* with priority $DEM(n)$

## 4. Ordering

A subtlety of priority queues concerns how elements of equal priority are treated. If elements of equal priority retain their order of insertion, then the priority queue may be said to be "stable" or to have a "total order". If elements of equal priority do not retain their order of insertion, then the priority queue has a "strict weak ordering": elements of equal priority are incomparable and may therefore be arranged in any order.

Fortunately, for depression filling with $\epsilon = 0$, it does not matter whether the priority queue has a total order or a strict weak order, as both will produce the same results. However, if $\epsilon \neq 0$, or if Priority-Flood is used for watershed labeling or to determine flow directions directly, then the ordering is important and a total order is the best choice.

A total order produces a predictable, reproducible result, while the results of using an underlying algorithm with a strict weak order may only, in general, be reproduced by using that same algorithm. If $\epsilon \neq 0$, then a total ordering guarantees the third depression-filling criterion; furthermore, a total order guarantees that each cell has a least-cost (i.e. shortest) path to its flooding source. This means that depressions are able to drain from multiple outlets, provided the outlets are of equal elevation. If the priority queue uses a strict weak ordering then it cannot make these guarantees.

In situations where a total order is necessary—e.g. $\epsilon \neq 0$, watershed labeling, determining flow directions—it is possible to make an algorithm with strict weak ordering produce a total order. To do so, each cell is associated with two priorities (i.e. keys). The first is the cell's elevation, as before; the second is an integer which begins at zero and is incremented every time a cell is added to the priority queue. In the instance that two cells have equal elevation, the secondary priorities are used to determine the ordering. This assures that all cells are comparable and that cells of equal elevation emerge from the priority queue in the order they were inserted.

| Data Structure | Insert | DecreaseKey | DeleteMin |
|---|---|---|---|
| Hierarchical Queue† | $O(1)$ | — | $O(1)$ |
| Calendar Queue‡ | $O(1)$ | — | $O(1)$ |
| 1-Level Bucket | $O(1)$ | $O(1)$ | $O(C)$ |
| 2-Level Bucket | $O(1)$ | $O(1)$ | $O(\sqrt{C})$ |
| Radix Heap | $O(1)^*$ | $O(1)^*$ | $O(\lg C)^*$ |
| Emde Boas | $O(\lg \lg N)$ | $O(\lg \lg N)$ | $O(\lg \lg N)$ |
| Binary Search Tree | $O(\lg n)$ | $O(\lg n)$ | $O(\lg n)$ |
| Binomial Queue | $O(\lg n)$ | $O(\lg n)$ | $O(\lg n)$ |
| Heap | $2 \lg n$ | $2 \lg n$ | $2 \lg n$ |
| Weak Heap | $\lg n$ | $\lg n$ | $\lg n$ |
| Pairing Heap | $O(1)$ | $O(2^{\sqrt{\lg \lg n}})$ | $O(\lg n)^*$ |
| Fibonacci Heap | $O(1)$ | $O(1)^*$ | $O(\lg n)^*$ |
| Relaxed Weak Queue | $O(1)$ | $O(1)$ | $O(\lg n)$ |

**Table 2:** Time complexities of priority queue operations for various underlying data structures—there are many other possibilities which are not listed. The top section of the table assumes that all priorities are integers. In the table, $C$ is the maximal edge weight, $N$ the maximum key, and $n$ the number of nodes stored. Values marked with ($^*$) denote amortized costs. The function 'lg' stands for '$\log_2$'. †As described by Beucher and Meyer [1992]. ‡As described by Brown [1988], among other authors. All other entries are drawn from Edelkamp and Schrödl [2011].

## 5. Analysis

The essence of the Priority-Flood Algorithm is that a priority queue is initialized with some number of seed cells. The DEM is then progressively flooded by repeatedly pushing the neighbors of the highest priority cell into the priority queue. The improvemed Priority-Flood uses, in essence, a specialized priority queue.

As Table 2 shows, there are many possible ways to manage a priority queue—and the list here is by no means exhaustive, Knuth [1998] discusses many others in addition to presenting detailed analyses of some of the underlying priority queue algorithms. Yet, according to Edelkamp and Schrödl [2011], such management requires at most three functions. INSERT pushes an element onto a priority queue, DECREASEKEY deletes an arbitrary element and reinserts it into a more appropriate place, and DELETEMIN simultaneously accesses and pops the element with the greatest priority. The time complexity of operations on the priority queue is dependent on the time complexity of these operations.

For integer DEMs, it is evident that the priority queue should be based on one of the algorithms from the upper portion of Table 2. The algorithms from the lower portion will also work, albeit sub-optimally. In particular, hierarchical queues [Beucher and Meyer, 1992] are $O(1)$ with extremely low overhead for all operations; however, if there are many different elevation levels, the methods of Beucher and Beucher [2011] will need to be applied. Calendar queues [Brown, 1988] have higher overhead, but this becomes negligible for large problems and they are well suited to the monotonically-increasing priorities exhibited by Priority-Flood. Calendar queues may be improved through delayed sorting [Rönngren et al., 1991; Steinman, 1994, 1996; Rönngren and Ayani, 1997], choosing or adaptively determining appropriately sized bins [Oh and Ahn, 1997, 1999; Tan and Thng, 2000; Hui and Thng, 2002; Siangsukone et al., 2003; Tang et al., 2005], reducing resize operations [Goh and Thng, 2003, 2004], or leveraging statistical properties of the priority of newly-inserted elements [Yan and Eidenbenz, 2006]. However, with the efficiency of the calendar queue comes subtleties in programming and determining algorithmic correctness. Bucket sorts or radix heaps may be simpler to implement while still retaining acceptable performance for most problems.

Regardless, since the priority queue operates in $O(1)$ time for integer data, Priority-Flood will operate in $O(n)$ time, which is optimal. In special cases, floating-point DEMs may be quantized and treated as integer DEMs. This may be the case when there are known resolution increments or if a floor function can be used to make suitably small buckets. Alternatively, an "untidy priority queue" which rounds floating-point values may be used; Yatziv et al. [2006] describe such a queue and show that the error introduced by using it can be limited to the same order of magnitude as the errors introduced by spatial discretization, making them "virtually insignificant".

In the general case of floating-point DEMs, an $O(\log_2 n)$ data structure must be used, such as one of those listed in the bottom portion of Table 2. The time complexity of Priority-Flood in such a scenario is $O(n \log_2 n)$. This may be reduced to $O(n \log_2 k)$ if the mapped queues method of Liu et al. [2009] is used.

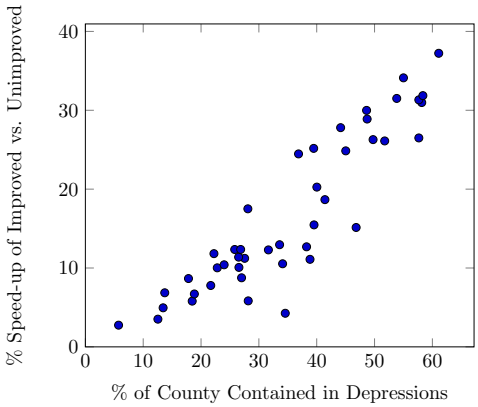


**Figure 3:** Comparison of the improved Priority-Flood (Alg. 2) and the unimproved Priority-Flood (Alg. 1) algorithm. Tests were performed on 3 m floating-point DEMs of 44 of Minnesota's counties, a 72,500 km$^2$ area covering approximately 33% of the state. It is clear that counties with more depressions are processed faster.

However, time complexity alone does not tell the whole story: data structures have hidden overheads which are not expressible in terms of time complexity. Luengo Hendriks [2010] performed extensive testing on implementations of eleven different priority queue algorithms and found sometimes dramatic differences in the time required to remove and then add an item to a filled queue. The algorithms also differed markedly for operations which filled an empty queue or emptied a filled queue.

Based on these tests, Luengo Hendriks recommended the *implicit heap* as the best choice for a priority queue: it used the least memory of all the algorithms tested and operated the fastest, though it does use a strict weak ordering. Hierarchical heaps and ladder queues operated faster for very large data sets (greater than $10^6$ elements), but used significantly more memory.

For floating-point DEMs, the improved Priority-Flood above reduces both the time complexity—to $O(m \log_2 m)$, where $m \leq n$—and run-time regardless of which underlying algorithm is used for the priority queue. It may also decrease run-times even for integer data by avoiding the overhead associated with maintaining a priority queue. If non-integer keys are used or guarantees regarding the internal structure of the priority queue are absent, prudent design suggests that the improved algorithm be used.

For all variants of the Priority-Flood Algorithm, the spatial distribution and connectedness of depressions in the DEM does not affect the algorithms' time complexities.

## 6. Empirical Testing

An implementation of the improved Priority-Flood was tested against an implementation of the unimproved Priority-Flood; both implementations are included in the Supplemental Materials. The `C++` STL priority queue was chosen to simplify programming and the tests were conducted on floating-point DEMs using a 64-bit Intel Xeon X5560 2.8GHz 8192kB cache processor and 24GB of RAM. The results are shown in Fig. 3. LIDAR-based 3 m DEMs of 44 counties in Minnesota were tested, comprising an area of approximately 72,500 km$^2$—one-third of the state—with approximately 111,111 cells per square kilometer for a total of approximately $8 \cdot 10^9$ cells; the average county size was 1,741 km$^2$ ($2 \cdot 10^8$ cells). The improved Priority-Flood Algorithm out-performed the unimproved algorithm in every case. The average speed-up over all the counties tested was 16.8%, while the median was 12.5%. The maximal speed-up was 37.2%.

Earlier, the Planchon–Darboux Algorithm was discussed. This algorithm continues to see usage despite being slower than several Priority-Flood variants which have been developed both before and since. Notably, Wallis et al. [2009] produced an MPI-based parallel implementation of the Planchon–Darboux Algorithm for the TauDEM terrain analysis package and showed that it ran faster than a serial implementation for DEMs of moderate to large size. To demonstrate that the Priority-Flood Algorithm can run faster than

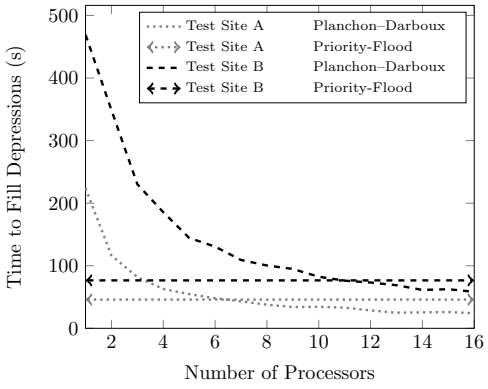


**Figure 4:** Comparison of the TauDEM parallel implementation [Wallis et al., 2009] of the Planchon–Darboux Algorithm versus the improved Priority-Flood algorithm. While the Planchon–Darboux Algorithm was tested with varying numbers of processors, the improved Priority-Flood was tested using only a single processor, resulting in the flat lines.

the Planchon–Darboux Algorithm and to clarify a previous comparison made by Liu et al. [2009], the two algorithms are compared here.

Both algorithms were compiled with full optimizations. The parallel implementation of Planchon–Darboux was run on varying numbers of processors while Priority-Flood was run on only a single processor. The machine used for the testing was the same as described above; it had eight processors per node and two dedicated nodes were used for parallel testing.

Due to long run-times, two test sites were chosen for comparing the speed of the improved Priority-Flood algorithm against the TauDEM 5.0.6 parallel implementation of the Planchon–Darboux algorithm. Test Site A was Minnesota's Steele County and Test Site B was Minnesota's Nicollet County. The two test sites represent counties with an average (35%) and a large (61%) number of depressions, respectively. Thus, the results of the test should not be biased by the choice of counties. The test sites were also representative in terms of size. Test Site A was a 3 m DEM of 10891 x 13914 cells ($152 \cdot 10^6$ total) and Test Site B was a 3 m DEM of 24140 x 13183 cells ($318 \cdot 10^6$ total).

As shown in Fig. 4, the parallel implementation of the Planchon–Darboux Algorithm required six processors in order to process Test Site A in the same time as the improved Priority-Flood algorithm did with one processor. The speed difference was even more striking in the case of Test Site B, where 11 processors were required.

Fig. 6 depicts the numerous large and small depressions of Test Site A. Although the topography may appear complex, the run-times of the algorithms presented here are largely independent of the distribution and connectedness of flats. Careful judgement should be exercised when determining whether or not it is appropriate to apply a particular algorithm to a particular landscape for a particular application.

## 7. Variants

### *7.1. Automatic Flat Resolution*

Priority-Flood, as it has been described above, will produce mathematically-flat surfaces as a by-product of filling depressions. Such surfaces confound attempts to determine flow directions and derivative hydrologic properties. A feature of the Planchon–Darboux Algorithm is that it does not suffer this problem because it produces surfaces for which each cell has a defined gradient from which flow directions can be determined.

As Alg. 3 demonstrates, Priority-Flood can be adapted to provide this functionality. To do so, depressions are filled such that adjacent cells along paths running to outlets are not set to the same elevation, but, rather, have some minimal elevation difference $\epsilon$ between themselves; that is, each cell is made to be a part of an $\epsilon$-descending path with $\epsilon > 0$.

**Algorithm 3** PRIORITY-FLOOD+$\epsilon$: This variation of the IMPROVED PRIORITY-FLOOD ensures that all filled depressions have surfaces with a height difference of at least $\epsilon$ between any two consecutive cells as one moves away from or towards the depression's outlet. To minimize disruption to the DEM, Lines 24 and 28 use the NEXTAFTER function, as described in the text on Page 12. The algorithm raises a flag if the modified DEM has been altered in a way which cannot be minimized. **Upon entry, (1)** *DEM* contains the elevations of every cell or the value NODATA for cells not part of the DEM. **At exit, (1)** *DEM* contains the elevations of every cell or the value NODATA for cells not part of the DEM. **(2)** The elevations of *DEM* are such that there are no digital dams and every cell will drain to the edge of the DEM.

**Require:** *DEM*
1: Let *Open* be a priority queue with total order
2: Let *Pit* be a plain queue
3: Let *Closed* have the same dimensions as *DEM*
4: Let *Closed* be initialized to FALSE
5: **for all** $c$ on the edges of *DEM* **do**
6: Push $c$ onto *Open* with priority $DEM(c)$
7: $Closed(c) \leftarrow$ TRUE
8: **while** either *Open* or *Pit* is not empty **do**
9: **if** the top of *Open* = the top of *Pit* **then**
10: $c \leftarrow$ POP(*Open*)
11: *PitTop* $\leftarrow$ NONE
12: **else if** *Pit* is not empty **then**
13: $c \leftarrow$ POP(*Pit*)
14: **if** *PitTop* = NONE **then**
15: *PitTop* $\leftarrow DEM(c)$
16: **else**
17: $c \leftarrow$ POP(*Open*)
18: *PitTop* $\leftarrow$ NONE
19: **for all** neighbors $n$ of $c$ **do**
20: **if** $Closed(n)$ **then repeat loop**
21: $Closed(n) \leftarrow$ TRUE
22: **if** $DEM(n) =$ NODATA **then**
23: Push $n$ onto *Pit*
24: **else if** $DEM(n) \leq$ NEXTAFTER$(DEM(c), \infty)$ **then**
25: **if** *PitTop* $< DEM(n)$ **and** NEXTAFTER$(DEM(c), \infty) \geq DEM(n)$ **then**
26: A significant alteration of the DEM has occurred
27: The inside of the pit is now higher than the terrain surrounding it
28: $DEM(n) \leftarrow$ NEXTAFTER$(DEM(c), \infty)$
29: Push $n$ onto *Pit*
30: **else**
31: Push $n$ onto *Open* with priority $DEM(n)$

Although Alg. 3 is similar to Alg. 2, there are important differences. Line 1 requires that the priority queue have a total order—as discussed in §4. This allows depressions to drain from multiple outlets. Line 28 produces the $\epsilon$-descending path. The clause beginning on Line 9 maintains the total order by ensuring that all cells of equal elevation which may border a depression are treated equally.

Ideally, $\epsilon$ is large enough to direct flow, but sufficiently small as to have no other effects on the DEM's hydrologic properties. However, to work with such small values is to expose the intricacies of floating-point arithmetic. If too small an $\epsilon$ is used, then adding it to a cell may produce no change in its elevation; if too great an $\epsilon$ is used, then large depressions may be converted into mesas rising above the surrounding landscape.

Choice of an appropriate value for $\epsilon$ is neither straight-forward nor universal. Therefore, it is best to use the NEXTAFTER function defined by the `C99` and `POSIX` standards, or a similar function. This function

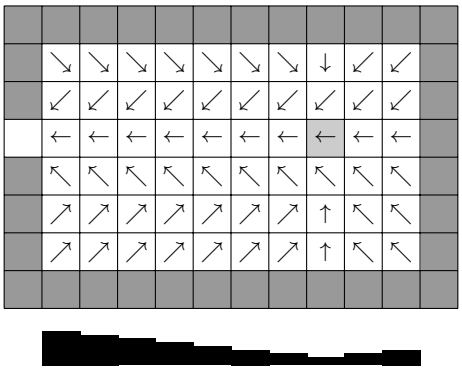

**Figure 5:** Demonstration of Priority-Flood+FlowDirs (Alg. 4). The dark shaded edge cells represent the border of a depression: their flow directions are omitted. The depression has an outlet along its left-hand side and a lightly shaded sink cell to the right of its center. The cells of each column—except for the sink, which is lower than any cell—have a relative elevation corresponding to the black elevation profile.

increases or decreases a floating-point number by what is guaranteed to be the smallest possible increment in the direction of a second number. While this is better than using an arbitrarily-defined $\epsilon$, it is still not possible to guarantee that Alg. 3 will behave correctly in situations where a DEM's precision approaches that of its storage data type.

Therefore, the variable *PitTop* is set (Line 15) whenever a new depression is encountered and reset (Lines 11 and 18) after the depression has been resolved. If incrementing a cell causes it to rise above a cell which was previously higher than the level of the pit's outlet, as defined by *PitTop*, then Line 25 issues a warning. The Planchon–Darboux Algorithm does not provide such a safety check.

An alternative approach to the algorithm just described would be to use Priority-Flood to fill the DEM's depressions and a secondary algorithm to resolve the flat surfaces which result. Barnes et al. [2013] provides an $O(n)$ algorithm for generating convergent flow patterns in such a situation. One could also impose flow directions on the DEM directly using a variant of the Priority-Flood Algorithm described below.

### *7.2. Flow Directions*

An alternative to using infinitesimal increments to direct flow is to calculate flow directions directly from the DEM, as in Alg. 4. As in the other Priority-Flood Algorithms, the terrain is flooded inwards from the edges; however, rather than altering cells' elevations, this algorithm always causes the neighbors of the lowest cell to point towards it before pushing them into the priority queue using their unaltered elevations as their priorities. As a result, the algorithm climbs into depressions through their lowest outlet and takes the path of steepest descent to the depressions' minima. Depressions then drain towards their minima and this path, as shown in Fig. 5. All depressions will drain to the edge of the DEM, but elevation information within depressions is still utilized for determining flow directions. The algorithm works well with nested depressions. Alg. 4 is, in effect, a depression-carving algorithm: rather than filling depressions, it drills through their walls.

As noted by Metz et al. [2010], the flow directions determined by this algorithm yield regions of flow accumulation which are located closer to actual rivers than methods which determine flow directions after performing depression-filling.

### *7.3. Watershed Labeling*

Watershed labeling applies a common label—such as an integer number—to all cells which drain to a given outlet. The algorithm (Alg. 5) for this works in much the same way as the improved Priority-Flood (Alg. 2). The DEM is flooded inwards from its edges with the lowest cell always being processed first. Rather than filling cells in depressions, this algorithm merely prioritizes them to the level of their outlet.

Watershed outlets are identified as being unlabeled cells adjacent to a NoData cell. These cells are given a unique label which then floods inwards to cover all the cells in the watershed.

**Algorithm 4** Priority-Flood+FlowDirs: This variation of Priority-Flood is based on the work of Metz et al. [2011] and determines flow directions for all cells. This is, in effect, a depression-carving operation. On 8-connected grids, non-diagonal neighbors should be processed first on Line 13 as they have the greatest center-to-center slopes and therefore should take priority. A plain queue cannot be used to speed this variation up because all the elevation information inside the depressions is needed. **Upon entry, (1)** *DEM* contains the elevations of every cell or the value NoData for cells not part of the DEM. **At exit, (1)** *FlowDirs* contains the flow direction of every cell or the value NoData for cells not part of the DEM. **(2)** All cells which are part of the DEM have a defined flow direction.

**Require:** *DEM*, *FlowDirs*
1: Let *Open* be a priority queue with total order
2: Let *Closed* have the same dimensions as *DEM*
3: Let *Closed* be initialized to FALSE
4: **for all** $c$ on the edges of *DEM* **do**
5: Push $c$ onto *Open* with priority $DEM(c)$
6: $Closed(c) \leftarrow$ TRUE
7: **if** $DEM(c) =$ NoData **then**
8: $FlowDirs(c) \leftarrow$ NoData
9: **else**
10: $FlowDirs(c)$ points off of the DEM
11: **while** *Open* is not empty **do**
12: $c \leftarrow$ POP(*Open*)
13: **for all** neighbors $n$ of $c$ **do** ▷ Non-diagonal neighbors first
14: **if** $Closed(n)$ **then repeat loop**
15: **if** $DEM(n) =$ NoData **then**
16: $FlowDirs(n) \leftarrow$ NoData
17: **else**
18: $FlowDirs(n)$ points towards $c$
19: $Closed(n) \leftarrow$ TRUE
20: Push $n$ onto *Open* with priority $DEM(n)$

Following execution, watershed boundaries may be identified by locating adjacent cells with differing labels. To mark the border cells, consistently choose either the cell with the lower label, the cell with the higher label, or both.

## 8. Coda

Special cases and variants of the Priority-Flood Algorithm have been described by many authors. This paper generalizes this body of literature (Alg. 1) to work optimally on either integer or floating-point data (§5), as well as on irregular meshes or 4-, 6-, 8-, or $n$-connected grids.

An improvement to the Priority-Flood priority queue (Alg. 2) has been described and tested. It runs in $O(m \log_2 m)$ time, where $m \leq n$, on floating-point data and in $O(n)$ time on integer data. By comparison, the Planchon–Darboux Algorithm has a time complexity of at least $O(n^{1.2})$ and the generalized Priority-Flood Algorithm has a time complexity of $O(n \log_2 n)$ for floating-point DEMs and $O(n)$ for integer DEMs. Under testing, the new algorithm outperformed the generalized Priority-Flood Algorithm in all cases. Improvements were often greater than 16% and were as high as 37%. In addition, a parallel implementation of the Planchon–Darboux Algorithm required upwards of six processors to match the improved Priority-Flood's speed with a single processor.

For its simplicity and speed, Priority-Flood is a good choice. It can fill depressions in such a way that they are guaranteed to drain; it is explicable in 20 lines of pseudocode; and, as demonstrated by the `C++` reference code (see Supplemental Materials), it can be implemented in fewer than a hundred lines of code.

The Priority-Flood algorithm is also versatile. It can be used to label watersheds (Alg. 5) as well as to determine flow directions either by terrain increments (Alg. 3) or by carving depressions (Alg. 4).

**Algorithm 5** Improved Priority-Flood+Watershed Labels: This variation of the Improved Priority-Flood follows the work of Beucher and Meyer [1992] and Beucher and Beucher [2011]. It applies a common label to all cells draining to an outlet. Line 21 should be interpreted as pushing a copy of the cell's coordinates into *Pit* with the copy's $z$-value set to $c.z$. If simultaneous watershed labeling and depression-filling is desired, change the original $z$ value of $n$ to $c.z$ before making the copy. **Upon entry, (1)** *DEM* contains the elevations of every cell or the value NoData for cells not part of the DEM. **At exit, (1)** *Labels* contains a label for every cell or the value NoData for cells not part of the DEM. **(2)** All cells which drain to a common point at the edge of the DEM bear the same label.

**Require:** *DEM*, *Labels*
1: Let *Open* be a priority queue
2: Let *Pit* be a plain queue holding cells' $(x, y, z)$
3: Let *Labels* have the same dimensions as *DEM*
4: Let *Labels* be initialized to CANDIDATE
5: *label* $\leftarrow 1$
6: **for all** $c$ on the edges of *DEM* **do**
7: Push $c$ onto *Open* with priority *DEM*$(c)$
8: *Labels*$(c) \leftarrow$ QUEUED
9: **while** either *Open* or *Pit* is not empty **do**
10: **if** *Pit* is not empty **then**
11: $c \leftarrow$ POP(*Pit*)
12: **else**
13: $c \leftarrow$ POP(*Open*)
14: **if** *Labels*$(c) =$ QUEUED **and** *DEM*$(c) \neq$ NoData **then**
15: *Labels*$(c) \leftarrow$ *label*
16: Increment *label*
17: **for all** neighbors $n$ of $c$ **do**
18: **if** *Labels*$(n) \neq$ CANDIDATE **then repeat loop**
19: *Labels*$(n) \leftarrow$ *Labels*$(c)$
20: **if** *DEM*$(n) \leq c.z$ **then**
21: Push $n$ onto *Pit* with $z = c.z$
22: **else**
23: Push $n$ onto *Open* with priority *DEM*$(n)$

The Supplemental Materials for this paper are available from the journal, as well as at https://github.com/r-barnes/Barnes2013-Depressions. Many of the algorithms presented here are implemented in the RichDEM analysis package, available at http://rbarnes.org/richdem or via email from the authors.

## 9. Acknowledgments

Funding for this work was provided by the Minnesota Environment and Natural Resources Trust Fund under the recommendation and oversight of the Legislative-Citizen Commission on Minnesota Resources (LCCMR). David Mulla was the P.I. on this grant. Supercomputing time and data storage were provided by the Minnesota Supercomputing Institute. Adam Clark provided feedback on the paper; Joel Nelson provided LIDAR DEMs and ArcGIS support. In-kind support was provided by Earth Dance Farm, Tess Gallagher, Justin Konen, and Steffi O'Brien.

Arge, L., Chase, J., Halpin, P., Toma, L., Vitter, J., Urban, D., Wickremesinghe, R., 2003. Efficient flow computation on massive grid terrain datasets. GeoInformatica 7 (4), 283–313. doi: http://dx.doi.org/10.1023/A:1025526421410

Barnes, R., Lehman, C., Mulla, D., 2013. An efficient assignment of drainage direction over flat surfaces in raster digital elevation models. Computers & Geosciences. doi: http://dx.doi.org/10.1016/j.cageo.2013.01.009

Beucher, N., Beucher, S., April 2011. Hierarchical queues: general description and implementation in mamba image library. URL http://cmm.ensmp.fr/~beucher/publi/HQ_algo_desc.pdf

Beucher, S., Meyer, F., 1992. The morphological approach to segmentation: the watershed transformation. Optical Engineering 34, 433–481.

Brown, R., Oct. 1988. Calendar queues: a fast 0(1) priority queue implementation for the simulation event set problem. Communications of the ACM 31 (10), 1220–1227. doi: http://dx.doi.org/10.1145/63039.63045

Edelkamp, S., Schrödl, S., 2011. Heuristic Search: Theory and Applications. Morgan Kaufmann.

Ehlschlaeger, C., 1989. Using the at search algorithm to develop hydrologic models from digital elevation data. In: International Geographic Information Systems (IGIS) Symposium. Vol. 89. pp. 275–281.

Goh, R., Thng, I., 2003. Mlist: An efficient pending event set structure for discrete event simulation. International Journal of Simulation-Systems, Science & Technology 4 (5-6), 66–77.

Goh, R., Thng, I., 2004. Dsplay: An efficient dynamic priority queue structure for discrete event simulation. In: SimTecT Simulation Technology and Training Conference, Australia. Citeseer.

Gomes, T. L., Magalhães, S. V. G., Andrade, M. V. A., Franklin, W. R., Pena, G. C., 2012. Computing the drainage network on huge grid terrains. In: Proceedings of the 1st ACM SIGSPATIAL International Workshop on Analytics for Big Geospatial Data. BigSpatial '12. ACM, New York, NY, USA, pp. 53–60. URL http://doi.acm.org/10.1145/2447481.2447488 doi: http://dx.doi.org/10.1145/2447481.2447488

Hui, T., Thng, I., 2002. Felt: A far future event list structure optimized for calendar queues. Simulation 78 (6), 343–361. doi: http://dx.doi.org/10.1177/0037549702078006573

Jenson, S., Domingue, J., 1988. Extracting topographic structure from digital elevation data for geographic information system analysis. Photogrammetric engineering and remote sensing 54 (11), 1593–1600.

Knuth, D. E., May 1998. Art of Computer Programming, Volume 3: Sorting and Searching, 2nd Edition. Addison-Wesley Professional.

Lindsay, J., Creed, I., 2005. Removal of artifact depressions from digital elevation models: towards a minimum impact approach. Hydrological processes 19 (16), 3113–3126.

Liu, Y., Zhang, W., Xu, J., 2009. Another fast and simple dem depression-filling algorithm based on priority queue structure. Atmospheric and Oceanic Science Letters 2, 214–219.

Luengo Hendriks, C., 2010. Revisiting priority queues for image analysis. Pattern Recognition 43 (9), 3003–3012. doi: http://dx.doi.org/10.1016/j.patcog.2010.04.002

Magalhães, S. V. G., Andrade, M. V. A., Randolph Franklin, W., Pena, G. C., 2012. A new method for computing the drainage network based on raising the level of an ocean surrounding the terrain. In: Gensel, J., Josselin, D., Vandenbroucke, D., Cartwright, W., Gartner, G., Meng, L., Peterson, M. P. (Eds.), Bridging the Geographic Information Sciences. Lecture Notes in Geoinformation and Cartography. Springer Berlin Heidelberg, pp. 391–407. doi: http://dx.doi.org/10.1007/978-3-642-29063-3_21

Marks, D., Dozier, J., Frew, J., 1984. Automated basin delineation from digital elevation data. Geo-Processing 2, 299–311.

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

Metz, M., Mitasova, H., Harmon, R., 2010. Accurate stream extraction from large, radar-based elevation models. Hydrology and Earth System Sciences Discussions 7, 3213–3235.

Metz, M., Mitasova, H., Harmon, R., 2011. Efficient extraction of drainage networks from massive, radar-based elevation models with least cost path search. Hydrology and Earth System Sciences 15 (2), 667. doi: http://dx.doi.org/10.5194/hess-15-667-2011

Meyer, F., 1994. Topographic distance and watershed lines. Signal processing 38 (1), 113–125.

Nardi, F., Grimaldi, S., Santini, M., Petroselli, A., Ubertini, L., 2008. Hydrogeomorphic properties of simulated drainage patterns using digital elevation models: the flat area issue/propriétés hydro-géomorphologiques de réseaux de drainage simulés à partir de modèles numériques de terrain: la question des zones planes. Hydrological Sciences Journal 53 (6), 1176–1193. doi: http://dx.doi.org/10.1623/hysj.53.6.1176

O'Callaghan, J., Mark, D., Dec. 1984. The extraction of drainage networks from digital elevation data. Computer Vision, Graphics, and Image Processing 28 (3), 323–344. doi: http://dx.doi.org/10.1016/S0734-189X(84)80011-0

Oh, S., Ahn, J., 1997. Dynamic lazy calendar queue: An event list for network simulation. In: High Performance Computing on the Information Superhighway, 1997. HPC Asia'97. IEEE, pp. 254–259. doi: http://dx.doi.org/10.1109/HPC.1997.592156

Oh, S., Ahn, J., 1999. Dynamic calendar queue. In: Simulation Symposium, 1999. Proceedings. 32nd Annual. IEEE, pp. 20–25. doi: http://dx.doi.org/10.1109/SIMSYM.1999.766449

Planchon, O., Darboux, F., 2002. A fast, simple and versatile algorithm to fill the depressions of digital elevation models. Catena 46 (2-3), 159–176. doi: http://dx.doi.org/10.1016/S0341-8162(01)00164-3

Rönngren, R., Ayani, R., Apr. 1997. A comparative study of parallel and sequential priority queue algorithms. ACM Transactions on Modeling and Computer Simulation 7 (2), 157–209. doi: http://dx.doi.org/10.1145/249204.249205

Rönngren, R., Riboe, J., Ayani, R., Apr. 1991. Lazy queue: an efficient implementation of the pending-event set. ACM SIGSIM Simulation Digest 21 (3), 194–204. doi: http://dx.doi.org/10.1145/106073.306860

Siangsukone, T., Aswakul, C., Wuttisittikulkij, L., 2003. Study of optimised bucket widths in calendar queue for discrete event simulator. In: Proceedings of Thailand's Electrical Engineering Conference (EECON-26). Citeseer, pp. 6–7.

Soille, P., Gratin, C., 1994. An efficient algorithm for drainage network extraction on dems. Journal of Visual Communication and Image Representation 5 (2), 181–189. doi: http://dx.doi.org/10.1006/jvci.1994.1017

Steinman, J. S., Jul. 1994. Discrete-event simulation and the event horizon. ACM SIGSIM Simulation Digest 24 (1), 39–49. doi: http://dx.doi.org/10.1145/195291.182490

Steinman, J. S., 1996. Discrete-event simulation and the event horizon part 2: event list management. In: Proceedings of the tenth workshop on Parallel and distributed simulation. PADS '96. IEEE Computer Society, Washington, DC, USA, pp. 170–178. doi: http://dx.doi.org/10.1145/238788.238841

Tan, K. L., Thng, L.-J., 2000. Snoopy calendar queue. In: Proceedings of the 32nd conference on Winter simulation. WSC '00. Society for Computer Simulation International, San Diego, CA, USA, pp. 487–495.

Tang, W. T., Goh, R. S. M., Thng, I. L.-J., Jul. 2005. Ladder queue: An o(1) priority queue structure for large-scale discrete event simulation. ACM Transactions on Modeling and Computer Simulation 15 (3), 175–204. doi: http://dx.doi.org/10.1145/1103323.1103324

Vincent, L., Soille, P., Jun. 1991. Watersheds in digital spaces: An efficient algorithm based on immersion simulations. IEEE Transactions on Pattern Analysis and Machine Intelligence 13 (6), 583–598. doi: http://dx.doi.org/10.1109/34.87344

Wallis, C., Wallace, D., Tarboton, D., Watson, D., Schreuders, K., Tesfa, T., 2009. Hydrologic terrain processing using parallel computing. In: 18th World IMACS Congress and MODSIM09 International Congress on Modelling and Simulation, Modelling and Simulation Society of Australia and New Zealand and International Association for Mathematics and Computers in Simulation. pp. 2540–2545.

Wang, L., Liu, H., 2006. An efficient method for identifying and filling surface depressions in digital elevation models for hydrologic analysis and modelling. International Journal of Geographical Information Science 20 (2), 193–213. doi: http://dx.doi.org/10.1080/13658810500433453

Yan, G., Eidenbenz, S., 2006. Sluggish calendar queues for network simulation. In: Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, 2006. MASCOTS 2006. 14th IEEE International Symposium on. IEEE, pp. 127–136. doi: http://dx.doi.org/10.1109/MASCOTS.2006.46

Yatziv, L., Bartesaghi, A., Sapiro, G., 2006. O(n) implementation of the fast marching algorithm. Journal of computational physics 212 (2), 393–399. doi: http://dx.doi.org/10.1016/j.jcp.2005.08.005

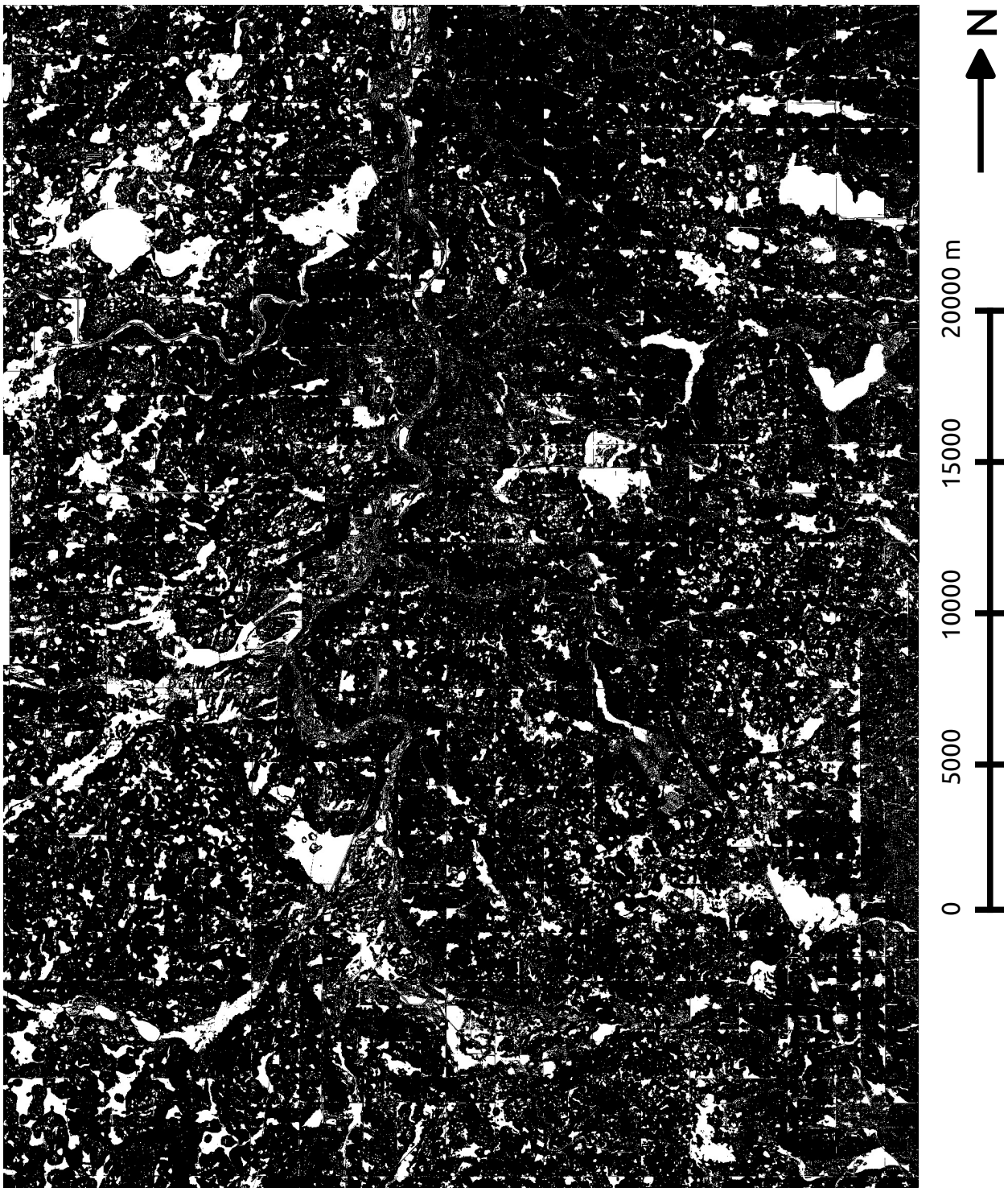


**Figure 6:** Steele County, Minnesota. This is a zoomed-out view of a 10891 x 13914 cell ($152 \cdot 10^6$ total) 3 m DEM. The DEM has been processed to show landscape depressions in white and all other terrain in black. The landscape depressions would be filled to mathematically-level surfaces by Algorithms 1 and 2, converted to near-level draining surfaces by 3, or carved by Algorithm 4.