Contents lists available at ScienceDirect

Computers & Geosciences

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

Research paper

An efficient variant of the Priority-Flood algorithm for filling depressions in raster digital elevation models

Guiyun Zhou[a,*], Zhongxuan Sun[a], Suhua Fu[b]

[a] School of Resources and Environment, University of Electronic Science and Technology of China, Chengdu 611731, China
[b] State Key Laboratory of Earth Surface Processes and Resource Ecology, School of Geography, Beijing Normal University, Beijing 100875, China

ARTICLE INFO

Article history:
Received 2 June 2015
Received in revised form
12 December 2015
Accepted 26 February 2016
Available online 27 February 2016

Keywords:
Depression filling
Flow direction
DEM
Hydrological analysis

ABSTRACT

Depressions are common features in raster digital elevation models (DEMs) and they are usually filled for the automatic extraction of drainage networks. Among existing algorithms for filling depressions, the Priority-Flood algorithm substantially outperforms other algorithms in terms of both time complexity and memory requirement. The Priority-Flood algorithm uses a priority queue to process cells. This study proposes an efficient variant of the Priority-Flood algorithm, which considerably reduces the number of cells processed by the priority queue by using region-growing procedures to process the majority of cells not within depressions or flat regions. We present three implementations of the proposed variant: two-pass implementation, one-pass implementation and direct implementation. Experiments are conducted on thirty DEMs with a resolution of 3m. All three implementations run faster than existing variants of the algorithm for all tested DEMs. The one-pass implementation runs the fastest and the average speed-up over the fastest existing variant is 44.6%.

1. Introduction

The automatic extraction of drainage networks from raster digital elevation models (DEMs) is required in many scenarios such as soil erosion modeling, hydrological process simulations and geomorphological analyses (Bai et al., 2015; Beskow et al., 2009; Nobre et al., 2011). A raster DEM is a two-dimensional array of elevation values at regularly spaced positions on the ground. A preprocessing step of DEMs that is commonly required for extracting drainage networks is the filling of depressions so that any cell in a DEM has a non-ascending path toward the border of a DEM (Barnes et al., 2014a; Jenson and Domingue, 1988). A depression in a DEM is composed of cells that do not have a non-ascending path toward the border of the DEM. Depressions may result from natural terrains and they may also be the products of processing algorithms used to generate the DEM. In addition to the filling of depressions, depressions may also be resolved by a carving procedure (Martz and Garbrecht, 1999; Soille, 2004).

Three representative algorithms are available for the filling of depressions. The first representative algorithm is proposed by Jenson and Domingue (1988) and extended in many studies (Arge et al., 2003; Martz and Garbrecht, 1999). The second representative algorithm for the filling of depressions is proposed by Planchon and Darboux (2002) and further studied by many researchers (Qin and Zhan, 2012; Rueda et al., 2013). The third representative algorithm is referred to as the Priority-Flood algorithm by Barnes et al. (2014a). The algorithm outperforms the methods of Jenson–Domingue and Planchon–Darboux in terms of both time complexity and memory requirement.

In this paper, we propose an efficient variant of the Priority-Flood algorithm over existing variants, which considerably reduces the running time of the algorithm. The remainder of the paper is organized as follows. Section 2 reviews the various variants of the Priority-Flood algorithm for floating-point DEMs. In Section 3, we propose our variant of the algorithm and present three implementations of the proposed variant. The experimental results of our algorithm are presented in Section 4. We conclude the paper in Section 5.

2. Review of the Priority-Flood algorithm

Barnes et al. (2014a) give a detailed history of the Priority-Flood algorithm and summarize important variants of the algorithm. According to Barnes et al. (2014a), the Priority-Flood algorithm can be traced back to Ehlschlaeger (1989). For an integer DEM, the algorithm has a time complexity of $O(N)$ (Beucher and Beucher, 2011; Gomes et al., 2012; Magalhães et al., 2012; Soille and Gratin, 1994). For DEMs with a floating-point data type, the generic form of the algorithm has a time complexity of $O(N\log N)$. This study

* Corresponding author.
E-mail address: zhouguiyun@uestc.edu.cn (G. Zhou).

http://dx.doi.org/10.1016/j.cageo.2016.02.021
0098-3004/© 2016 Elsevier Ltd. All rights reserved.

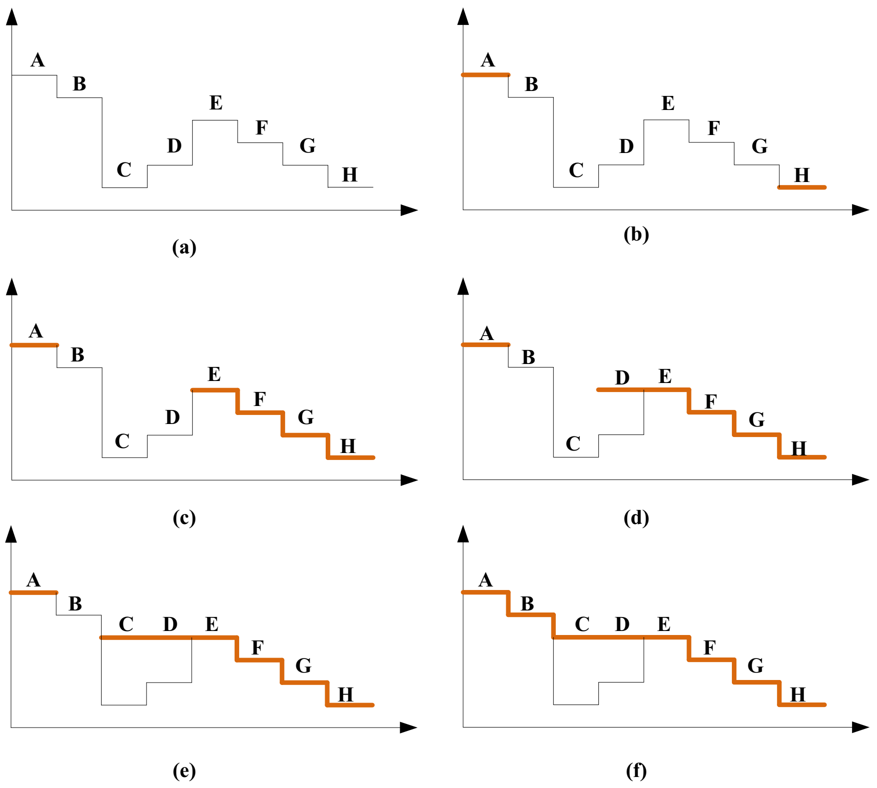

Fig. 1. One dimensional view of the depression-filling process using the Priority-Flood algorithm in Wang and Liu (2006). Processed cells are shown in orange. (a) Original DEM; (b) border cells A and H are pushed into a priority queue PQ; (c) cells E, F and G are processed by PQ; (d) cell D is raised to have the same elevation as E; (e) cell C is raised to have the same elevation as the raised cell D; (f) cell B is processed by PQ. (1.5-column fitting image). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

focuses on floating-point DEMs.

A key concept used in the Priority-Flood algorithm is the spill elevation of a cell, which is the minimum elevation the cell needs to be raised to for it to have a non-ascending path toward the border of the DEM (Wang and Liu, 2006). The key data structure used by the Priority-Flood algorithm for floating-point DEMs is the priority queue PQ, which is used to find the cell with the lowest spill elevation. The priority queue can be implemented in many ways (Bai et al., 2015; Barnes et al., 2014a), which may affect the time complexity of the algorithm.

Wang and Liu (2006) propose the first variant of the Priority-Flood algorithm for floating-point DEMs. Their variant starts from the border cells and processes other cells that are neighbors of processed cells one by one in the ascending order of their spill elevations. Fig. 1 shows the depression-filling scheme of their variant. Barnes et al. (2014a) propose an improved variant of the Priority-Flood algorithm and use a plain queue to process cells in depressions. For the cross section of the DEM surface in Fig. 1a, cells C and D are located within a depression and they can be processed using a plain queue. The variant of Barnes et al. finds all cells in a depression once the spill outlet of the depression is found. This is a typical region-growing process (Region growing, 2015). A region-growing process requires a set of seed cells. The region is grown from the seed cells to adjacent cells depending on a region membership criterion. Cells in the region is marked using a mask matrix of Boolean values. The pseudocode of a generic region-growing procedure is shown in Algorithm 1. In Algorithm 1, the argument *Flag* is the mask matrix and it is both input and output arguments. The input argument *Q* is a collection of seed cells. The input argument *regionMembership* represents the region membership criterion function that takes currently processed cell *c* and its neighbor *n* as its input arguments and returns a Boolean value (Line 8). If *regionMembership* returns true, cell *n* is within the region. Otherwise, cell *n* is not within the region based on its relationship with *c*. The argument *regionCellOps* represents a function that takes *n* and *c* as its arguments and it represents a collection of operations on newly added region cells. *regionCellOps* can be NULL if no operations are applied on newly added cells. The input argument *nonRegionCellOps* represents a function that takes *n* and *c* as its arguments (Line 15) and it represents a collection of operations on *n* and *c* when *n* is determined to be not within the region based on its relationship with *c* (Line 8). If the *nonRegionCellOps* collection is applied to *c*, it is usually applied once within the **for** loop (Line 6). If it is applied to *n*, it is usually applied to each neighbor of *c*. The variable *isProcessed* (Line 5 and 13–14) controls whether *nonRegionCellOps* is applied once or multiple times. If it is applied to each neighbor, *nonRegionCellOps* should change the value of *isProcessed* to false or the variable *isProcessed* should not be used at all in the algorithm. *nonRegionCellOps* can also be NULL.

Algorithm 2 presents a modified version of the improved Priority-Flood algorithm by Barnes et al. (2014a) using an explicit region-growing procedure to process cells in depressions and flat regions. Note that in Algorithm 2, the first two operations in *nonRegionCellOps* are applied to all neighboring cells of *c* that are outside the region based on their relationship with *c*. Algorithm 2 highlights the separation of the processing of cells in depressions and flat regions from the processing of other cells.

Algorithm 1. A generic region-growing procedure. Flag is the mask matrix of the region. Q is the collection containing all seed cells. regionMembership is the region membership criterion. regionCellOps is a collection of operations on newly added region cells. nonRegionCellOps is a collection of operations on non-region cells at current stage.

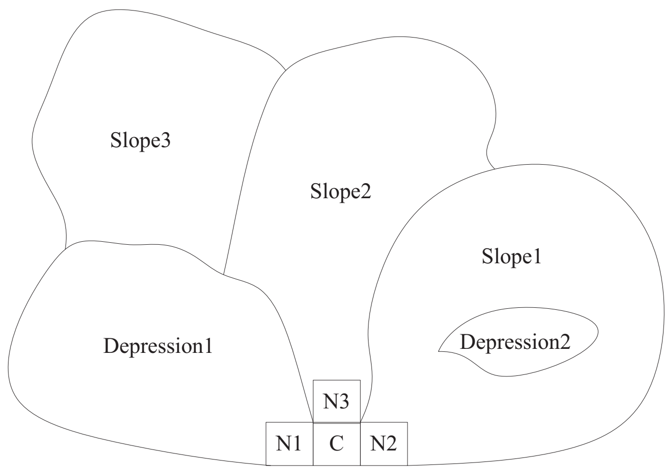

Fig. 2. A schematic diagram for the distribution of slopes and depressions within a DEM. (1-column fitting image).

```
1:  void RegionGrowing(Flag,Q, regionMembership, regionCellOps, nonRegionCellOps) {
2:      Let isProcessed be a Boolean variable;
3:      while (Q is not empty) {
4:          Pop cell c from Q;
5:          isProcessed=false;
6:          for each neighbor n of c {
7:              if (Flag(n)) continue;
8:              if (regionMembership (n,c) is satisfied) {
9:                  Push n into Q;
10:                 Flag(n)=true;
11:                 if (regionCellOps !=NULL) regionCellOps (n,c);
12:             }
13:             else if (!isProcessed) {
14:                     isProcessed=true;
15:                     if (nonRegionCellOps !=NULL) nonRegionCellOps (n,c);
16:             }
17:         }
18:     }
19: }
```

Algorithm 2. The modified version of the variant of the Priority-Flood algorithm by Barnes et al. (2014a). RegionGrowing is the function in Algorithm 1.

```
1:  Let DEM be the input DEM;
2:  Let Flag be a matrix of Boolean values of the size of DEM;
3:  Let PQ be an empty priority queue;
4:  Let Q be an empty plain queue;
5:  Initialize Flag as false;
6:  Push all edge cells in DEM into PQ;
7:  Mark all edge cells in Flag as true;
8:  while (PQ is not empty) {
9:      Pop cell c off PQ;
10:     for each neighboring cell n of c {
11:         if (Flag(n)) continue;
12:         if (DEM(n)<=DEM(c)) {
13:                 Flag(n)=true;
14:                 DEM(n)=DEM(c);
15:                 Push n into Q;
16:                 RegionGrowing(Flag, Q, 'DEM(n)<=DEM(c)', 'DEM(n)=DEM(c);',
17:                                  'Push n into PQ; Flag(n)=true; isProcessed=false;');
18:         }
19:         else {
20:                 Push n into PQ;
21:                 Flag(n)=true;
22:         }
23:     }
24: }
```

3. Proposed variant of the Priority-Flood algorithm

In this section, we propose our variant of the Priority-Flood algorithm. For a DEM, our variant can use plain queues to process the majority of cells not within depressions or flat regions with a time complexity of $O(N)$. This is an important improvement because it considerably reduces the number of cells that need to be processed by the priority queue and reduces the running time of the algorithm.

Our variant of the Priority-Flood algorithm classifies cells in a DEM into two categories: depression cells and slope cells. For a depression cell, its spill elevation is equal to or less than the spill elevation of any of its neighbors. A depression cell can be located in a depression or in a flat region. In Fig. 1a, cells C and D are depression cells. For a slope cell, its spill elevation is greater than the spill elevation of at least one of its neighbors. The region formed by connected slope cells is called a slope in this study. In Fig. 1a, cells A, B, E, F, G and H are slope cells. As shown in Fig. 2, the spatial relationship among depressions and slopes can be very complex.

Slope cells can be classified into two types: interior slope cells and boundary slope cells. An interior slope cell of a slope does not border any other slopes or other depressions. A boundary slope cell c of a slope has at least one neighboring cell n that is not within the slope. Only boundary slope cells can be the spill outlets of depressions. In the Priority-Flood algorithm, cells are processed in the ascending order of their spill elevations. When a boundary slope cell c is processed before its neighbor n in a neighboring region, cell c is called a potential spill cell because it may be the spill outlet of an unprocessed depression. All cells on the border of a DEM are potential spill cells because the outside region of a DEM is treated as a very low depression in the Priority-Flood algorithm. In Fig. 1a, cells F and G are interior slope cells. Cells A, E and H are potential spill cells. Cell B is not a potential spill cell as it is processed after the depression where cell C and D are located is filled.

Fig. 3 shows the distribution of different types of cells in a sample DEM. It can be seen that it is very common for a slope to be adjacent to another slope or depression in a DEM. In the sample DEM, potential spill cells only account for a small portion of the total number of cells.

In the Priority-Flood algorithm, the potential spill cells need to be processed by PQ to find the cell with the lowest spill elevation. All other slope cells do not need to be processed by PQ. Based on this finding, we propose a new variant of the Priority-Flood algorithm. The flowchart of our variant of the Priority-Flood algorithm is shown in Fig. 4. Our proposed variant has the basic structure of the Priority-Flood algorithm. In our variant, two different region-growing procedures are applied to depression cells and slope cells respectively. Once a depression is filled, those unprocessed slope cells that border the depression are used as seed cells to trace more slope cells. Potential spill cells of the slopes are identified and pushed into PQ for processing. Apparently, the critical part of our variant of the Priority-Flood algorithm is to identify the potential spill cells in a slope. In this section, we provide three

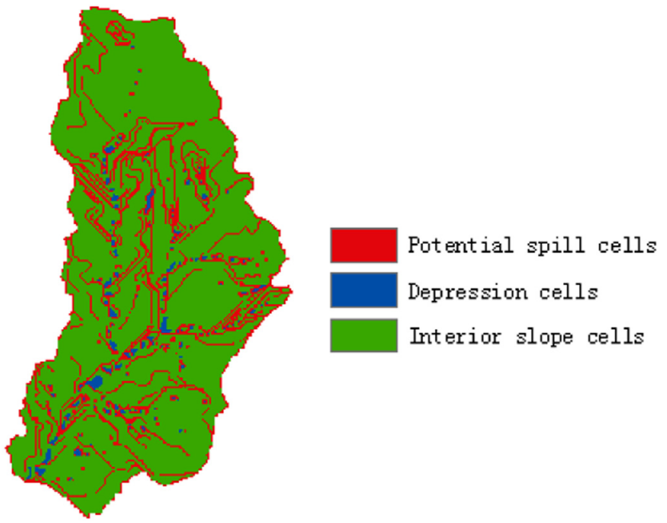

Fig. 3. Distribution of different types of cells in a sample DEM. (1-column fitting image).

implementations of our variant to identify the potential spill cells with different time complexity and implementation complexity.

3.1. *Two-pass implementation*

This implementation pushes only potential spill cells into PQ so as to minimize the number of cells processed by PQ. To find the potential spill cells of a slope, we need to find all the boundary cells of the slope that have neighbors in other unprocessed regions. Because a slope is found by tracing cells from a list of seed slope cells, the potential spill cells can be found by applying a second region-growing process from the same seed cells. The pseudocode of the two-pass implementation is shown in Algorithm 3. Given an initial list of seed slope cells, the first region-growing process uses a plain queue to trace all other unprocessed slope cells and the main Boolean matrix is used as the mask matrix (Line 25). The second region-growing process uses another plain queue to identify the potential spill cells in the traced slope and the second Boolean matrix is used as the mask matrix (Line 26). The potential spill cells are pushed into PQ for further processing. Note that in the second region-growing procedure, the currently processed boundary slope cell *c* is pushed only once into PQ by not changing the value of the variable *isProcessed* to be false (Line 25).

A worked example of the two-pass implementation of our proposed variant is given in Fig. 5. In the example, we use the same sample dataset as in Wang and Liu (2006) to better illustrate the differences between the two variants of the Priority-Flood algorithm. First, all border cells are pushed into PQ and marked as processed. The lowest cell in PQ, D7, is popped off (Fig. 5c). We process all unprocessed neighbors of a center cell in a clockwise order and start from the neighbor to the right of the center cell. D6 is first processed during the iteration of the neighboring cells of D7. Since D6 is higher than D7, it is a seed slope cell and the slope contains D6 is traced using a region-growing procedure (Fig. 5e). Another region-growing procedure is applied to find the potential spill cells of the traced slope using the same seed cell D6. The three cells, B4, C4 and D4, are the potential spill cells and pushed into PQ (Fig. 5f). A6 and C4 are popped off PQ in turn as they are the two lowest cells (Fig. 5g). All the neighbors of A6 have been processed. For C4, its neighbor D3 is first processed. Since D3 is lower than C4, a depression is encountered and C4 is the spill outlet of the depression. The depression is processed using a region-growing procedure and all cells in the depression are marked as processed (Fig. 5h). We also find the unprocessed slope cells bordering the depression during the region-growing procedure. These slope cells include B2, C2 and D2 and they are used as seed cells to trace other slope cells. Since all cells in the DEM have been processed, the process does not trace any new slope cells. All remaining cells in PQ are popped off in turn and no new region-growing process is started.

Algorithm 3. The two-pass implementation of the proposed variant.

```
1:  Let DEM be the input DEM;
2:  Let Flag and Flag 2 be two matrices of Boolean value of the
    size of DEM;
3:  Let PQ be an empty priority queue;
4:  Let SQ and Q be two empty plain queues;
5:  Initialize Flag and Flag 2 as false;
6:  Push all edge cells in DEM into PQ;
7:  Mark all edge cells in Flag and Flag2 as true;
8:  while (PQ is not empty) {
9:          Pop cell c off PQ;
10:         for each neighboring cell n of c {
11:                 if (Flag(c)) continue;
12:                 if (DEM(n) < =DEM(c)) {
13:                         Flag(n)=Flag2(n)=true;
14:                         DEM(n)=DEM(c);
15:                         Push n into Q;
16:                         RegionGrowing(Flag, Q, 'DEM
    (n) < =DEM(c)', 'DEM(n)=DEM(c);
17:                                                     Flag2(n) =true',
    'Push n into SQ; Flag(n)=Flag2(n)=true;
18:                                                     isProcessed=false;');
19:                 }
20:                 else {
21:                         Push n into SQ;
22:                         Flag(n)=Flag2(n)=true;
23:                 }
24:                 Copy all cells from SQ to Q;
25:                 RegionGrowing(Flag, SQ, 'DEM(n) > DEM(c)',
    NULL, NULL );
26:                 RegionGrowing(Flag2, Q, 'Flag(n)==true',
    NULL, 'Push c into PQ;');
27:         }
28: }
```

3.2. *One-pass implementation*

The two-pass implementation of our proposed variant reduces the number of cells pushed into PQ to the minimum. However, it requires two region-growing procedures to process slope cells. The one-pass implementation aims to balance the time complexity for processing PQ and slope cells using only one region-growing process. A boundary slope cell *c* is a potential spill cell if it has one unprocessed neighbor in another region, which is lower than or equal to *c*. If an unprocessed neighbor *n* of a cell *c* has a processed lower neighbor *j*, *n* can be traced as a slope cell from *j*. In this case, *n* should not be counted as an unprocessed neighbor of *c*. If none of the neighbors of *n* are processed or lower than *n*, *c* may be a potential spill cell and is pushed into PQ in the one-pass implementation of our proposed variant. Note that *c* may not be a true potential spill cell because *n* may still have a lower neighbor that can be processed later on in the same round of region-growing process. In this implementation, only one region-growing process is required. The price paid is that some cells that are not potential spill cells are pushed into PQ. The pseudocode of the one-pass implementation is shown in Algorithm 4.

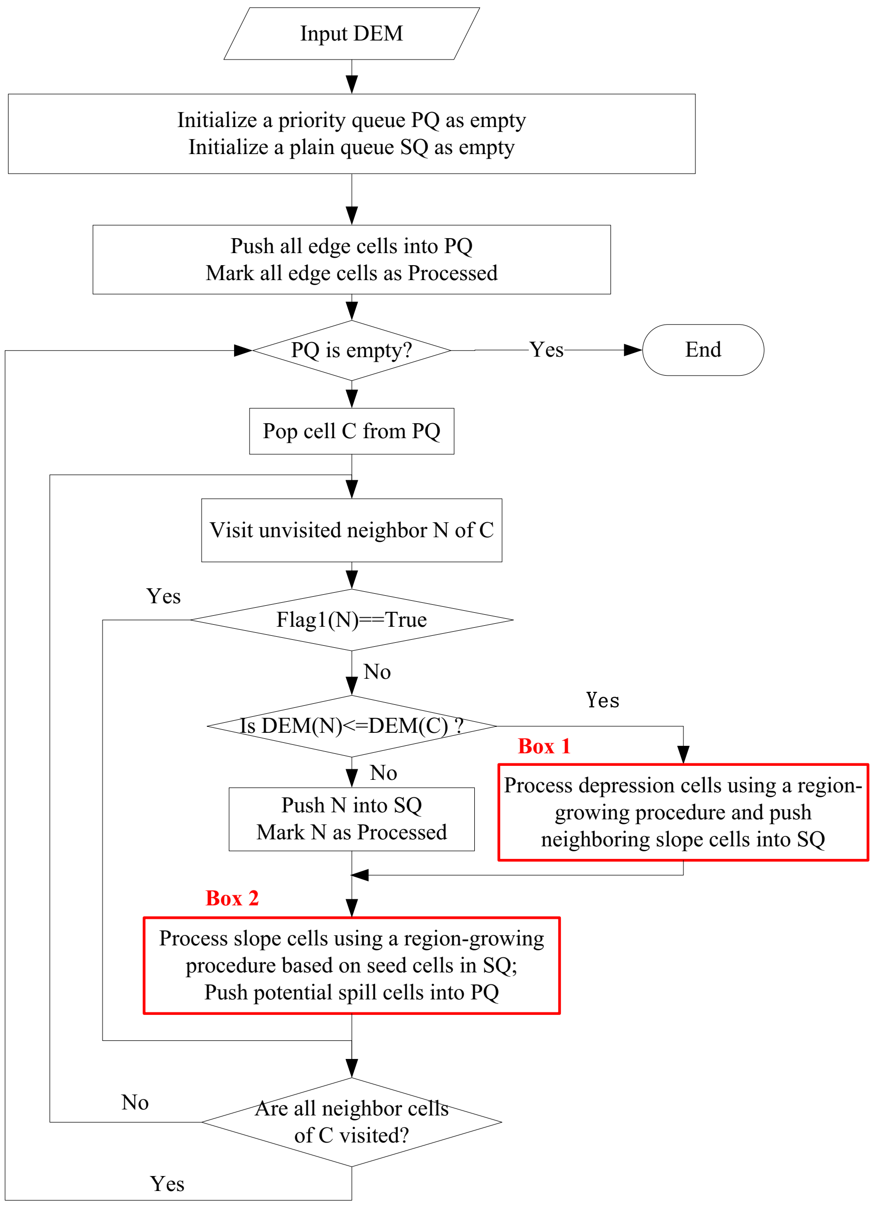
Input DEM
Initialize a priority queue PQ as empty
Initialize a plain queue SQ as empty
Push all edge cells into PQ
Mark all edge cells as Processed
PQ is empty?
Yes
End
Pop cell C from PQ
Visit unvisited neighbor N of C
Yes
Flag1(N)==True
No
Yes
Is DEM(N)<=DEM(C) ?
Box 1
Process depression cells using a region-growing procedure and push neighboring slope cells into SQ
No
Push N into SQ
Mark N as Processed
Box 2
Process slope cells using a region-growing procedure based on seed cells in SQ;
Push potential spill cells into PQ
No
Are all neighbor cells of C visited?
Yes

Fig. 4. The flowchart of the proposed variant of the Priority-Flood algorithm. Two different procedures, shown in Box1 and Box2, are applied to depression cells and slope cells, respectively. (1.5-column fitting image).

A worked example of the one-pass implementation is shown in Fig. 6. The sample DEM used in Fig. 6 is used again. A plain queue SQ is used to process slope cells. When the seed slope cell D6 is found, its three unprocessed neighbors, D5, C5 and C6, are traced (Fig. 6c) and pushed into SQ. D5 is popped of SQ. The first neighbor of D5 is D4, which is higher than D5 and pushed into SQ (Fig. 6d). The second neighbor of D5 is C4 and is lower than D5. Although C4 is unprocessed and lower than D5, C4 has a neighbor C5, which is lower than C4 and processed. Therefore, C4 is a slope cell and D5 is not pushed into PQ. The next cell popped off SQ is C5 and all its unprocessed neighbors are higher than C5 and pushed into SQ. When D4 is popped off SQ, because its two unprocessed neighbors are both lower than D4 and do not have processed lower neighbors, D4 is a potential spill cell and pushed into PQ (Fig. 6f). In a similar way, B4 and C4 are pushed into PQ. The region-growing process continues until SQ is empty. In this example, all potential spill cells of the slope are identified within one round of region-growing process and the cells pushed into PQ are true potential spill cells. In other cases, some cells that are not potential spill cells may be pushed into PQ. For example, if the elevations of C5 and C6 are exchanged and the elevation of B5 is changed to 10, the potential spill cells do not change but D5 will be pushed into PQ although it is not a potential spill cell.

Algorithm 4. The one-pass implementation of our proposed variant.

```
1:  Let DEM be the input DEM;
2:  Let Flag be a matrix of Boolean value of the size of DEM;
3:  Let PQ be a empty priority queue;
4:  Let Q and SQ be two empty plain queues;
5:  Push all edge cells in DEM into PQ;
6:  Mark all edge cells in Flag as true;
7:  Initialize Flag as false;
8:  while (PQ is not empty) {
9:      Pop cell c off PQ;
10:     for each neighboring cell $n$ of $c$ {
11:         if (Flag(c)) continue;
12:         if (DEM(n) < =DEM(c)) {
```

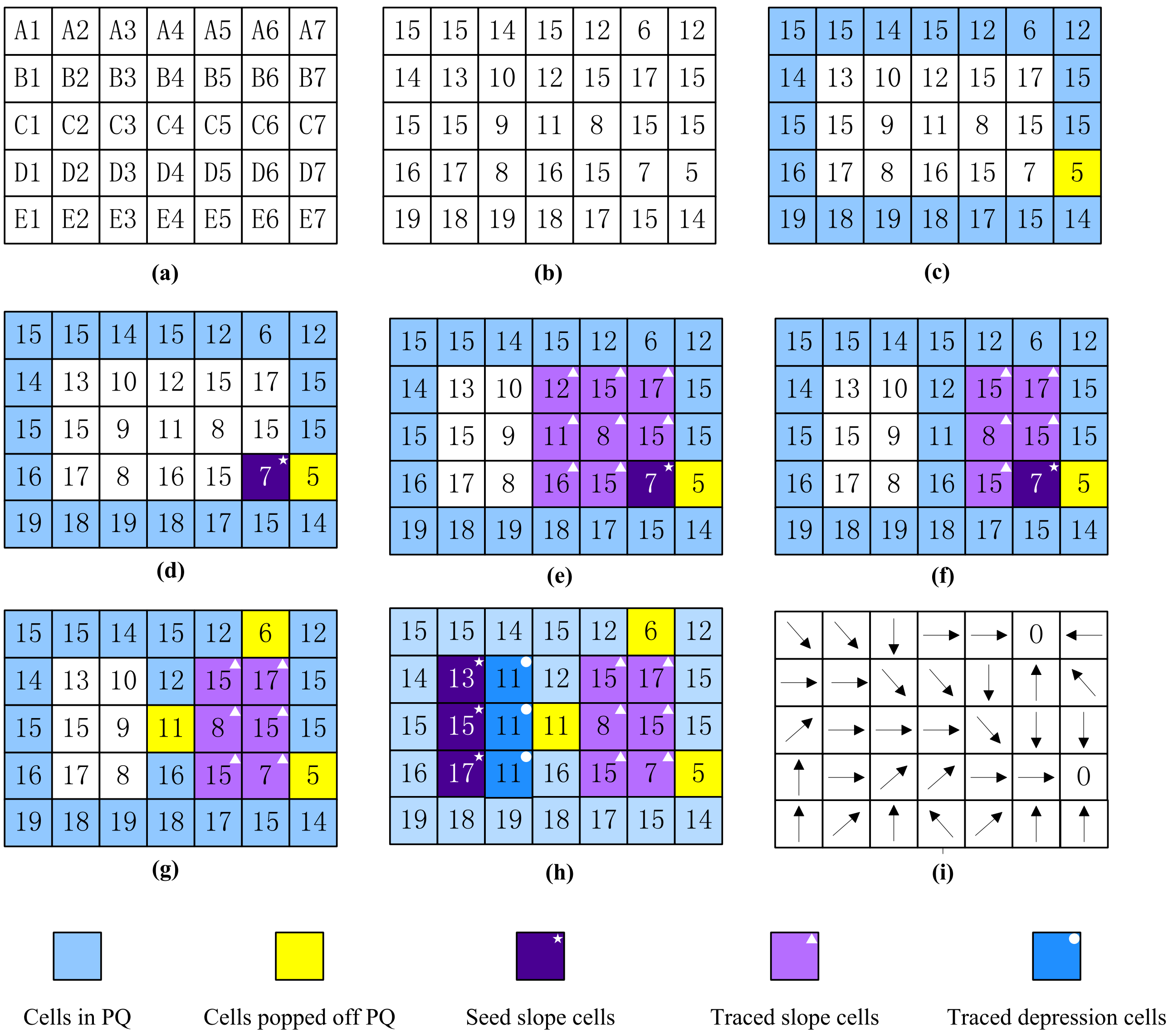

Fig. 5. A worked example of the two-pass implementation of the proposed variant. The sample dataset in Wang and Liu (2006) is used in the example. PQ is the priority queue used in the algorithm. (a) A sample DEM with labeled cells; (b) DEM with elevation values; (c) border cells are pushed into PQ and the lowest cell is popped off; (d) a seed slope cell is identified; (e) all unprocessed slope cells are traced in the first region-growing process; (f) potential spill cells are identified in the second region-growing process and pushed into PQ; (g) two lowest cells are popped from PQ; (h) depression cells are processed; a new list of seed slope cells are identified; (i) flow direction matrix of the DEM derived using the method in Wang and Liu (2006). (2-column fitting image).

```
                        Flag(n)=true;
                        DEM(n)=DEM(c);
                        Push n into Q;
                        RegionGrowing(Flag, Q, ‘DEM(n)< =DEM(c)’, ‘DEM(n)=DEM(c)’, ‘Push n into SQ; Flag(n)=true; isProcessed=false;’);
                  }
                  **else** {
                        Push n into SQ
                        Flag(n)=true;
                  }
                  RegionGrowing(Flag, SQ, ‘DEM(n)>DEM(c)’, NULL, nonRegionCell);
            }
      }

      **Operations in nonRegionCell**
            isBoundary=true;
            **for** each neighboring cell **j** of **n** {
                  **if** (Flag(j) && DEM(j)<DEM(n)) {
                        isBoundary=false;
                        **break**;
                  }
            }
            **if** (isBoundary)
                  Push c into PQ;
            **else**
                  isProcessed =false;
```

3.3. *Direct implementation*

The direct implementation is a simplified implementation of our proposed variant. We present this implementation for comparison purposes. The pseudocode of the direct implementation is shown in Algorithm 5. In this implementation, when an

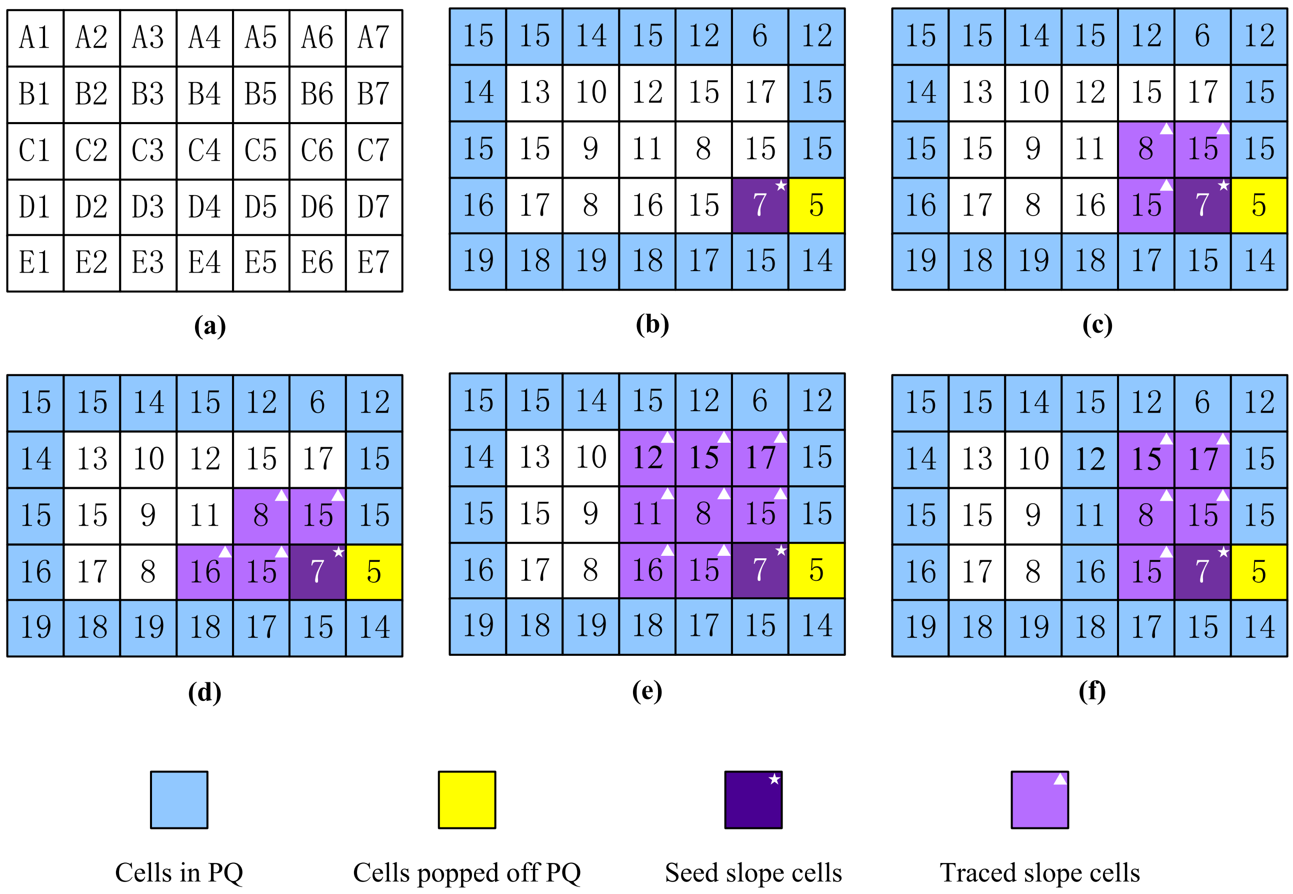

Fig. 6. A worked example of the one-pass implementation. (a) A sample DEM with labeled cells; (b) a seed slope cell is identified; (c) neighboring slope cells are traced from D6; (d) D4 is traced from D5. C4 is not a slope cell when it is compared with D5 because C4 is higher than the processed cell C5. D5 is not pushed into PQ; (e) C4, B4, B5 and B6 are traced from C5. (f). D4, C4 and B4 are identified as potential spill cells. (2-column fitting image).

unprocessed neighboring cell *n of cell c* is lower than c in the region-growing procedure, *c* is immediately pushed into PQ. Compared to the one-pass implementation, it increases the number of cells pushed into PQ. In the example in Fig. 6, D5 will immediately be pushed into PQ because its neighbor C4 is unprocessed and lower than D5.

Algorithm 5. The direct implementation of our proposed variant.
```
1:  Let DEM be the input DEM
2:  Let Flag be a matrix of Boolean value of the size of DEM
3:  Let PQ be a empty priority queue
4:  Let Q and SQ be two empty plain queues
5:  Push all edge cells in DEM into PQ
6:  Mark all edge cells in Flag as true
7:  Initialize Flag as false
8:  while (PQ is not empty) {
9:          Pop cell c off PQ;
10:         for each neighboring cell n of c {
11:                 if (Flag(c)) continue;
12:                 if (n is lower than or equals c) {
13:                         Flag(n)=true;
14:                         DEM(n)=DEM(c);
15:                         Push n into Q
16:                         RegionGrowing(Flag, Q, 'DEM
    (n)<=DEM(c)', 'DEM(n)=DEM(c)',
17:                                               'Push n into SQ;
    Flag(n)=true; isProcessed=false;');
18:                 }
19:                 else {
20:                         push n into SQ
21:                                 Flag(n)=true;
22:                         }
23:                                 RegionGrowing(Flag, SQ, 'DEM
    (n)>DEM(c)', NULL, 'Push c into PQ;');
24:                 }
25: }
```

3.4. Time complexity and memory requirement

Our variant improves the existing variants of the Priority-Flood algorithm by processing slope cells using a plain queue instead of a priority queue. The time complexity of our method is $O(N\log N)$, where N is the number of cells pushed into PQ. In our variant, N is much smaller than the number of slope cells.

The two-pass implementation of our variant pushes only potential spill cells into PQ. It requires one additional region-growing procedure to find all potential spill cells. In terms of the memory requirement, it requires one additional Boolean array to hold the processed flag in the second region-growing process to trace the potential spill cells in a slope area. The Boolean array can be efficiently implemented using a bit array, which requires $(N+7)/8$ bytes. The one-pass implementation of our variant pushes not only potential spill cells but also some interior slope cells into PQ. It requires only one region-growing procedure. The direct implementation of our variant pushes more interior slope cells into PQ than the one-pass implementation. All implementations of our variant use plain queues to trace slope cells and depression cells. The maximum number of cells in these queues generally is much smaller than the total number of cells in a DEM and the queues are emptied many times during the region-growing processes.

3.5. Flow direction computation

Wang and Liu (2006) present a method to derive the flow direction matrix during the processing of the cells in the Priority-Flood algorithm. The flow direction of a depression cell is assigned by reversing the search direction. For slope cells, their flow directions are determined by identifying the steepest descent direction using the *D8* method (Jenson and Domingue, 1988). The flow direction matrix can be derived in the same way when our variant is used to fill depressions. Fig. 5i shows the flow direction matrix derived for the sample dataset using the above method. There exist many other ways to derive flow direction matrices (Barnes et al., 2014b; Garbrecht and Martz, 1997; Metz et al., 2011; Nardi et al., 2008).

4. Experimental results

Five depression-filling algorithms, including the variant of Wang and Liu (2006), the variant of Barnes et al. (2014a) and the three implementations of our proposed variant, are implemented in C++. The priority queue used in the algorithms is provided by the C++ standard template library. The source codes are available at GitHub (https://github.com/zhouguiyun-uestc/FillDEM). The LiDAR-based DEMs of thirty counties in the state of Minnesota, USA, are downloaded from the FTP site operated by the Minnesota Geospatial Information Office (http://www.mngeo.state.mn.us/). The first 30 counties in Minnesota in alphabetic order are chosen for the experiments to avoid selection bias. On average, each county contains approximately 3.96×10^8 cells. All tests are run on a 64-bit Windows 7 operating system with an Intel Xeon E5-2620 2.0GHz processor and 56GB RAM.

The five algorithms produce the same depression-filled DEMs for each tested DEM. The running times of the five algorithms for all test DEMs are listed in Table 1. Table 1 also lists the width, height and the percentage of NODATA cells of each DEM. Fig. 7 plots the running times of the five algorithms. The average running time is 180 s for the variant of Wang and Liu (2006), 166 s for the variant of Barnes et al. (2014a), 103 s for the two-pass implementation, 92 s for the one-pass implementation and 111 s for the direct implementation. Suppose that the speed-up of algorithm A over B is defined as the difference of running times of the two algorithms divided by the running time of B. The average speed-up of the variant of Barnes et al. over the variant of Wang and Liu is 7.8%. This value is lower than the average speed-up value of 16.8% reported in Barnes et al. (2014a). Considering that more DEMs are tested in Barnes et al. (2014a) and that details of the implementations may differ slightly, the difference between the two speed-up values is acceptable. All three implementations of our proposed variant run substantially faster than the variant of Barnes et al. for all tested DEMs. The average speed-up over the variant of Barnes et al. is 38.0% for the two-pass implementation, 44.6% for the one-pass implementation and 33.1% for the direct implementation. For all tested DEMs, the one-pass implementation runs the fastest than all other four algorithms. While the two-pass implementation runs faster than the direct implementation on average, the direct implementation takes less time for some DEMs. Apparently, the actual running time of the one-pass and direct implementations of our proposed variant depends not only on the percentage of slope cells but also on their spatial distribution. The way in which the priority queue is implemented affects the time complexity of all variants of the Priority-Flood algorithm and our variants should outperform existing variants as long as the priority queue is implemented in the same way.

Table 1
List of running times (seconds) of five depression-filling algorithms for 3-m LiDAR DEM data of 30 counties in Minnesota, USA. The width, height and percentage of NODATA cells of each DEM are also listed. The abbreviation of 'IMPL.' stands for 'implementation'.

County	Width	Height	NODATA percentage	Variant in Wang and Liu	Variant in Barnes et al.	Two-pass IMPL.	One-pass IMPL.	Direct IMPL.
Aitkin	20,133	33,681	13.23	421	374	295	257	282
Anoka	13,938	15,116	21.39	95	83	52	46	54
Becker	27,160	18,307	7.64	298	285	136	126	160
Beltrami	30,382	43,654	29.38	538	468	424	361	383
Benton	16,252	11,823	19.69	77	71	51	45	54
Big Stone	19,409	17,116	43.90	101	91	59	52	63
Blue Earth	16,897	16,427	11.62	147	136	77	71	86
Brown	21,037	15,412	31.17	115	103	64	55	68
Carlton	20,845	14,007	2.80	155	143	121	108	121
Carver	14,147	14,035	28.18	84	77	37	35	45
Cass	27,393	46,261	39.62	502	480	302	259	294
Chippewa	22,228	15,658	40.88	121	110	66	60	72
Chisago	13,022	16,204	38.32	71	64	44	39	46
Clay	17,555	20,476	8.13	153	131	107	91	103
Cleanwater	11,003	33,934	9.29	182	167	118	105	123
Cook	39,834	30,262	55.19	297	281	163	148	191
Cottonwood	17,048	14,409	13.66	113	103	61	56	75
Crow Wing	16,138	24,217	14.88	231	197	142	126	140
Dakota	16,561	17,394	30.89	127	119	59	54	75
Dodge	10,886	13,903	5.61	78	76	44	39	54
Douglas	17,476	14,495	5.48	149	136	63	57	72
Faribault	17,029	15,330	6.21	133	123	70	62	76
Fillmore	20,301	14,103	2.17	211	202	78	77	125
Freeborn	16,857	15,110	9.32	154	133	71	64	82
Goodhue	22,425	19,720	38.44	198	196	75	69	106
Grant	14,307	14,458	7.03	96	86	56	50	59
Hennepin	16,492	17,438	27.17	123	120	62	57	71
Houston	15,368	14,108	13.63	146	144	51	50	86
Hubbard	14,068	24,673	5.48	203	185	109	100	120
Isanti	13,893	12,810	13.27	92	81	47	43	50

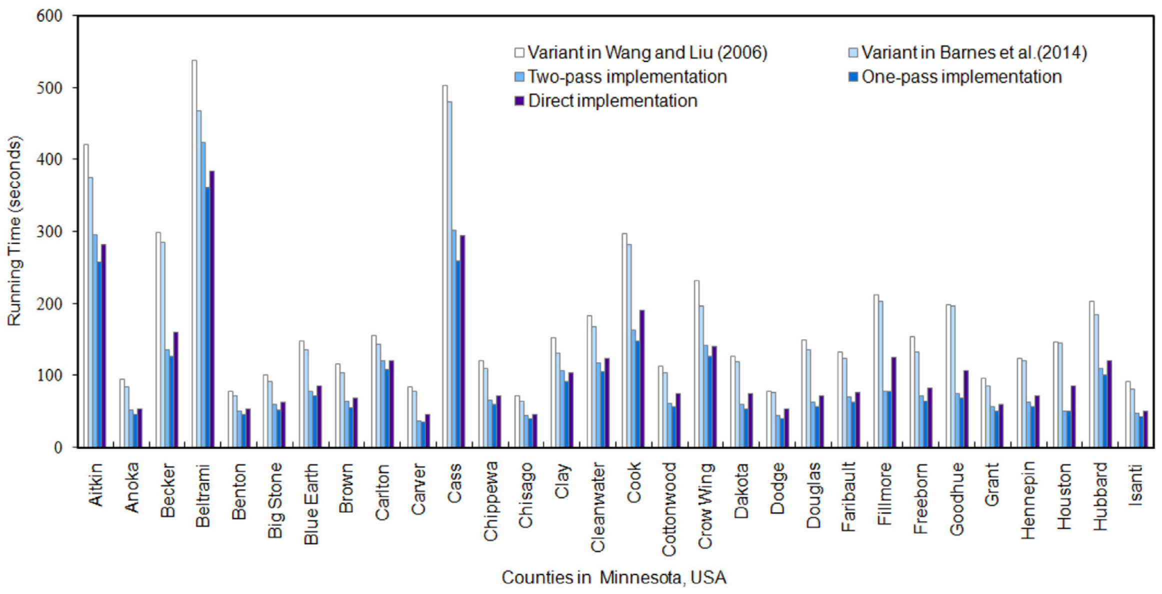

Fig. 7. Running time (seconds) of five depression-filling algorithms for 3-m LiDAR-based DEM data of 30 counties in Minnesota, USA. (2-column fitting image).

5. Conclusion

Among existing algorithms for filling depressions, the Priority-Flood algorithm substantially outperforms other algorithms in terms of both time complexity and memory requirement. In this study, we propose an efficient variant of the Priority-Flood algorithm, which uses region-growing procedures to process the majority of slope cells with a time complexity of $O(N)$. The proposed variant aims to minimize the number of cells processed by the priority queue, which has a time complexity of $O(N\log N)$. We present three implementations of the proposed variant: two-pass implementation, one-pass implementation and direct implementation. Five depression-filling algorithms, including the variant of Wang and Liu (2006), the variant of Barnes et al. (2014a) and the three implementations of our proposed variant, are implemented in C++. Experiments are conducted on 3-m LiDAR-based DEMs of thirty counties in Minnesota, USA. All three implementations run faster than existing variants of the algorithm for all tested DEMs. The one-pass implementation runs the fastest and the average speed-up over existing variants is 44.6%.

All existing variants of the Priority-Flood algorithm do not have parallel implementations. In future work, we will develop a parallel computation scheme for the proposed variant.

Acknowledgments

This work was supported by the National Natural Science Foundation of China (41371399 and 41371341) and the Chinese State Key Basic Research Project (2013CB733406). We are grateful to our colleague, Professor Binbin He, for providing the test environment. We thank the anonymous referees for their constructive criticism and comments.

References

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. http://dx.doi.org/10.1023/a:1025526421410.

Bai, R., Li, T., Huang, Y., Li, J., Wang, G., 2015. An efficient and comprehensive method for drainage network extraction from DEM with billions of pixels using a size-balanced binary search tree. Geomorphology 238 (0), 56–67. http://dx.doi.org/10.1016/j.geomorph.2015.02.028.

Barnes, R., Lehman, C., Mulla, D., 2014a. Priority-flood: an optimal depression-filling and watershed-labeling algorithm for digital elevation models. Comput. Geosci. 62 (0), 117–127. http://dx.doi.org/10.1016/j.cageo.2013.04.024.

Barnes, R., Lehman, C., Mulla, D., 2014b. An efficient assignment of drainage direction over flat surfaces in raster digital elevation models. Comput. Geosci. 62 (0), 128–135. http://dx.doi.org/10.1016/j.cageo.2013.01.009.

Beskow, S., Mello, C.R., Norton, L.D., Curi, N., Viola, M.R., Avanzi, J.C., 2009. Soil erosion prediction in the Grande River Basin, Brazil using distributed modeling. CATENA 79 (1), 49–59. http://dx.doi.org/10.1016/j.catena.2009.05.010.

Beucher, N., Beucher, S., 2011. Hierarchical Queues: General Description and Implementation in MAMBA Image Library [online]. Available from 〈http://cmm.ensmp.fr/~beucher/publi/HQ_algo_desc.pdf〉 (accessed 21.04.15.).

Ehlschlaeger, C., 1989. Using the AT search algorithm to develop hydrologic models. Int. Geogr. Inf. Syst. (IGIS) Symp. 89, 275–281.

Garbrecht, J., Martz, L.W., 1997. The assignment of drainage direction over flat surfaces in raster digital elevation models. J. Hydrol. 193 (1–4), 204–213. http://dx.doi.org/10.1016/S0022-1694(96)03138-1.

Gomes, T.L., Magalhes, G.V.S., Andrade, A.V.M., Franklin, R.W., Pena, C.G., 2012. Computing the drainage network on huge grid terrains. In: Proceedings of the 1st ACM SIGSPATIAL International Workshop on Analytics for Big Geospatial Data. ACM. New York, NY, USA, pp. 53–60. 〈http://dx.doi.org/10.1145/2447481.2447488〉.

Jenson, S.K., Domingue, J.O., 1988. Extracting topographic structure from digital elevation data for geographic information system analysis. Photogramm. Eng. Remote Sens. 54 (11), 1593–1600.

Magalhães, S.G., Andrade, M.A., Randolph, F.W., Pena, G., 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. (Eds.), Bridging the Geographic Information Sciences. Springer, Berlin Heidelberg, pp. 391–407. http://dx.doi.org/10.1007/978-3-642-29063-3_21.

Martz, L.W., Garbrecht, J., 1999. An outlet breaching algorithm for the treatment of closed depressions in a raster DEM. Comput. Geosci. 25 (7), 835–844. http://dx.doi.org/10.1016/S0098-3004(99)00018-7.

Metz, M., Mitasova, H., Harmon, R.S., 2011. Efficient extraction of drainage networks from massive, radar-based elevation models with least cost path search. Hydrol. Earth Syst. Sci. 15 (2), 667–678. http://dx.doi.org/10.5194/hess-15-667-2011.

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. Hydrol. Sci. J. 53 (6), 1176–1193. http://dx.doi.org/10.1623/hysj.53.6.1176.

Nobre, A.D., Cuartas, L.A., Hodnett, M., Rennó, C.D., Rodrigues, G., Silveira, A., Waterloo, M., Saleska, S., 2011. Height above the nearest drainage – a hydrologically relevant new terrain model. J. Hydrol. 404 (1–2), 13–29. http://dx.doi.org/10.1016/j.jhydrol.2011.03.051.

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. http://dx.doi.org/10.1016/S0341-8162(01)00164-3.

Qin, C.Z., Zhan, L., 2012. Parallelizing flow-accumulation calculations on graphics processing units—from iterative DEM preprocessing algorithm to recursive multiple-flow-direction algorithm. Comput. Geosci. 43 (0), 7–16. http://dx.doi.

org/10.1016/j.cageo.2012.02.022.

Region growing, 2015. Region Growing [online]. Available from ⟨http://en.wikipedia.org/wiki/Region_growing⟩ (accessed 21.04.15.).

Rueda, A., Noguera, J.M., Martínez-Cruz, C., 2013. A flooding algorithm for extracting drainage networks from unprocessed digital elevation models. Comput. Geosci. 59 (0), 116–123. http://dx.doi.org/10.1016/j.cageo.2013.06.001.

Soille, P., 2004. Optimal removal of spurious pits in grid digital elevation models. Water Resour. Res. 40 (12), W12509. http://dx.doi.org/10.1029/2004WR003060.

Soille, P., Gratin, C., 1994. An efficient algorithm for drainage network extraction on DEMs. J. Vis. Commun. Image Represent. 5 (2), 181–189. http://dx.doi.org/10.1006/jvci.1994.1017.

Wang, L., Liu, H., 2006. An efficient method for identifying and filling surface depressions in digital elevation models for hydrologic analysis and modelling. Int. J. Geogr. Inf. Sci. 20 (2), 193–213. http://dx.doi.org/10.1080/13658810500433453.