A linear time algorithm to compute the drainage network on grid terrains

Salles V. G. Magalhães, Marcus V. A. Andrade, W. Randolph Franklin and Guilherme C. Pena

ABSTRACT

We present a new and faster internal memory method to compute the drainage network, that is, the flow direction and accumulation on terrains represented by raster elevation matrices. The main idea is to surround the terrain by water (as an island) and then to raise the outside water level step by step, with depressions filled when the water reaches their boundary. This process avoids the very time-consuming depression filling step used by most of the methods to compute flow routing, that is, the flow direction and accumulated flow. The execution time of our method is very fast, and linear in the terrain size. Tests have shown that our method can process huge terrains more than 100 times faster than other recent methods.

Key words | flow accumulation, flow direction, hydrology, terrain modeling

Salles V. G. Magalhães
Marcus V. A. Andrade (corresponding author)
Guilherme C. Pena
Department of Informatics (DPI),
Universidade Federale de Viçosa,
Viçosa,
Brazil
E-mail: *marcus.ufv@gmail.com*

W. Randolph Franklin
ECSE – Rensselaer Polytechnic Institute,
Troy, NY,
USA

INTRODUCTION

An important component of terrain analysis in geographic information systems (GIS) is the computation of hydrologic structures such as flow direction and accumulated flow. These structures are usually extracted from digital elevation models (DEMs) stored as matrices. The critical issues for computing these elements are assigning directions over flat areas and processing depressions. Traditionally, the depressions are removed by increasing the cell's elevation to the minimal elevation of the cells on the depression boundary. Then the flow over the flat area is directed to the lowest cells on the flat area boundary. All that is very time-consuming. The long execution time of those operations is more critical now because of many huge terrains available covering broad regions with very high resolution, from STRM (SRTM 2011) and IFSARE (Jakowatz *et al.* 1996).

As described by Metz *et al.* (2011), even with recent significant advances in flow routing algorithms, accurate extraction of drainage networks from DEMs remains challenging. The problems are the depressions and flat areas arising during the DEM generation, from interpolation errors or the limited spatial resolution used. Usually they arise because of the interference during the elevation mapping and the majority of depressions are spurious. Thus, they need to be removed before the flow routing computation.

There are many methods for handling depressions and flat areas. Most of them (O'Callaghan & Mark 1984; Tarboton 1997; Planchon & Darboux 2002; Arge *et al.* 2003; Zhu *et al.* 2006; Wang & Liu 2006; Danner *et al.* 2007; Yong-he *et al.* 2009) remove depressions by increasing the cell's elevation and then direct the flow over the flat area to the lowest cells on the flat area boundary. Some others (Grimaldi *et al.* 2007; Santini *et al.* 2009) after sink filling, force a flow direction by creating a gradient in the flat areas.

Those strategies, which modify the elevation values to remove the depressions, assume that they are artifacts introduced during the digital model generation. Recently, Metz *et al.* (2011) described an alternative approach based on Ehlschlaeger (1989) to handle depressions using the least cost drainage paths where the elevation values are not changed. This method is implemented in GRASS (GRASS 2011), an open source/free general purpose geographical information system.

doi: 10.2166/hydro.2013.068

In this paper, we present a new and faster terrain flow computation method that surrounds the terrain by water (as an island) and then raises the outside water level step by step filling the depressions when the water reaches their boundary. The implementation execution time of our method is very fast, and linear in the terrain size. It was tested against some other methods, both classic and recent, such as ArcGIS and GRASS modules *r. watershed* (Metz *et al.* 2011) and *r. terraflow* (Arge *et al.* 2003). As the tests have shown, our method can process huge terrains more than 100 times faster than existing methods.

THE PROPOSED ALGORITHM

The basic idea of the proposed algorithm, named *RWFlood*, is to remove the depressions by simulating the raising of an imaginary ocean that surrounds the terrain. In this process, the terrain is supposed to be an island surrounded by water that is iteratively raised. When the water level increases, it gradually floods the terrain cells and when it reaches a depression, it is filled by 'water'. That is, in the beginning, the water level is set to the elevation of the lowest cell in the terrain boundary, which means that these lowest cells are flooded. Then all cells adjacent to these flooded cells are stored for future processing. But, those cells that are lower than the current water level are raised to the current level (see Figure 1).

While similar to Yong-he *et al.* (2009), *RWFlood* is much improved. Since the terrain elevations can be stored as 16-bit integers (Farr *et al.* 2007; SRTM 2011), it is possible to raise the water level in discrete increments. That is, the water level is initialized to the lowest elevation in the terrain boundary and, at each step, it is incremented by 1 until it reaches the highest terrain elevation. In this process, *RWFlood* uses an array *Q* of queues for the cells that need to be stored for later processing such that *Q* contains one queue for each elevation – queue *Q*[*m*] stores the cells (to be processed) with elevation *m*. Initially, each cell in the terrain boundary is inserted into the corresponding queue. Supposing the lowest cells have elevation *k*, the process starts at queue *Q*[*k*] and, for each elevation *z* (water level) such that *Q*[*z*] is not empty, a cell is removed (conceptually, it is flooded) and its neighbors are visited. That is, given a neighboring cell *v*, if *v* has already been visited, it is done; on the other hand, if *v* has not been visited yet, and if its elevation is not lower than *z*, it is inserted in its corresponding queue; otherwise, if its elevation is lower than *z*, the elevation is set to *z* and it is inserted into *Q*[*z*]. Notice that the latter case corresponds to flood a depression point.

Figure 1 illustrates this process: in Figure 1(a) the water level is 70 m (no depression was flooded yet). Figure 1(b) shows the water level after some iterations (10 in the case); notice the depressions in the center of the terrain are below the water level but they are not flooded yet because they are not neighbors to the water. In Figure 1(c), the water level is 99 m and the cells in queue *Q*[*99*] are processed (only cells neighbors to the water were inserted in the queues). In this process, the 100 m cell elevation, in the rightmost peak, is inserted into the queue *Q*[*100*] and, when the water level is set to 100 m, the cells in *Q*[*100*] are processed (Figure 1(d)). Thus, the depression is now neighbor to the water and the cells' elevation in the depression are set to 100 m. Figure 1(e) shows the water level at 105 m.

The algorithm in Figure 2 shows the pseudocode for *RWFlood*. First, it creates an array Q of queues with indices ranging from the minimum elevation in the terrain boundary (*minElev*) to the maximum elevation in the terrain (*maxElev*). Then, cells in the terrain boundary are inserted into their corresponding queue and their directions are set to outside the terrain. After inserting the terrain border cells into the queues, the ocean level *z* is initialized with *minElev* and raised step by step to *maxElev*. Given a water level *z*, the cells in the queue *Q*[*z*] are processed ('flooded'). Notice that, when a cell *c* is processed, all cells adjacent to *c* that are inserted in a queue have their flow direction set to *c*. That is, the water in the adjacent cells flows to the cell *c* (conceptually, the flow direction is set to the opposite direction as the water gets into the cells). The direction of a cell is also used to check if that cell was not visited yet, that is, a flow direction null means the cell was not visited.

The cells in a flat area will be processed using a similar approach as in other methods such as *Terraflow* (Arge *et al.* 2003) and *r. watershed* (Metz *et al.* 2011), that is, the flow in a flat area will be directed to the spill point(s) (the cell(s) in the flat area boundary having a lower neighbor cell). In Figure 3 a portion of terrain is shown that includes a flat

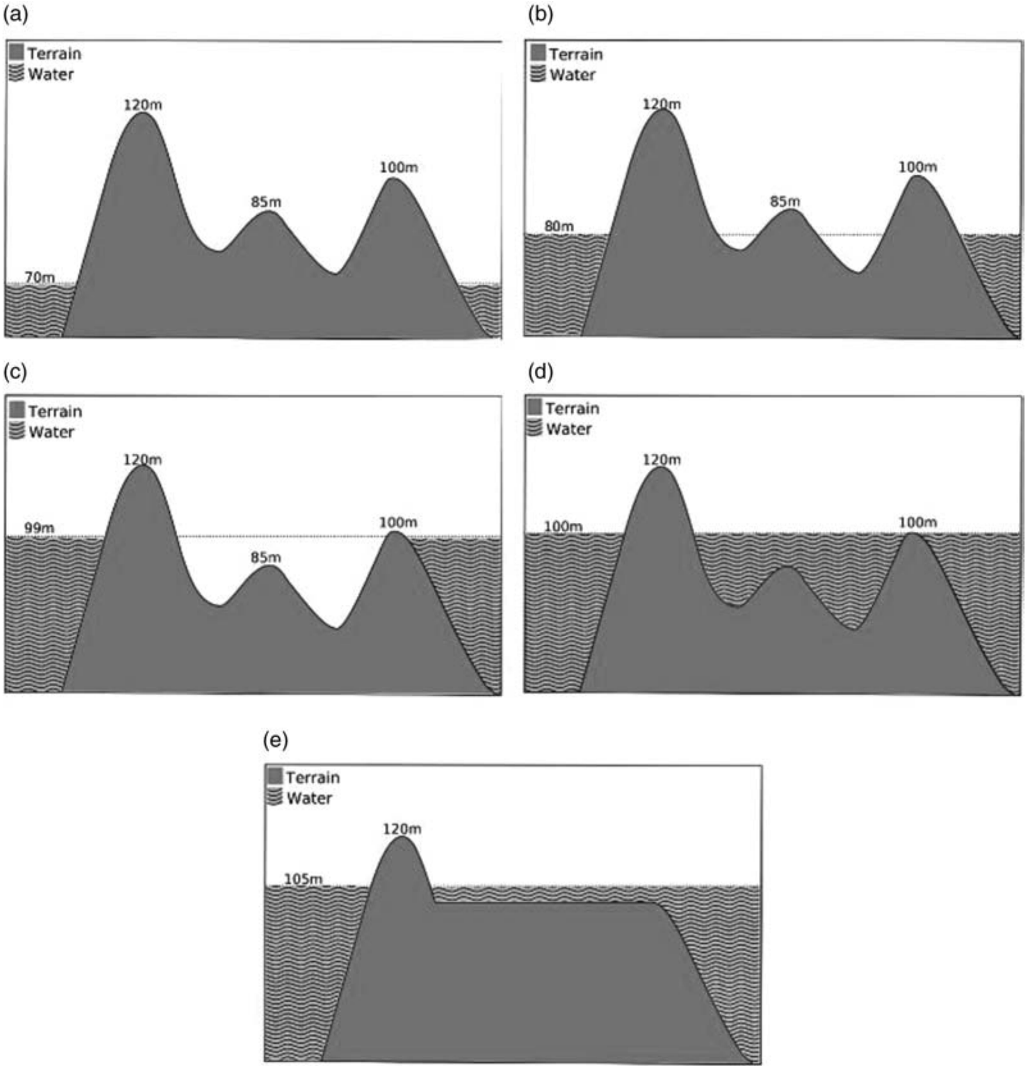

Figure 1 | The flooding process in five water levels: (a) 70 m, (b) 80 m, (c) 99 m, (d) 100 m, (e) 105 m.

Q1

area (light gray cells), surrounded by higher cells (dark gray), having two spill points (s_1 and s_2). After processing the spill points s_1 and s_2 (Figure 3(a)), the flow direction of the neighbor cells (f_1, f_2, f_3, and f_4) is set to these spill points (Figure 3(b)) and these cells are included in the queue corresponding to their elevation. Since the cells in the flat area have the same elevation, they will be in the same queue and, therefore, this processing will be done in a DFS order. Thus, the cells in a flat area will have directions set to the shortest path to the spill points (Figure 3(c)).

The method described above supposes the terrain elevations are represented using integer values as is usual in SRTM data. However, it is possible to adapt the code to process the terrain if the elevation data are stored using another format. For example, if the elevation is stored using real values with a precision of 10 centimeters and considering that the terrain elevations range from −424 to 8,850 m (the smallest and highest elevations on the Earth), it is possible to convert the data multiplying it by 10 and dropping its fractional component off. So, the data would require an array with $(8{,}850 - (-424)) \times 10 = 92{,}740$ queues to be processed. Notice that, even if this array of queues is sparse, the space used to store the empty queues is usually very small compared to the space required to

```
Let Q[minElev...maxElev] be an array of queues
for all cell $c$ in the terrain do
  $c.dir \leftarrow NULL$
end for
for all cell $c$ in the terrain border do
  $Q[c.elev].insert(c)$
  $c.dir \leftarrow OutsideTerrain$
end for
for $z = minElev \rightarrow maxElev$ do
  while $Q[z]$ is not empty do
    $c \leftarrow Q[z].remove()$
    for all cell $d$ neighbor of $c$ where $d.dir = NULL$ do
      $d.dir \leftarrow c$
      if $d.elev < z$ then
        $d.elev \leftarrow z$
      end if
      $Q[d.elev].insert(d)$
    end for
  end while
end for
```

Figure 2 | *RWFlood* algorithm: Fill depressions and compute flow directions.

store other data structures such as the terrain digital elevation matrix.

It is important to mention that while *RWFlood* automatically determines the flow direction of each cell during the flooding process, the Yong-he *et al.* (2009) method uses this similar flooding strategy only to remove the terrain depressions (it does not compute the flow direction). Thus, it is only a preprocessing step to remove the depressions that can be used by other flow direction methods. Additionally, another important improvement of *RWFlood* when compared to Yong-he *et al.*'s (2009) method is the time required to manage the cells during the flooding process, that is, storing the cells that need to be processed later and obtaining the next cell to be processed. In *RWFlood*, thanks to the use of the array of queues, both inserting a cell and obtaining the next cell take a constant time while in Yong-he *et al.*s (2009) method, where a priority queue is used, both operations take *O(log m)* time where *m* is the queue size (that, in the worst case, can have the same order of magnitude as the terrain size).

After computing the flow direction, *RWFlood* uses an algorithm based on graph topological sorting to compute the accumulated flow (see Figure 4). Conceptually, the idea is to process the flow network as a graph where each terrain cell is a vertex and there is a directed edge connecting a cell c_1 to a cell c_2 if and only if c_1 flows to c_2. Initially, all vertices in the graph have 1 unit of flow. Then, in each step, a cell *c* with in-degree 0 is set as visited and its flow is added to *next(c)*'s flow where *next(c)* is the cell following *c* in the graph. After processing *c*, the edge connecting *c* to *next(c)* is removed (i.e. *next(c)*'s in-degree is decremented) and if the in-degree of *next(c)* becomes 0, *next(c)* is also similarly processed.

As one can see, the proposed algorithm is very simple and its complexity is linear in the terrain size. Since, in the first step (flow direction computation) each terrain cell is inserted and removed from a queue exactly only one time and both are constant time operations. The second step (computing the flow accumulation) is based on the topological sorting which is linear time too.

EXPERIMENTAL ANALYSIS

The proposed algorithm *RWFlood* was experimentally evaluated comparing its execution time against other widely used methods such as ArcGIS version 9.0 and the methods *Terraflow* (Arge *et al.* 2003) and *Watershed* (Metz *et al.* 2011) included in GRASS GIS 6.4 as the modules *r. terraflow* and *r. watershed*.

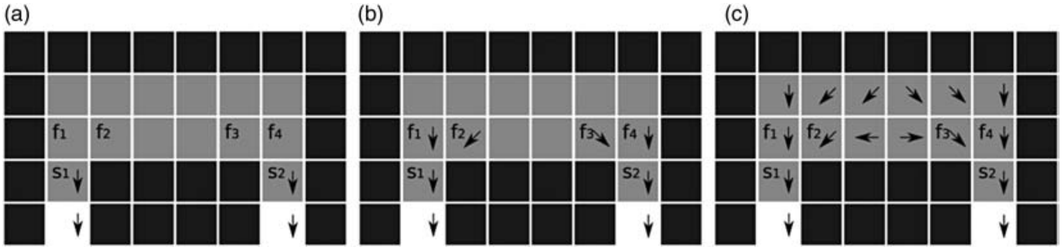

Figure 3 | Flat area processing: (a) a flat area (light gray cells) surrounded by higher cells (dark gray) having two spill points s_1 and s_2; (b) after processing the spill points s_1 and s_2, the flow direction of the neighbor cells f_1, f_2, f_3, and f_4 is set and, after some iterations, the flow direction of the cells in the flat area is set (c).

```
Let $next(c)$ be the cell following $c$ in the flow route
For all $c$, $Degree[c] \leftarrow 0$ and $Flow[c] \leftarrow 1$
for all cell $c$ in the terrain do
    $Degree[next(c)] \leftarrow Degree[next(c)] + 1$
end for
for all Non-visited terrain cell $c$ do
    $d \leftarrow c$
    while $Degree[d] = 0$ do
        $d.visited \leftarrow true$
        if next(d) is outside the terrain then
            $Break - while$
        end if
        $Flow[next(d)] \leftarrow Flow[next(d)] + Flow(d)$
        $Degree[next(d)] \leftarrow Degree[next(d)] - 1$
        $d \leftarrow next(d)$
    end while
end for
```

Figure 4 | Algorithm to compute the flow accumulation.

The *r. watershed* module is an efficient method for computing flow direction and flow accumulation in terrains stored in internal memory. While the older versions of *r. watershed* were very slow for processing of large terrains (Arge *et al.* 2003; Danner *et al.* 2007), the version evaluated in this paper implements a fast flow computation algorithm proposed by Metz (Metz *et al.* 2011) which is, as far as we know, the fastest flow computation method designed for internal memory processing.

However, for huge terrains, the *r. watershed* module may need more memory than that available internally. Thus, the method needs to do external memory processing and so, *r. terraflow* module may be more efficient than *r. watershed* since *Terraflow* is an I/O efficient (Arge *et al.* 2003) algorithm designed to process huge terrains. Therefore, in the tests, both methods included in GRASS (*r. watershed* and *r. terraflow*) were executed.

As well as the processing time, the coherence of the flow network obtained by *RWFlood* algorithm was also evaluated comparing it against the networks computed using GRASS.

PERFORMANCE TESTS

The algorithm *RWFlood* was implemented in C++, compiled using g++ 4.5.2, and several tests were done to evaluate its execution time. All tests were executed in a Core 2 Duo machine with 2.8 GHz and 4 GB of memory. *RWFlood*, *r. watershed*, and *r. terraflow* were executed in the Ubuntu Linux 11.04 64bit Operating System, and ArcGIS in the Windows XP 32bit Operating System.

We generated terrains with different dimensions using SRTM data representing four different regions. Table 1 and charts in Figures 5 and 6 show the four methods' processing time on those terrains. Notice that, in all tests, the *RWFlood* was much faster than all the other three methods and, in many cases, it was more than 100 times faster.

As expected, the internal memory processing (when possible) is more efficient than the external processing. In particular, considering the results presented in Table 1, all the internal memory methods were faster than *r. terraflow*

Table 1 | Processing time for different regions and terrain sizes

		Processing time (in seconds)			
Terrain dataset	**Size (No. cells)**	***RWFlood***	***r. watershed***	***r. terraflow***	***ArcGIS***
1	$5,000^2$	5	47	405	293
	$10,000^2$	14	233	2,075	3,860
	$20,000^2$	68	8,776	9,924	17,509
2	$5,000^2$	3	48	401	376
	$10,000^2$	16	242	2,059	2,869
	$20,000^2$	73	9,063	10,015	13,707
3	$5,000^2$	5	44	411	219
	$10,000^2$	27	231	2,106	1,586
	$20,000^2$	125	9,185	10,140	7,693
	$30,000^2$	1062	74,135	24,746	26,338
4	$5,000^2$	5	46	389	264
	$10,000^2$	27	246	2,038	1,449
	$20,000^2$	145	9,374	9,804	8,546
	$30,000^2$	912	81,195	24,013	33,829

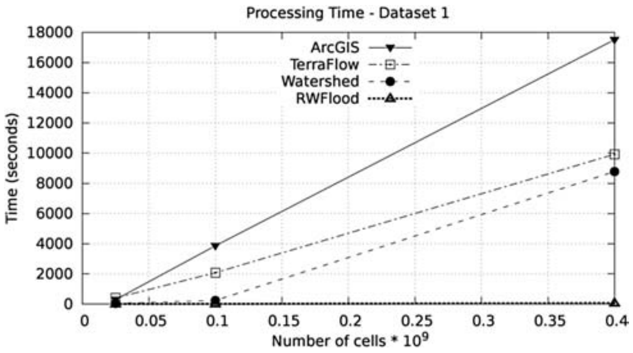

Figure 5 | Processing time of *RWFlood, Watershed, Terraflow*, and *ArcGIS* executed in dataset 1.

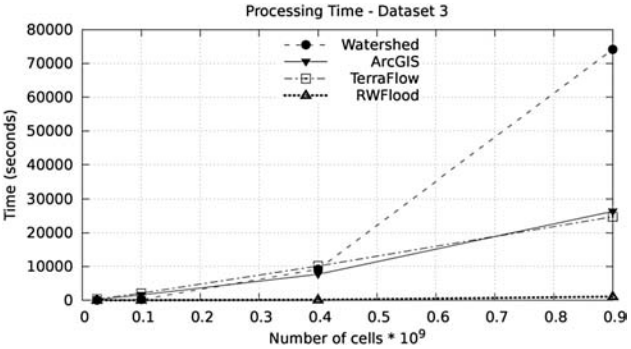

Figure 6 | Processing time of *RWFlood, Watershed, Terraflow*, and *ArcGIS* executed in dataset 3.

Table 2 | *RWFlood* and *r. terraflow* processing time considering larger terrain portions

		Processing time (sec.)	
Terrain	**Size (No. cells)**	***RWFlood***	***r. terraflow***
Dataset 4	$5{,}000^2$	5	389
	$10{,}000^2$	27	2,038
	$15{,}000^2$	72	5,044
	$20{,}000^2$	145	9,804
	$25{,}000^2$	288	15,838
	$30{,}000^2$	912	24,013
	$35{,}000^2$	2,798	32,123
	$40{,}000^2$	8,872	44,965
	$45{,}000^2$	18,515	51,290
	$50{,}000^2$	103,572	66,703

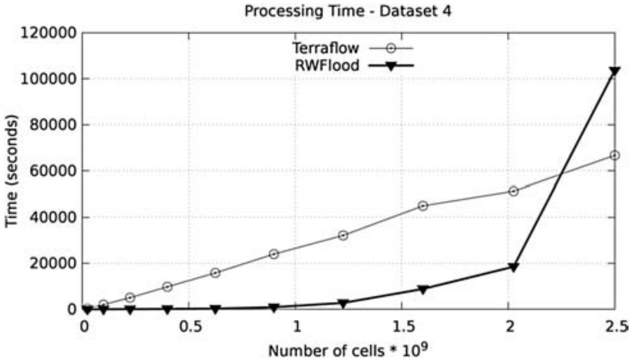

Figure 7 | Processing time graph of *RWFlood* and *Terraflow* considering larger terrain portions.

for terrains having $20{,}000^2$ cells or less. And, *r. terraflow* became more efficient than *r. watershed* and ArcGIS for terrains with about $20{,}000^2$ and $25{,}000^2$ cells respectively. However, even for terrains with $30{,}000^2$ cells, *r. terraflow* was slower than *RWFlood*.

Although the *RWFlood* complexity is linear in the terrain size (as described in the section, The proposed algorithm) the execution time presented in Table 1 seems to grow more than linearly with the terrain size, but this nonlinear behavior can be explained mainly because of the random access to the terrain matrix since the access time to huge matrices cells depends on the access pattern (sequential or not) and the memory hierarchy, in particular, the cache memory size.

Of course, there comes a point when the terrain cannot be processed in the internal memory by *RWFlood* and so, the external memory methods such as *r. terraflow* will be more efficient than it. Thus, to compare the performance of *RWFlood* and *r. terraflow* in very huge terrains, more tests were executed considering portions of the terrain in dataset 4 with larger sizes (see Table 2 and Figure 7).

Notice that *RWFlood* was faster than *r. terraflow* for terrains with 2×10^9 (about $45{,}000^2$) cells or less and its execution time became higher only for terrains having about $50{,}000^2$ cells or more when the terrains need to be processed using the external memory. This threshold is much larger than the terrain size for which *r. watershed* and *ArcGIS* became slower than *r. terraflow*. It happens because *RWFlood* was very carefully implemented to save memory and, thus, it can process huge terrains in internal memory. It does not use a priority queue, as do many other methods, to organize the terrain cells when removing the depressions. Instead, it uses an array of queues, one for each elevation, and so the cells can be processed in constant time. Also, the flow direction is determined simultaneously to the depression removal – many other methods can only compute the flow direction after removing all depressions. And, the idea of raising water and flooding the cells makes the depression filling very fast and simple. Finally, instead of creating a structure to store all the non-visited cells during the flooding step, the cell's flow direction attribute is used as a flag to indicate if a cell was visited or not.

In conclusion, *RWFlood* was much faster than *r. watershed* (the current fastest internal memory method) and, also, *RWFlood* was able to process terrains much bigger than could *r. watershed*. Thus, besides being faster, *RWFlood* can postpone the point where methods designed for external memory processing are better than internal memory methods. For example, as the tests have shown, using 4 GB of memory, *RWFlood* is more efficient than *r. terraflow* for terrains with up to 10^9 cells and, as one may expect, this terrain size could be bigger using more internal memory.

COMPARING THE FLOW NETWORKS

The accuracy of the flow network obtained by *RWFlood* algorithm was also evaluated. Figure 8 shows the networks obtained by *RWFlood* and GRASS (*r. watershed*) in datasets 1 and 3. Notice that, the two networks from dataset 1 are very similar and the networks from dataset 3 are also similar but have small differences mainly in flat areas (as indicated by the rectangles), which can be explained because the methods use different strategies to process flat areas.

CONCLUSIONS

We presented a simple and very fast internal memory algorithm for computing the flow network (that is, flow accumulation and flow direction) on terrains represented by an elevation matrix. The algorithm is linear in the terrain size and its processing time was compared against some other classic and recent methods included in GRASS

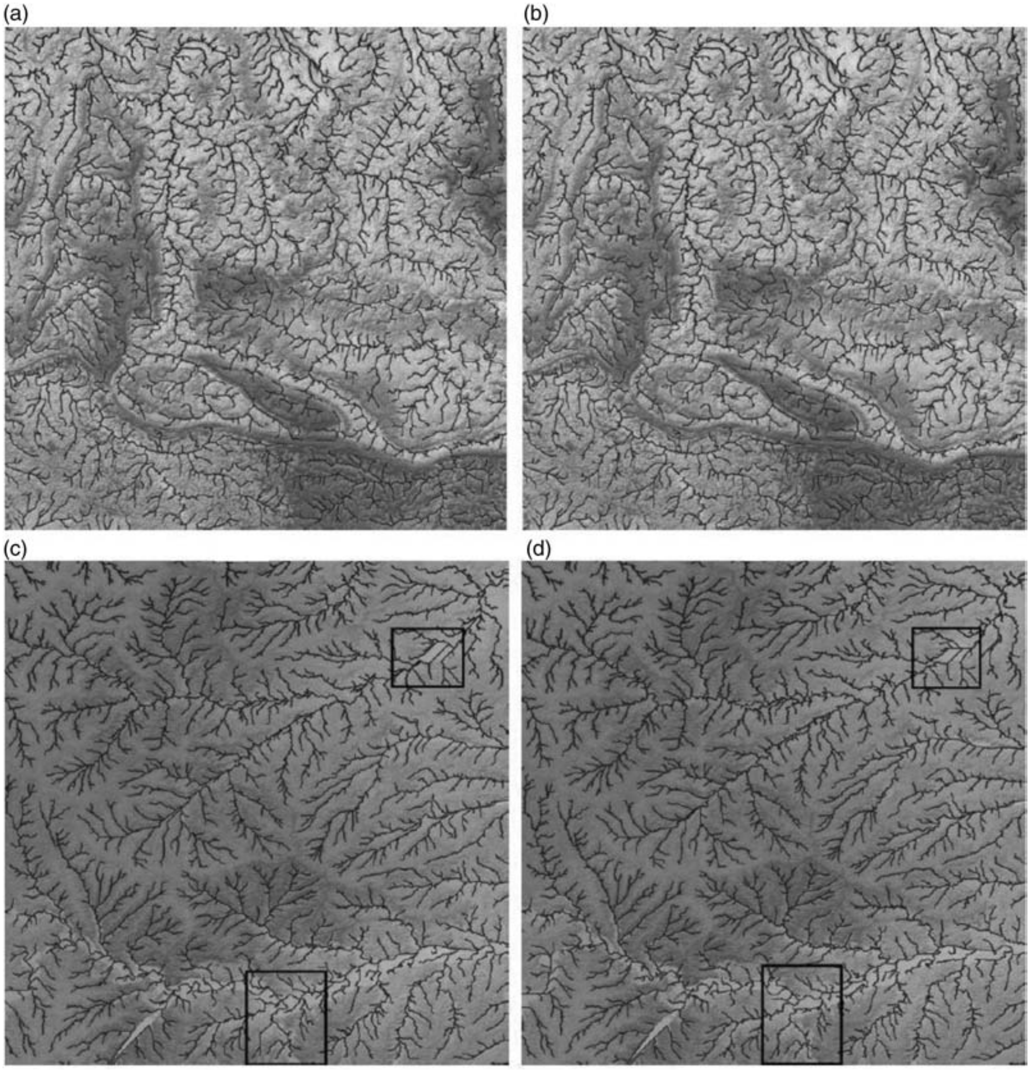

Figure 8 | Networks extracted by *RWFlood* ((a) and (c)) and by *GRASS* ((b) and (d)) in datasets 1 (first row) and 3 (second row).

(*r. watershed* and *r. terraflow*) and *ArcGIS*. As tests have shown, the proposed method was much faster (in some cases, more than 100 times) than the other methods. Also, it was able to process efficiently, in the internal memory, terrains larger than other internal memory methods did.

A next step is to adapt the flooding process based on raising the water level to compute other hydrological features, such as the ridge lines and watershed. The *RWFlood* source code can be downloaded from www.dpi.ufv.br/~marcus/RWFlood.

ACKNOWLEDGEMENTS

This research was partially supported by FAPEMIG – The Minas Gerais State Research Foundation, CAPES, CNPq – National Council for Scientific and Technological Development and NSF grants CMMI-0835762 and IIS-1117277.

REFERENCES

Arge, L., Chase, J. S., Halpin, P., Toma, L., Vitter, J. S., Urban, D. & Wickremesinghe, R. 2003 Flow computation on massive grid terrains. *Geoinformatica* **7** (4), 283–313.

Danner, A., Agarwal, P. K., Yi, K. & Arge, L. 2007 Terrastream: from elevation data to watershed hierarchies. In *Proc. ACM Sympos. on Advances in Geographic Information Systems*, pp. 212–219.

Ehlschlaeger, C. 1989 Using the A* search algorithm to develop hydrologic models from digital elevation data. In: *International Geographic Information Systems (IGIS) Symposium*. 18–19 March, Baltimore, MD, pp. 275–281.

Farr, T. G., Rosen, P. A., Caro, E., Crippen, R., Duren, R., Hensley, S., Kobrick, M., Paller, M., Rodriguez, E., Roth, L., Seal, D., Shaer, S., Shimada, J., Umland, J., Werner, M., Oskin, M., Burbank, D. & Alsdorf, D. 2007 The shuttle radar topography mission. *Rev. Geophys.* **45**, RG2004 + .

GRASS 2011 *Geographic Resources Analysis Support System (GRASS GIS) Software*. Open Source Geospatial Foundation. Available at: http://grass.osgeo.org (accessed 11/05/2013).

Grimaldi, S., Nardi, F., Benedetto, F. D., Istanbulluoglu, E. & Bras, R. L. 2007 A physically-based method for removing pits in digital elevation models. *Adv. Water Resour.* **30**, 2151–2158.

Jakowatz Jr, C. V., Wahl, D. E., Eichel, P. H., Ghiglia, D. C. & Thompson, P. A. 1996 *Spotlight-Mode Synthetic Aperature Radar: A Signal Processing Approach*. Kluwer Academic Publishers, Boston, MA.

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 System Sci.* **15**, 667–678.

O'Callaghan, J. F. & Mark, D. M. 1984 The extraction of drainage networks from digital elevation data. *Comput. Vis. Graph.* **28**, 323–344.

Planchon, O. & Darboux, F. 2002 A fast, simple and versatile algorithm to fill the depressions of digital elevation models. *Catena* **46**, 159–176.

Santini, M., Grimaldi, S., Nardi, F., Petroselli, A. & Rulli, M. C. 2009 Pre-processing algorithms and landslide modelling on remotely sensed dems. *Geomorphology* **113**, 110–125.

SRTM 2011 *SRTM Topography Documentation*. Available at: http://dds.cr.usgs.gov/srtm/version21/Documentation/ (accessed 11/05/2013).

Tarboton, D. G. 1997 A new method for the determination of flow directions and upslope areas in grid digital elevation models. *Water Resour. Res.* **33**, 309–319.

Wang, L. & Liu, H. 2006 An efficient method for identifying and filling surface depressions in digital elevation models for hydrologic analysis and modeling. *Int. J. Geogr. Inform. Sci.* **20**, 193–213.

Yong-he, L., Wan-Chang, Z. & Jing-Wen, X. 2009 Another fast and simple dem depression-filling algorithm based on priority queue structure. *Atmos. Ocean. Sci. Lett.* **2**, 117–120.

Zhu, Q., Tian, Y. & Zhao, J. 2006 An efficient depression processing algorithm for hydrologic analysis. *Comput. Geosci.* **32**, 615–623.

First received 27 May 2013; accepted in revised form 15 October 2013. Available online 14 November 2013

Author Queries

Journal: Journal of Hydroinformatics

Manuscript: HYDRO-D-13-00068

Q1 In supplied Figure 1 is not sufficient print quality. Please resupply as a high resolution file (300 dpi or above) with sharp lines and text.