Computers & Geosciences 62 (2014) 117–127

Contents lists available at ScienceDirect

Computers & Geosciences

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

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

ARTICLE INFO

Article history:
Received 5 October 2012
Received in revised form
19 April 2013
Accepted 25 April 2013
Available online 23 May 2013

Keywords:
Pit filling
Terrain analysis
Hydrology
Drainage network
Modeling
GIS

ABSTRACT

Depressions (or pits) are 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 was impermeable, is often necessary in preprocessing DEMs. The depression-filling algorithm presented here – called Priority-Flood – unifies and improves 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.

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 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 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 in 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

* Corresponding author. Tel.: +1 321 222 7637.
E-mail addresses: rbarnes@umn.edu (R. Barnes), lehman@umn.edu (C. Lehman), mulla003@umn.edu (D. Mulla).

0098-3004/$ - see front matter © 2013 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.cageo.2013.04.024

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 ϵ between any two cells on the path, where ϵ may be zero. Such a path is referred to as an ϵ-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.

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 3×3 neighborhoods 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

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.

Year	Authors	Operation	Data type	Connectedness	Time complexity
1989	Ehlschlaeger	Flow directions, accumulation	Integer/float*	8, Any	$O(n^2)$*
1991	Vincent and Soille	Watershed labels	Integer	Gridded, n-dim	$O(n)$
1992	Beucher and Meyer	Watershed labels	Integer	4, 6, 8	$O(n)$*
1994	Meyer	Watershed labels	Integer*	8*	$O(n)$*
1994	Soille and Gratin	Filling	Integer	4, 6, 8	$O(n)$*
2006	Wang and Liu	Filling	Floating*	8*	$O(n \log_2 n)$
2009	Liu et al.	Filling	Floating*	8	$O(n \log_2 k)$[a]
2010	Metz et al.	Flow directions	Floating*	8*	$O(n \log_2 n)$*
2011	Metz et al.	Flow directions	Floating*	"Gridded"	$O(n \log_2 n)$*
2011	Beucher and 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*	$O(n)$

[a] Liu et al. (2009) claim a $O(8n \log_2 n)$ time complexity, but implement an $O(n \log_2 k)$ algorithm.