A new method for computing the drainage network based on raising the level of an ocean surrounding the terrain

Salles V. G. Magalhães[1], Marcus V. A. Andrade[1], W. Randolph Franklin[2], and Guilherme C. Pena[1]

[1]Department of Informatics (DPI) - Univ. Fed. Viçosa, Viçosa, Brazil
{sallesviana, marcus.ufv, guipena}@gmail.com
[2]ECSE - Rensselaer Polytechnic Institute, Troy, NY, USA
wrf@ecse.rpi.edu

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 matrix. 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.

1 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 a matrix. 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 Jr. et al. 1996).

As described by Metz (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 and Mark, 1984; Tarboton, 1997; Arge et al., 2003; Danner et al., 2007; Zhu et al., 2006; Planchon and Darboux, 2002; Wang and Liu, 2006; Yong-he et al., 2009) remove depressions by increasing the cell's elevation to fill the sinks and then directing 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 (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.

In this paper, we present a new and faster terrain flow computation method. The proposed method surrounds the terrain by water (as an island). Then it raises the outside water level step by step, and fills 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.

2 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, the depression 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. First, 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.

Next, RWFlood uses an array Q of queues for the cells that need to be stored for later processing. 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 flooding a depression point.

The implementation 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 is stored using another format. For example, if the elevation is stored using real values with precision of 10 centimeters and considering that the terrain elevations range from -424m to 8850m (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 (8850 - (-424)) x 10 = 92740 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 store others data structures such as the terrain digital elevation matrix.

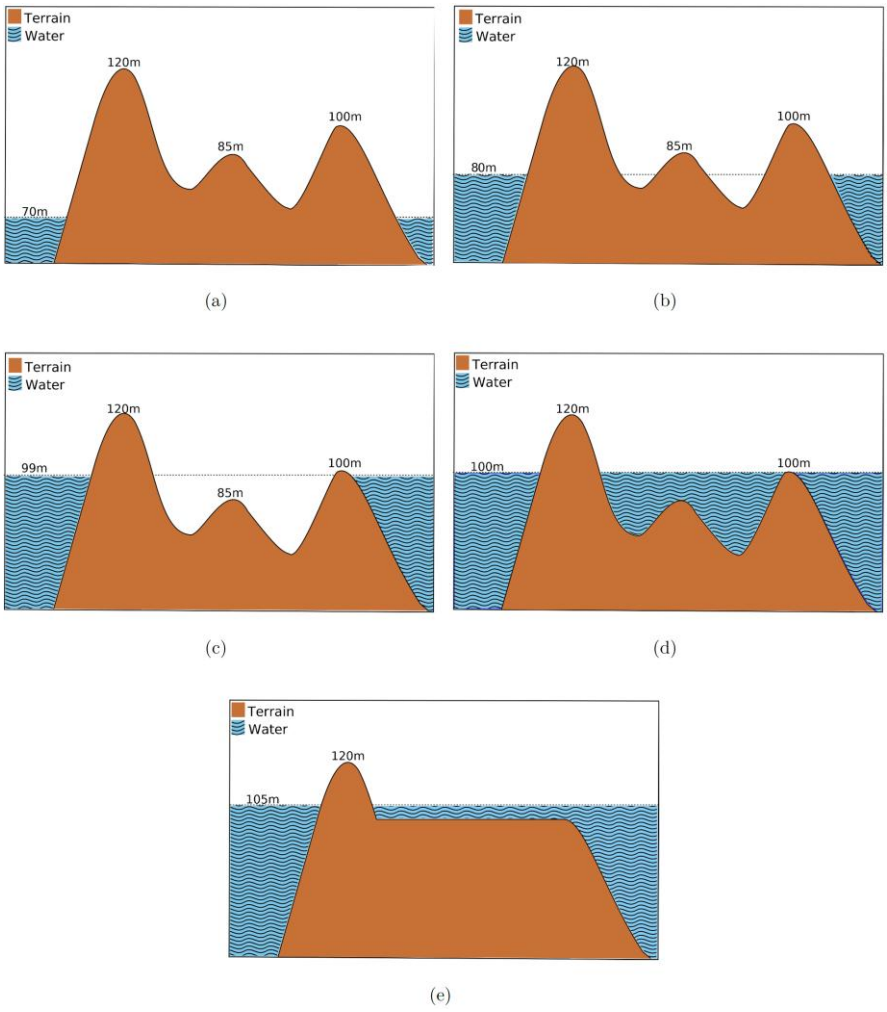

Fig. 1. The flooding process, shown at 5 different water levels: (a) 70m, (b) 80m, (c) 99m, (d) 100m, (e) 105m.

Furthermore, RWFlood automatically determines the flow direction of each cell during the flooding. When a cell *c* is processed, all the 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 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.

```
1: Let Q[minElev...maxElev] be an array of queues
2: for all cell c in the terrain do
3:    c.dir ← NULL
4: end for
5: for all cell c in the terrain border do
6:    Q[c.elev].insert(c)
7:    c.dir ← OutsideTerrain
8: end for
9: for z = minElev → maxElev do
10:   while Q[z] is not empty do
11:      c ← Queues[z].remove()
12:      for all cell n neighbor of c where n.dir = NULL do
13:         n.dir ← c
14:         if n.elev < z then
15:            n.elev ← z
16:         end if
17:         Q[n.elev].insert(n)
18:      end for
19:   end while
20: end for
```

Fig. 2. RWFlood algorithm – Fill depressions and compute flow directions

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 points in the queue *Q[z]* are processed ("flooded"). The direction is also used to check if a cell was not visited yet, that is, a flow direction null means the cell was not visited.

Figures 1(a), 1(b), 1(c), 1(d) and 1(e) illustrate the flooding process: in Figure 1(a) the water level is 70m (no depression was flooded yet). The 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 aren't neighbors to the water. In Figure 1(c), the water level is 99m and the cells in queue *Q[99]* are processed (only cells neighbors to the water were inserted in the queues).

In this process, the 100m cell elevation, in the rightmost peak, is inserted into the queue *Q[100]* and, when the water level is set to 100m, the cells in *Q[100]* are processed (Figure 1(d)). Thus, the depression is now neighbor to the water and cells' elevation in the depression are set to 100m. Figure 1(e) shows the water level at 105m.

After computing the flow direction, RWFlood uses an algorithm based on graph topological sorting to compute the accumulated flow. See algorithm in Figure 3. 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.

```
1: Let next(c) be the cell following c in the flow route
2: For all c, Degree[c] ← 0 and Flow[c] ← 1
3: for all cell c in the terrain do
4:    Degree[next(c)] ← Degree[next(c)] + 1
5: end for
6: for all Non-visited terrain cell c do
7:    d ← c
8:    while Degree[d] = 0 do
9:       d.visited ← true
10:      if next(d) is outside the terrain then
11:         Break − while
12:      end if
13:      Flow[next(d)] ← Flow[next(d)] + Flow(d)
14:      Degree[next(d)] ← Degree[next(d)] − 1
15:      d ← next(d)
16:   end while
17: end for
```

Fig. 3. Algorithm to compute the flow accumulation

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 (compu-

ting the flow accumulation) is based on the topological sorting which is linear time too.

3 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.

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 the 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.

3.1 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 2.8GHz and 4GB 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 some regions of the world - see Figures 4 and 5:

- Tres Marias: a region in the Brazil, containing the Tres Marias Dam.

- Tapajos: a region in Brazil containing the Tapajos River, an important tributary of the Amazon River.
- Region 2: SRTM USA region 2.
  Region 3: SRTM USA region 3.

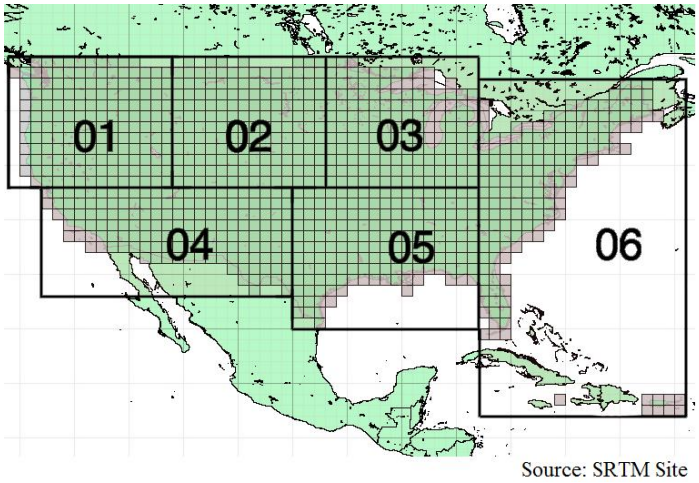

Fig. 4. USA SRTM regions.

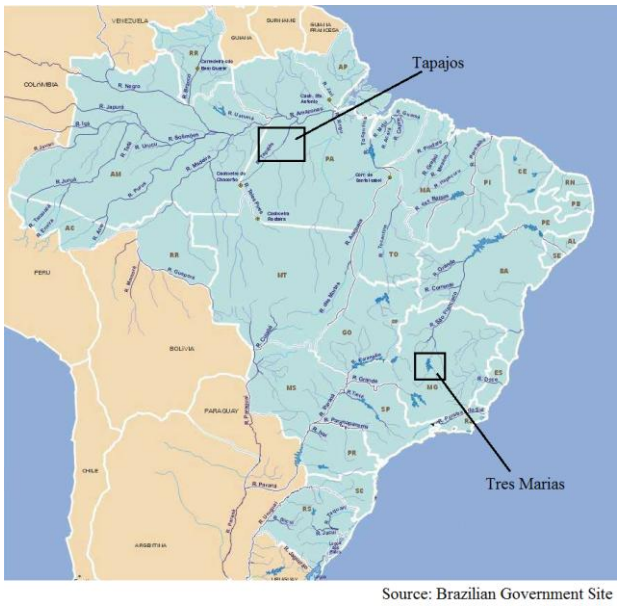

Fig. 5. Tapajos and Tres Marias Brazilian regions.

Table 3.1 and charts in Figures 6, 7, 8 and 9 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.

Table 3.1. Processing time for different regions and terrain sizes.

Terrain	Size	Processing time (in seconds)			
	[# cells]	RWFlood	r.watershed	r.terraflow	ArcGIS
Tres Marias	5000^2	5	47	405	293
	10000^2	14	233	2075	3860
	20000^2	68	8776	9924	17509
Tapajos	5000^2	3	48	401	376
	10000^2	16	242	2059	2869
	20000^2	73	9063	10015	13707
Region 3	5000^2	5	44	411	219
	10000^2	27	231	2106	1586
	20000^2	125	9185	10140	7693
	30000^2	1062	74135	24746	26338
Region 2	5000^2	5	46	389	264
	10000^2	27	246	2038	1449
	20000^2	145	9374	9804	8546
	30000^2	912	81195	24013	33829

As expected, the internal memory processing (when possible) is more efficient than the external processing. In particular, considering the results presented in the figures, mainly Figures 8 and 9, all the internal memory methods were faster than r.terraflow for terrains having 20000^2 cells or less. And, r.terraflow became more efficient than r.watershed and ArcGIS for terrains with about 20000^2 and 25000^2 cells respectively. However, even for terrains with 30000^2 cells, r.terraflow was slower than RWFlood.

Note that, although the RWFlood complexity is linear in the terrain size (as described in the section 2), the execution time presented in the Table 3.1 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.

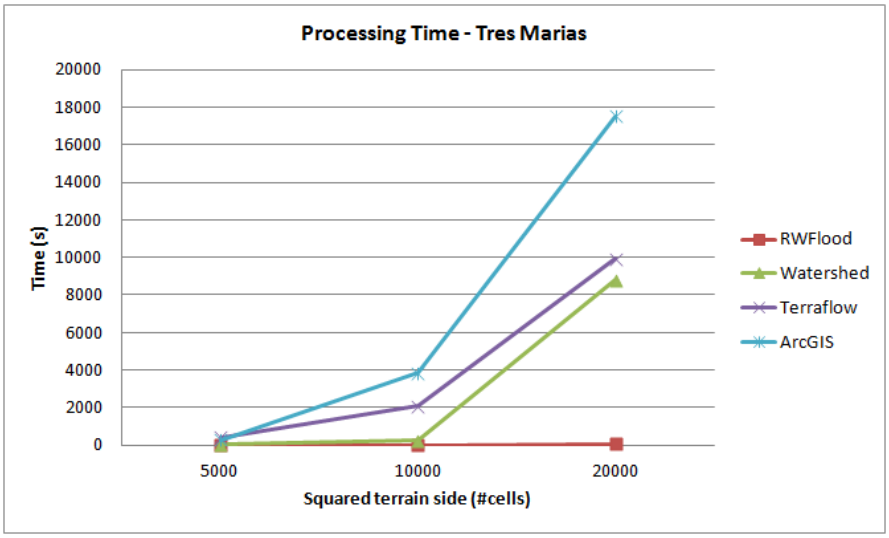

Fig. 6. Processing time graph of RWFlood, Watershed, Terraflow and ArcGIS executed in terrains from Tres Marias region.

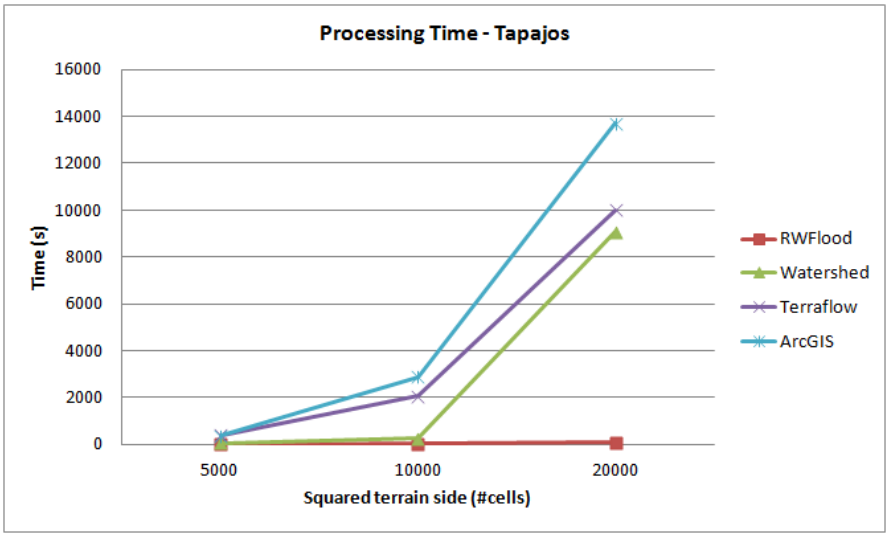

Fig. 7. Processing time graph of RWFlood, Watershed, Terraflow and ArcGIS executed in terrains from Tapajos region.

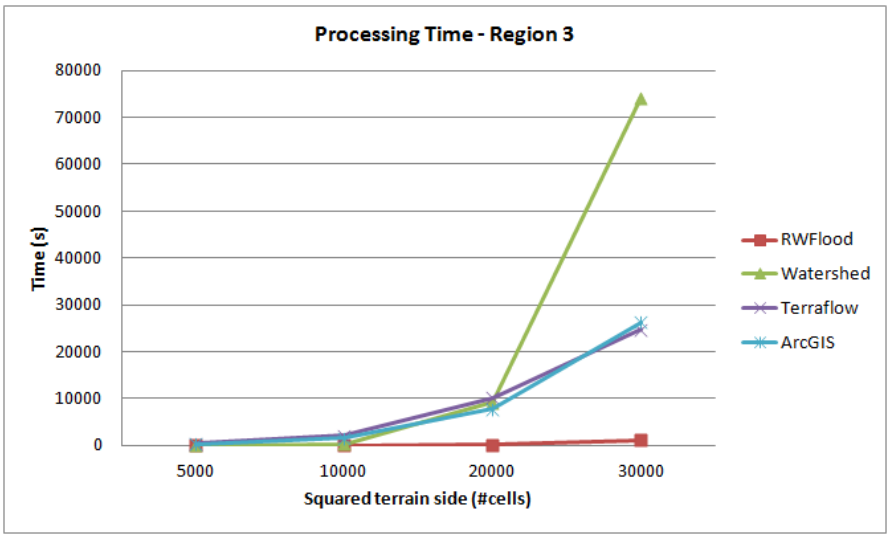

Fig. 8. Processing time graph of RWFlood, Watershed, Terraflow and ArcGIS executed in terrains from Region 3.

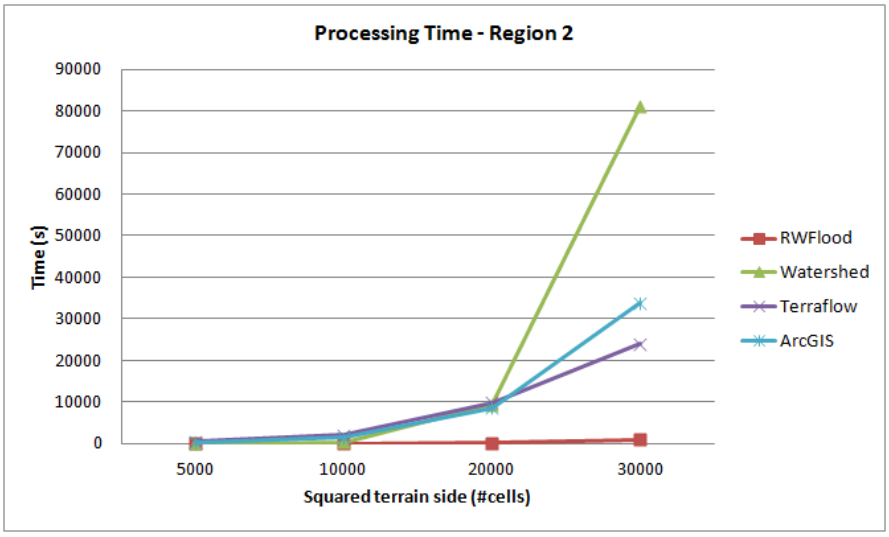

Fig. 9. Processing time graph of RWFlood, Watershed, Terraflow and ArcGIS executed in terrains from Region 2.

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 pieces of the terrain in Region 2 with larger sizes. See Table 3.2 and Figure 10.

Table 3.2. RWFlood and r.terraflow processing time considering larger terrain pieces from Region 2

Terrain	Size [# cells]	Processing Time (sec.) RWFlood	r.terraflow
Region 2	5000^2	5	389
	10000^2	27	2038
	15000^2	72	5044
	20000^2	145	9804
	25000^2	288	15838
	30000^2	912	24013
	35000^2	2798	32123
	40000^2	8872	44965
	45000^2	18515	51290
	50000^2	103572	66703

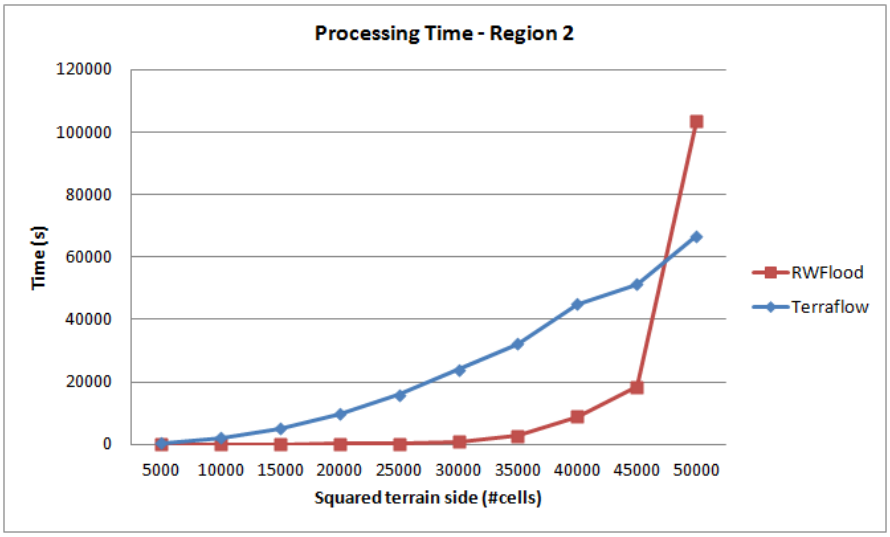

Fig. 10. Processing time graph of RWFlood and Terraflow considering larger terrain pieces from Region 2.

Notice that RWFlood was faster than r.terraflow for terrains with $2x10^9$ (about 45000^2) cells or less and its execution time became higher only for terrains having about 50000^2 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 doesn't use a priority queue, as 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.

Concluding, RWFlood was much faster than r.watershed (the current fastest internal memory method) and, also, RWFlood was able to process terrains much bigger than r.watershed did. 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 4GB of memory, RWFlood is more efficient than r.terraflow for terrains with up to 10^9 cells and, as one can expect, this terrain size could be bigger using more internal memory.

3.2 Comparing the flow networks

The accuracy of the flow network obtained by RWFlood algorithm was also evaluated. Figures 11, 12, 13 and 14 show the networks obtained by the methods RWFlood and GRASS (r.watershed) in the Tapajos and Region 3 terrains. Notice that, the two networks, in each terrain, are very similar with small differences mainly in flat areas what can be explained because the methods use different strategies to process flat areas.

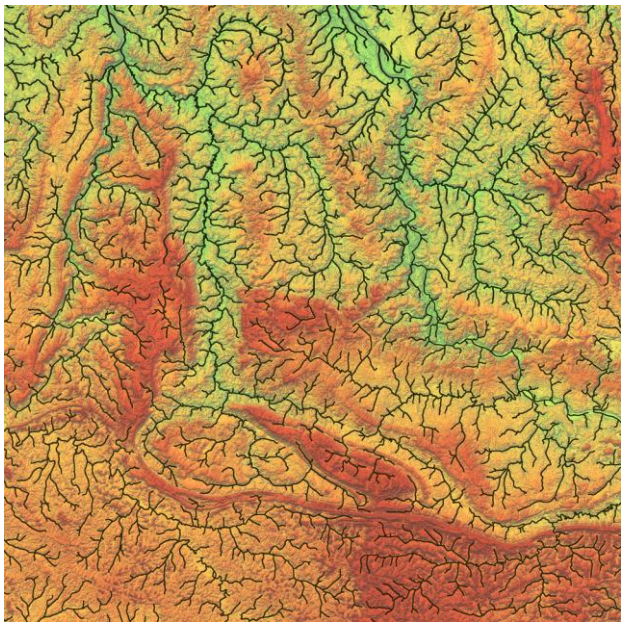

Fig. 11. Network extracted by RWFlood in a terrain from Tapajos region.

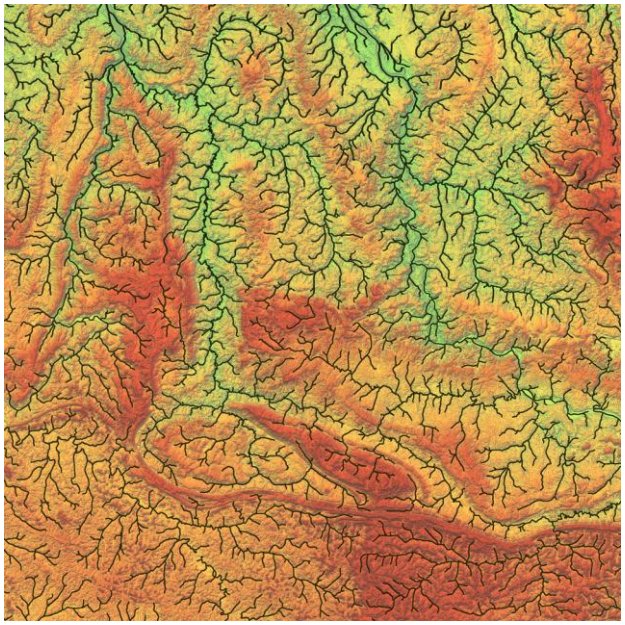

Fig. 12. Network extracted by GRASS in a terrain from Tapajos region.

Fig. 13. Network extracted by RWFlood in a terrain from Region 3.

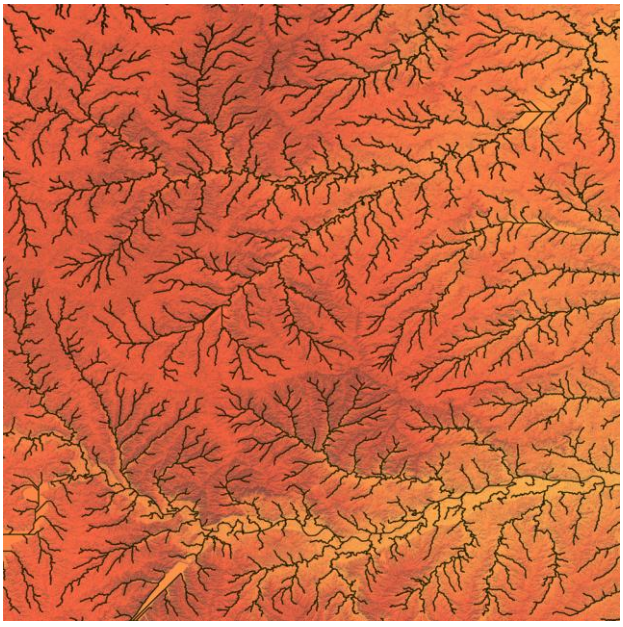

Fig. 14. Network extracted by GRASS in a terrain from Region 3.

4 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 (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.

Acknowledments

This research was partially supported by FAPEMIG - The Minas Gerais State Research Foundation, CAPES, CNPq - National Council for Scientific and Technological Development, Sydle, 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.

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. (Baltimore, MD, 18-19 March 1989), 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, Reviews of Geophysics 45 RG2004+.

GRASS (2011) Geographic Resources Analysis Support System (GRASS GIS) Software. Open Source Geospatial Foundation, http://grass.osgeo.org (accessed 11/10/2011).

Grimaldi, S., Nardi, F., Benedetto, F.D., Istanbulluoglu, E., Bras, R.L. (2007) A physically-based method for removing pits in digital elevation models, Advances in Water Resources 30, pp. 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, Massachusetts.

Metz, M., Mitasova, H., Harmon, R.S. (2011) Efficient extraction of drainage networks from massive, radar-based elevation models with least cost path search, Hydrology and Earth System Sciences 15, pp. 667-678.

O'Callaghan, J.F., Mark, D.M. (1984) The extraction of drainage networks from digital elevation data, Computer Vision, Graphics, and Image Processing 28, pp. 323-344.

Planchon, O., Darboux, F. (2002) A fast, simple and versatile algorithm to fill the depressions of digital elevation models, Catena 46, pp. 159-176.

Santini, M., Grimaldi, S., Nardi, F., Petroselli, A., Rulli, M.C. (2009) Preprocessing algorithms and landslide modelling on remotely sensed dems, Geomorphology 113, pp. 110-125.

SRTM (2011) SRTM Topography Documentation, http://dds.cr.usgs.gov/srtm/version2 1/Documentation/ (accessed 11/10/2011).

Tarboton, D.G. (1997) A new method for the determination of flow directions and upslope areas in grid digital elevation models, Water Resources Research 33, pp. 309-319.

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, pp. 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, Atmospheric and Oceanic Science Letters 2.

Zhu, Q., Tian, Y., Zhao, J. (2006) An efficient depression processing algorithm for hydrologic analysis, Comput. Geosci. 32, pp. 615-623.