

International Journal of Computational Geometry & Applications
© World Scientific Publishing Company

FLOODING COUNTRIES AND DESTROYING DAMS*

RODRIGO I. SILVEIRA and RENÉ VAN OOSTRUM

*Department of Information and Computing Sciences, Utrecht University
3508 TB Utrecht, the Netherlands
{rodrigo, rene}@cs.uu.nl*

Received 15 April 2008

Revised 05 December 2008

Communicated by Tetsuo Asano

ABSTRACT

In many applications of terrain analysis, pits or local minima are considered artifacts that must be removed before the terrain can be used. Most of the existing methods for local minima removal work only for raster terrains. In this paper we consider algorithms to remove local minima from polyhedral terrains, by modifying the heights of the vertices. To limit the changes introduced to the terrain, we also try to minimize the total displacement of the vertices. Two approaches to remove local minima are analyzed: lifting vertices and lowering vertices. For the former we show that all local minima in a terrain with n vertices can be removed in the optimal way in $\mathcal{O}(n \log n)$ time. For the latter we prove that the problem is NP-hard, and present an approximation algorithm with factor $2 \ln k$, where k is the number of local minima in the terrain.

Keywords: Polyhedral terrains; realistic terrains; local minima removal.

1. Introduction

Digital terrain analysis is an important area of GIS. In many cases, when the terrains are used for purposes concerning land erosion, landscape evolution or hydrology, it is generally accepted that the majority of the depressions present in the terrains are likely to be spurious features. The sources of such artifacts can be many, including low-quality input data, interpolation errors during the generation of the terrain model and truncation of interpolation values¹⁴. As a result, it is standard in many applications of terrain analysis, particularly in hydrologic applications such as automatic drainage analysis, to do some kind of preprocessing of the terrain to remove these spurious sinks^{22, 26}. This is because this kind of artifact can severely hinder flow routing. Several related terms have been used before to refer to these features, such as depressions, sinks, pits and local minima. In this paper, following the computational geometry literature, we use the term *local minimum*.

*This research has been partially funded by the Netherlands Organisation for Scientific Research (NWO) under the project GOGO. A preliminary version of this paper appeared in WADS 2007.

The most widely used type of *digital terrain model*, or simply *terrain*, is the square grid digital elevation model (raster DEM), mainly due to its simplicity. Another common type of terrain is the triangulated irregular network (TIN), which is a triangulation of a set of points with elevation. It involves a more complex data structure because it is necessary to store its irregular topology, but also has several advantages, such as variable density and continuity.

Regarding the removal of local minima, most of the literature in GIS has focused on algorithms for (raster) DEMs. Most of the proposed methods are some type of “pit filling” technique^{3,14,26}. They consist in raising the local minimum to the elevation of its lowest neighbor. This type of method implicitly assumes that most of the spurious local minima result only from underestimation errors, neglecting the ones caused by overestimation. Not many papers address the problem in the opposite way, removing local minima by lowering a neighboring vertex to a lower height. An example of such a technique is the one proposed by Rieger¹⁸ and is also part of the “outlet breaching” algorithm of Martz and Garbrecht¹⁵. Even though pit filling is the most widely implemented method for local minima removal, recent studies have shown that the lowering methods perform significantly better than the depression filling techniques, in terms of the impact on the terrain attributes¹².

When the terrain is modeled as a TIN, only a few algorithms have been presented to deal with the problem of local minima. This is in spite of the fact that for hydrologic modelling, TINs are considered superior to DEMs, as shown by the experiments of Vivoni et al.²⁵. Theobald and Goodchild²³ show experimental results on the number of local minima produced by different methods to extract TINs. Liu and Snoeyink¹³ present an algorithm to simulate the flooding of a TIN, a problem that, although different, is related to removing local minima by pit filling. A recent work of Agarwal et al.¹ addresses the same problem, but apply I/O efficient algorithms. A different approach against local minima is the one followed by de Kok et al.⁴ and Gudmundsson et al.⁸. Instead of modifying the elevation of the points, they choose the edges of the triangulation in such a way that the number of local minima is minimized. They optimize over a particular class of well-shaped triangulations, the *higher-order Delaunay triangulations*⁸.

In this paper we present algorithms to remove local minima from TINs by modifying the heights of the vertices. We study both lifting points (pit filling) and lowering points (breaching). These can be seen as the TIN analogs of the two most important approaches for local minima removal in GIS. In both cases we want to remove local minima while modifying the terrain as little as possible. To formalize this second goal, we introduce a cost function that is applied to the vertices whose height is modified. The objective is to minimize the total cost of the removal, in order to obtain a terrain that resembles the original one as much as possible. To our knowledge, no previous paper deals with optimization for local minima removal.

There is no obvious choice for this measure of the cost of changing the terrain, and many of them are reasonable. The one adopted throughout most of this paper is the total displacement of the vertices. This is a natural and flexible measure:

every vertex can be moved, but all changes introduced are considered in the cost function, thus vertices will be moved as little as possible. Other possible measures include the maximum displacement, the number of vertices moved, and the total volume change. The first one encourages (many) small displacements over (a few) larger ones. The second one can help limit the extent of change over the terrain, but at the cost of allowing arbitrarily large changes for vertices that need to be moved. The third one provides a way to measure the difference between the original and the new terrain, and is closely related to minimizing the total displacement. Adaptations of our results to these other measures are discussed in Section 3.5.

The different possibilities for the cost function give rise to different problems. Furthermore, another source of variants of the problem is choosing *what* local minima to remove. Possible options are: removing *a given subset* of the local minima, removing *all* of them, or removing the *cheapest k* , for a parameter k . When the removal method is lifting, the three options can be solved on a one-by-one basis, that is, by removing each of the local minima separately. This is possible because the removal of one minimum does not affect the removal of the others. For lowering, the situation is different, because the way a local minimum is removed may affect the cost of removing other minima.

We study both approaches, lifting and lowering, independently. Some comments on their combination are made in Section 4. For the lifting approach, we show how the use of *contour trees* allows to remove all (but the lowest) local minima in $\mathcal{O}(n \log n)$ time, by facilitating the location of the vertices that must be used to remove each minimum. This gives an efficient and practical algorithm for pit filling in TINs, which is, as mentioned before, the most widely used method to deal with local minima in the GIS community. The lowering approach turns out to be much harder than the lifting version. We start by showing that removing optimally one local minimum, or a constant number of them, while minimizing the total displacement, can be done in polynomial time. Then we show that removing all (but the lowest) local minima is NP-hard, and propose an approximation algorithm to solve the problem, based on an existing algorithm for the Node-Weighted Steiner Tree Problem.

We begin by studying the simplest of the two, the lifting approach, and then we focus on the lowering technique.

2. Removing Local Minima by Lifting

In this section we present an efficient algorithm to remove local minima by increasing the elevation of some of the vertices. This can be seen as a flooding or pit filling technique for TINs.

We begin with a few basic definitions that will also be used in the next sections. A *polyhedral terrain* T , or just *terrain*, is a triangulated point set in the plane where each point or vertex v has a height, denoted $h(v)$. Any terrain has an associated graph, G_T . Sometimes we will refer to both the terrain and the associated graph as

the *terrain*.

A (local) *minimum* is a maximally connected set of vertices $M \subset T$ such that all the vertices in M have the same height, and such that no vertex in M has a neighbor with a lower height.

Even though a minimum can be made of more than one vertex, for the purpose of this paper it is more convenient to treat each minimum as consisting of only one vertex. For example, if a minimum at vertex u is lifted to height h , we will assume that also all the other vertices of the minimum u belongs to are lifted in the same way. The same is done with the definition of *saddle*: below we define it as being one vertex, but in practice it can be a connected set of them. This does not affect our algorithms or their running times. These considerations apply to the whole paper. From now on, we treat each minimum or saddle as consisting of one vertex only.

A *saddle* is a vertex that has some neighboring vertices around it that are higher, lower, higher, lower, in cyclic order around it. In other words, a saddle vertex v has two higher neighbors and two lower neighbors, which alternate around v . To simplify the presentation of the algorithms, we will assume the terrain has only one global minimum, and we will adopt the convention that when we refer to *local* minima, we do not include the *global* minimum.

The cheapest way to remove a local minimum at a vertex v , with height $h(v)$, is by lifting it to $h(w)$, where w is the lowest neighbor of v . However, this may turn w into a local minimum. To fix this, the lifting procedure must be propagated until no new local minima are left.

Conceptually, the idea is as follows. We explain how to compute a list $S = \{s_1, \dots, s_k\}$ of vertices that must be lifted to remove a local minimum at v . Initially, $S = \{s_1 = v\}$, and it is expanded every time the lifting must be propagated. Let $U = \{u_1, \dots, u_m\}$ be the union of the neighbors of all vertices in S that are not in S themselves, and denote the vertex in U with lowest height with u_{\min} . We raise all vertices in S such that $h(s_i) = h(u_{\min})$ for $i \in \{1, \dots, k\}$. If u_{\min} is a saddle vertex of the terrain, then it is connected to another lower vertex and we are done. Otherwise, we remove u_{\min} from U and add it to S as s_{k+1} , and we set k to $k + 1$. Next, we add the neighbors of the new s_k that are not already in S to U . After the changes made to S and U , u_{\min} refers to the new lowest vertex in U . This iterative approach lifts the whole basin of the local minimum, in a bottom-up fashion, until it reaches its lowest saddle vertex.

The propagation of the lifting is facilitated by the variation of the *contour tree* used by van Kreveld et al.²⁴ Contour trees have been previously used in image processing and GIS research^{5,7,20,21} and are also related to the *Reeb graph* used in Morse Theory^{17,19}. A contour is a connected component of the level set of the height function of the terrain, at some height h_0 , (more precisely, $\{x \in R^2 | h(x) = h_0\}$). The contour tree is a tree that captures the changes in the topology of the contours of the terrain, as the height h changes. As h varies, the contour tree keeps track of contours that appear, join, split, and disappear. Minima and maxima in the terrain are represented by leaves in the contour tree, and saddle vertices in the terrain

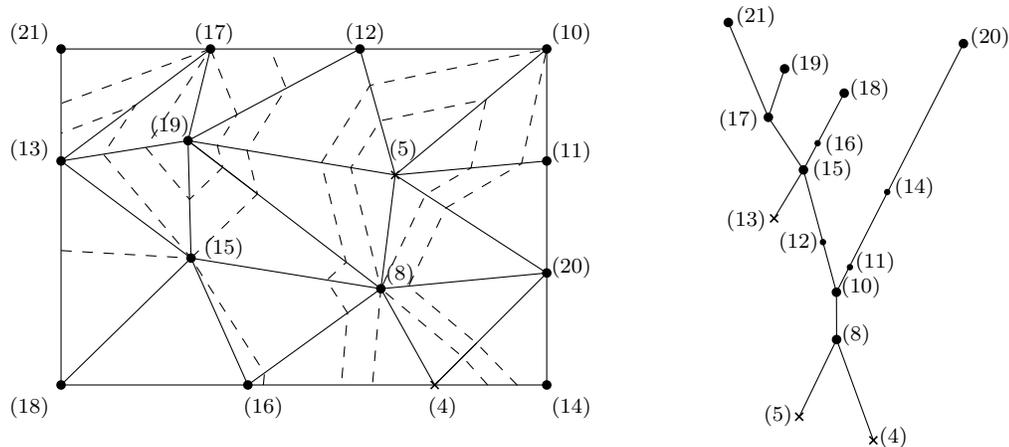


Fig. 1. Left: example of a terrain showing the elevation of the vertices (between parenthesis) and some of the contour lines. Right: the augmented contour tree of the terrain.

correspond to nodes of degree three or higher. Ordinary, non-critical vertices appear as vertices of degree two. See Figure 1 for an example. The contour tree for a terrain with n vertices can be computed in $\mathcal{O}(n \log n)$ time^{2,24}.

To remove a given local minimum at v in the terrain by propagated lifting, we look at the corresponding leaf v' in the contour tree. Let w' be the node of degree three or more in the contour tree that is closest to v' . It corresponds to a saddle w in the terrain, the first one encountered when “flooding” v . To remove the local minimum at v , we must lift all the vertices in the terrain that correspond to nodes on the path from v' to w' in the contour tree, including v' , but excluding w' , to the height of the saddle vertex.

After this change we must update the contour tree to reflect the changes. The branch that ended at v' disappears, and the nodes on it become ordinary (non-critical) ones, at the same height as the saddle node. If necessary, we can store any relevant information about the lifted vertices, like total displacement, together with the saddle node. If before the lifting step the saddle had only one downward branch, then after the lifting it became a new local minimum, and the lifting needs to be propagated until the lowest of the saddle ancestors is reached (there could be more than one). If before the lifting step the saddle had two or more downward branches, then the saddle might continue as a saddle or become an ordinary node. In both cases this lifting step is over. In any case, during the removal of the local minima every node involved is processed only once, because after lifting it, it becomes “part” of the saddle node.

As an example, consider the terrain and contour tree of Figure 1. For the explanation we will use their heights to refer to the vertices. Suppose the minimum 4 is removed first. Only that vertex is lifted, to height 8. After that, 8 is not a saddle

anymore, and becomes an ordinary vertex. If the local minimum to be removed next is 5, then it must be lifted to the height of the next saddle, 10, together with the two vertices at height 8. But that is not enough, because that would turn saddle 10 into a new local minimum. Hence the lifting needs to continue to height 15, lifting also 11 and 14 to the same height.

Note that this way to remove the local minima by lifting has minimum cost. Any alternative way to remove a local minimum, by lifting, implies lifting the content of the whole basin to the some higher saddle vertex, incurring in a larger displacement.

Removing all the local minima, given the contour tree, can be done in linear time. Hence the total running time equals the time needed to build the tree.

Theorem 1. *Given a terrain T with n vertices, a lifting of the vertices that removes all local minima while minimizing the total displacement of the vertices can be computed in $\mathcal{O}(n \log n)$ time.*

3. Removing Local Minima by Lowering

In this section we study how to remove local minima by lowering the heights of some vertices, and, at the same time, minimize the total displacement of the vertices. For the lifting approach, it was relatively simple to figure out what vertices had to be lifted to remove a given minimum. In the case of lowering this is not immediate at all, because, in principle, any neighbor of the minimum may be the first vertex to be lowered. We begin by defining some appropriate terminology and making some basic observations.

3.1. Preliminaries

Any vertex in a terrain can be *lowered*, meaning that its height can be decreased. The *cost* of lowering a vertex is defined as the difference between its original height and the new one. Some other suitable definitions are discussed in Section 3.5. Recall that each local minimum is treated as consisting of exactly one vertex.

A given local minimum at u is removed by lowering some neighboring vertex to a height less than or equal to $h(u)$. Since the lowered vertex can become a local minimum itself, and we do not want to create new local minima (or make an existing one worse), a propagation takes place, until the original and the newly created local minima are removed. Figure 2 shows an example. Observe that any given local minimum can be removed in this way (recall that we do not consider the global minimum to be a local minimum). The same can be done for any set of local minima that does not include the lowest minimum in the terrain.

Since the lowest minimum can never be removed by lowering (without creating a new local minimum), we will talk about removing *all* local minima, but we will mean removing *all but the lowest* local minimum.

The following definitions formalize the basic ideas related to lowering.

Definition 1. Let T be a terrain, and let u and v be two vertices of T , with

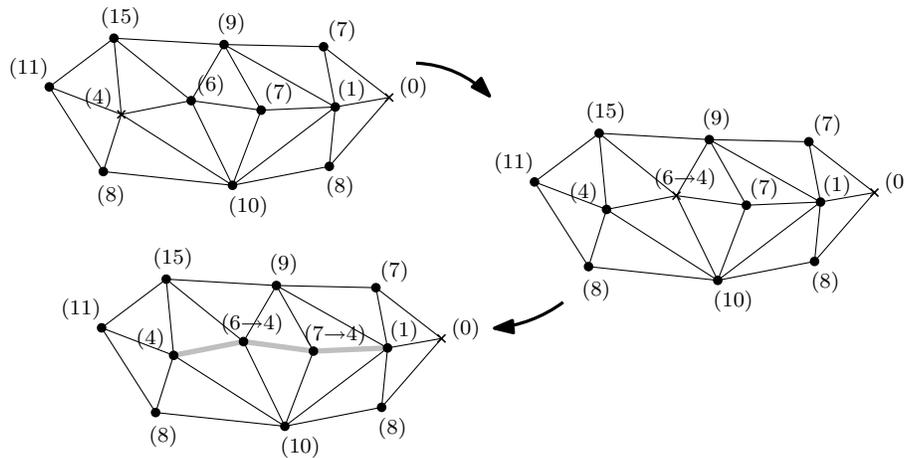


Fig. 2. Removing the local minimum at height 4 by lowering. Top: initial heights (between parenthesis). The change in height of the lowered vertices is shown with an arrow. Middle and bottom: first a vertex is lowered from 6 to 4, turning it into a new local minimum. To remove it, a neighboring vertex is lowered from 7 to 4. A breaching path from the vertex of height 4 to the one of height 1 is highlighted in gray. Note that this way to remove the minimum by lowering, with total cost 5, is optimal.

$h(u) > h(v)$. A *breaching path* from u to v is a tuple $\rho = (P, D)$, where P is a list of vertices that induce a path between u and v , that is, $P = \{u, w_1, w_2, \dots, w_\eta, v\}$, and $D = \{0, d_1, d_2, \dots, d_\eta, 0\}$ is a list of height displacements for the intermediate vertices of the path, such that $h(u) \geq h(w_1) + d_1$, $h(w_i) + d_i \geq h(w_{i+1}) + d_{i+1}$ for every $1 \leq i < \eta$, and $h(w_\eta) + d_\eta \geq h(v)$. The *cost* of a breaching path is $\text{Cost}(\rho) = \sum_i |d_i|$.

Intuitively, a breaching path is a path used to remove a local minimum at u , by connecting it to a lower vertex v , by modifying the heights of all the vertices in between (see Figure 2). A natural extension of the concept of breaching path is the *breaching graph*. A breaching graph connects a set of local minima with each other, removing all of them (but the lowest one).

Definition 2. Let T be a terrain. A *breaching graph* is a tuple $\psi = (P, D)$, where P is a set of vertices that induce a subgraph of T , such that all the vertices of degree one are minima, and each vertex with degree higher than one has a height displacement in D such that for each connected component, ψ includes a breaching path connecting each of its local minima to the lowest minimum of the component. The cost of the breaching graph is defined as $\text{Cost}(\psi) = \sum_i |d_i|$.

The problem of removing all local minima can now be restated as finding a minimum cost breaching graph connecting them. Observe that since we aim at removing *all* local minima, the breaching graph is always connected, because all the local minima must be connected to the global minimum of the terrain.

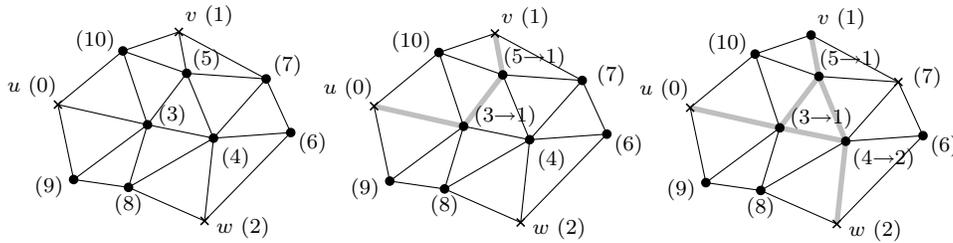
8 *Rodrigo I. Silveira and René van Oostrum*

Fig. 3. From left to right: a terrain with three local minima: at u, v and w ; a breaching path from v to u ; a breaching graph of minimum cost that connects u, v and w .

See Figure 3 for an example. Note that it is possible for a minimum cost breaching graph not to be a tree. However, it can be turned into a tree by discarding some edges, without changing its cost. This can be done by discarding an edge from each cycle as follows. Take the highest vertex in the cycle (highest according to the modified heights), and discard one of the two cycle edges incident to it. This is always possible, and guarantees that at least one breaching path is preserved for each local minimum. For this reason we will sometimes refer to a minimum cost breaching *tree*. Throughout this paper we refer to *connecting* two minima u and v , meaning that the minimum at u or v (the highest one) is removed by creating a breaching path connecting u and v .

Since we are minimizing the total displacement, each vertex will be lowered as little as possible. Hence if a vertex w_i needs to be lowered to connect a local minimum at u to a lower vertex v , the new height of w_i will be $h(u)$.

In a breaching graph it can occur that an intermediate vertex w_i is part of more than one breaching path. See for example the vertex of initial height 3 in Figure 3. In that case the new height of the vertex must be set to the height of the lowest of those minima. We introduce the concept of *paying* to denote this relation between intermediate vertices and local minima. We say that u *pays* for the lowering of a vertex w_i if u is the lowest local minimum that is removed by a breaching path through w_i .

It is interesting to look at the structure of an optimal solution to our problem, in relation to how the payment of the lowering is distributed among the local minima. As mentioned before, any optimal solution is a breaching graph, which induces a connected subgraph of T . Figure 4 shows an example of an optimal breaching graph. Notice that the breaching path that removes the second lowest minimum (vertex u) is paid entirely by that minimum. That is always the case. For some higher local minima, like v or w in the example, part of the costs related to removing them might be paid by lower minima. In the example, local minimum v only pays for the lowering of the vertex with initial height 5.

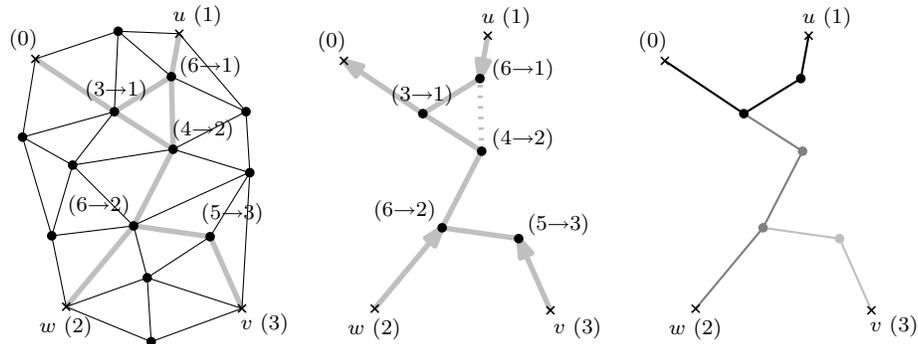


Fig. 4. Left: terrain with four local minima, together with the breaching graph of an optimal solution (only the relevant heights are shown). Center: the breaching graph seen as a tree, with the removal direction shown. Right: payment of the lowering the intermediate vertices. The black vertices are paid by u , the medium gray vertices by w , and the light gray vertex by v .

3.2. Removing one local minimum

We begin with the problem of removing one given local minimum u . As mentioned before, this is always possible as long as u is not the lowest one. We assume that is the case.

The goal is to remove u by lowering some vertices while minimizing the total displacement. Using the terminology introduced in the previous section, we are looking for a minimum cost breaching path from u to some lower vertex. Notice that any vertex that is not a local minimum has a breaching path of zero cost from it to some lower minimum. For this reason it is the same if we look for a minimum cost breaching path from u to some other lower minimum (this will simplify the presentation of the algorithms).

The cheapest way to connect u to a lower minimum can be found by looking for the shortest paths between u and each of the lower minima in a directed graph based on the terrain. The vertices and edges are the same as in the terrain, but edges are made directed: each undirected edge \overline{pq} is replaced by two edges \overrightarrow{pq} and \overleftarrow{qp} , and the weight of edge \overrightarrow{pq} is set to $\max\{0, h(q) - h(u)\}$, that is, if q is higher than u , it is the vertical distance between q and u . Once this graph is built, any standard single-source shortest path algorithm, such as Dijkstra's, can be used to find the minimum cost path from u to some other, lower, local minimum. This can be done in $O(n \log n)$ time.

Theorem 2. *Given a terrain T with n vertices and a local minimum u , a breaching path of minimum cost removing u can be found in $O(n \log n)$ time.*

In order to use an algorithm like the previous one to remove more than one local minimum, the shape of the solutions must be brought into play. For two given local minima u_1, u_2 , a minimum cost breaching graph can have only one of two different shapes: it can be made of two disjoint paths, or it can be Y-shaped,

meaning that it has two branches that are joined in a junction vertex and then continue together until the local minimum that is used to remove the lowest of u_1, u_2 . When the number of minima to be removed grows, the number of shapes increases exponentially, so trying all the possible shapes for the minimum cost breaching tree leads to a polynomial time algorithm only when the number of local minima is a constant. As will be shown in the next section, the problem becomes NP-hard when the number of local minima is not constant.

3.3. Removing all local minima

In this section we prove that the problem of removing all local minima from a terrain by lowering, while minimizing the total displacement, is NP-hard.

We use a reduction from Planar Connected Vertex Cover (PCVC), which is known to be NP-hard⁶. The optimization version of PCVC consists in given a planar graph $G = (V, E)$, finding a set of vertices of minimum size such that every edge has at least one end among the selected vertices, and the subgraph induced by the set is connected.

We show how to solve any instance of PCVC, given by a planar graph $G = (V, E)$, with an algorithm for our problem. We create a new graph G' as follows. See Figure 5 for an illustration. Take G as the initial graph G' . Assign height 1 to each existing vertex. Then for each edge $e \in E$, create a vertex somewhere in the middle of the edge at height 0 (these will be all minima). Finally, change the height of an arbitrary local minimum to -1 , to create a global minimum. The resulting graph has exactly $|E|$ minima. Each of them corresponds to an edge in G that must be covered by the vertex set. To turn the graph into a terrain, we compute a first arbitrary triangulation, and for every edge added during the triangulation, we add a vertex on its midpoint at height $+\infty$. The resulting non-triangular faces are triangulated in some arbitrary way. This guarantees that all the new edges have one endpoint with a vertex at height $+\infty$, so they will never be part of an optimal solution. It is straightforward to see that the whole construction can be done in polynomial time.

Removing all local minima in G' induces a set of vertices that must be lowered. If the local minima are removed in an optimal way, the set of lowered vertices is an optimal vertex set of G . The fact that all local minima have been removed implies that all edges have one end in the chosen set, so it is indeed a solution to Vertex Cover, and it is a tree, hence also connected. It can be easily verified that it is also optimal. Any connected vertex cover of G induces a lowering of some vertices (the ones in the result set). The vertex cover is connected and must cover all edges, hence every local minimum is adjacent to a vertex that is selected. If we lower all the vertices in the vertex cover, all local minima will get removed because they will be all connected to the global minimum (height -1). The cost of the vertex cover is exactly the total displacement needed to remove all local minima. Therefore the vertex cover induced by the removal of the local minima must be a minimum

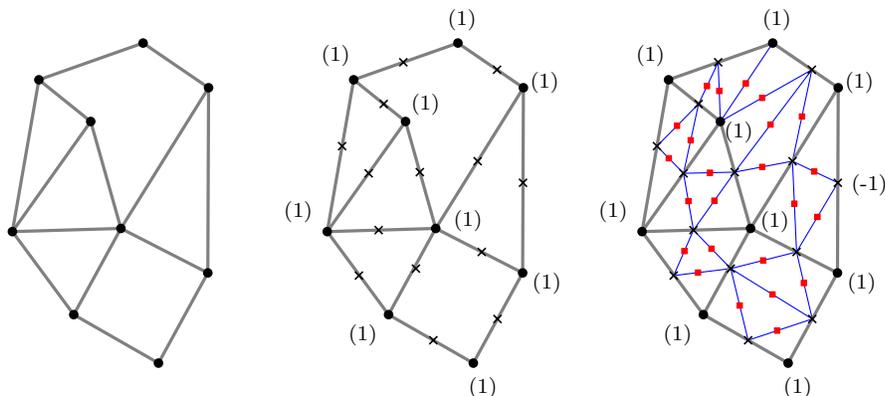


Fig. 5. NP-hardness construction. Left: original graph. Center: one local minimum is added on each edge. Right: a first arbitrary triangulation is computed, and a vertex at height $+\infty$ is added on each new edge (drawn with squares). The remaining faces can be triangulated in any way.

(connected) vertex cover, otherwise there would be a cheaper way to remove the local minima.

Theorem 3. *Given a terrain T with n vertices, it is NP-hard to compute a lowering of the vertices that removes all local minima, which minimizes the total displacement of the vertices.*

3.4. Approximation algorithm

As was shown in the previous section, the problem of removing all the local minima in a terrain through lowering while minimizing the total displacement is NP-hard. This motivates the search for approximation algorithms.

One of the simplest options that arise is computing an optimal breaching path for each of the k local minima, and then merging them. However, this leads to a k -approximation. A simple example where this factor is attained is shown in Figure 6.

The inherent difficulty of the problem of removing a set of local minima lies in finding the vertices that act as junctions of the different breaching paths. This resembles the Steiner Tree Problem, and in particular, the Node-Weighted Steiner Tree (NWST) problem in networks, which is a more general version of the classical Steiner Tree problem where the costs are assigned to the vertices instead of to the edges. Even though constant factor approximation algorithms are known for the standard Steiner Tree problem, no approximation algorithms with factor less than logarithmic exist for the NWST problem, unless $NP \subseteq DTIME[n^{O(\text{polylog } n)}]$ ¹¹. Our problem is still different from the NWST problem because the cost paid for using a vertex is not fixed. It depends on the heights of the local minima that are being removed through it. Any fixed-cost approach like, for example, assuming that all the lowering is paid by the second lowest vertex, may result in a solution a linear

vertex. This is repeated until only the global minimum is left.

The subset of local minima is chosen such that the average cost of removing the minima is minimized. This is made more precise next.

Each iteration of the algorithm selects (i) a center vertex c , and (ii) a subset of the current local minima F . The algorithm will consider connecting each local minimum in F to c . It is important to observe two aspects of this way to connect a subset of local minima. Firstly, the lowest minimum in F , $v^{(1)}$, if lower than c , will not be removed at this iteration, but will serve to remove all other minima in F .^a Secondly, the second-lowest minimum in F , $v^{(2)}$, will pay for lowering c and connecting c to $v^{(1)}$. For the other local minima, the cost incurred is the one to connect the minimum to c (not including lowering c), and is paid by that minimum.

The subset of local minima, at each iteration, is chosen as the one minimizing the following ratio

$$\frac{\text{cost connecting } v^{(2)} \text{ to } v^{(1)} \text{ via } c + \text{cost connecting each other minimum in } F \text{ to } c}{|F|} \quad (1)$$

Note that the cost of connecting each minimum in F to c is computed based on the initial heights, and is computed individually for each minimum in F . The ratio gives an upper-bound on the average cost to connect each minimum to c . In practice, the cost computed with Equation (1) can be higher than the actual cost of connecting all minima in F to c , because Equation 1 does not take into account that some minima may share part of the breaching path to c .

Next we explain how to implement an iteration of the algorithm.

In a preprocessing step, we compute the lowest-cost paths from each minimum to all the other vertices (note that cost is always taken in the breaching path sense). Then we compute for each vertex, a list with the k minima sorted by cost. Using a linear-time shortest-path algorithm¹⁰, the preprocessing takes $O(nk \log k)$ time.

The pseudo-code of an iteration of the algorithm is given in Algorithm 1. Every pair of center vertex and second-lowest minimum ($c, v^{(2)}$) is tried. $v^{(1)}$ is chosen as the local minimum—lower than $v^{(2)}$ —that is cheapest to connect to $v^{(2)}$ via c . Then the set F is augmented by adding the other (higher) minima by increasing cost. This guarantees that, at the end of the algorithm, the subset of minima that gives the lowest ratio will be found.

Regarding the running time, the execution of one iteration takes $O(k^2n)$ time, because the sorted distances are precomputed. Since each iteration removes at least one minimum, the total running time of the algorithm, including the preprocessing, amounts to $O(k^3n)$.

^aTo simplify the explanation, we are assuming that $v^{(1)}$ is lower than c , and that there is only one second-lowest minimum. The adaptations needed if that is not the case are straightforward.

Algorithm 1 Finds a center and subset of minima minimizing Equation (1)

```

1: for each vertex  $c$  do
2:   for each minimum  $v^{(2)}$  do
3:      $v^{(1)} \leftarrow$  nearest minimum lower than  $v^{(2)}$ , when connecting via  $c$ 
4:      $F \leftarrow \{v^{(1)}, v^{(2)}\}$  // Initial subset
5:      $\text{cost} \leftarrow$  cost of connecting  $v^{(2)}$  to  $v^{(1)}$ , via  $c$  // The current ratio is  $\text{cost}/2$ 
6:     Let  $\{d_3, \dots, d_m\}$  be the sorted distances from  $c$  to the minima higher or
       equal to  $v^{(2)}$ 
7:     for  $j = 3 \dots m$  do
8:        $v^{(j)} \leftarrow$  minimum associated with  $d_j$ 
9:       // Add it to current group of minima
10:       $F \leftarrow F \cup \{v^{(j)}\}$ 
11:       $\text{cost} \leftarrow \text{cost} + d_j$ 
12:      // Ratio for this center and subset
13:       $\text{ratio} \leftarrow \text{cost}/j$ 
14:     end for
15:   end for
16: end for
17: Select the  $F$  and  $c$  associated with the lowest ratio

```

3.4.2. Approximation factor

Our proof of the approximation factor follows the proof by Klein and Ravi¹¹. We first define some notation. T_i is the terrain just after iteration i . The number of local minima (not yet removed) in T_i is denoted by k_i . The number of minima connected at iteration i (which is one more than the number of local minima removed at that iteration) is denoted λ_i (thus in the context of the algorithm, $\lambda_i = |F|$). The cost of the lowering done at iteration i is C_i . Finally, OPT is the cost of an optimal solution. Since any solution induces a breaching graph of T , which can be seen as a tree, we will refer to the *optimal tree*, meaning a tree associated with an optimal solution.

In order to prove the approximation factor, we will relate the ratio of the combination chosen at step i , to the ratio of an optimal solution. This is stated in the following lemma.

Lemma 1. *At any iteration i of the algorithm,*

$$\frac{C_i}{\lambda_i} \leq \frac{OPT}{k_{i-1}} \quad (2)$$

Proof. Let Ψ be a tree of an optimal solution. Some of the vertices in Ψ may correspond to local minima that have been already removed in the current terrain (T_{i-1}). Let Ψ_i be a tree based on Ψ where all the leaves that correspond to local minima that have been removed in T_{i-1} have been deleted, together with all the

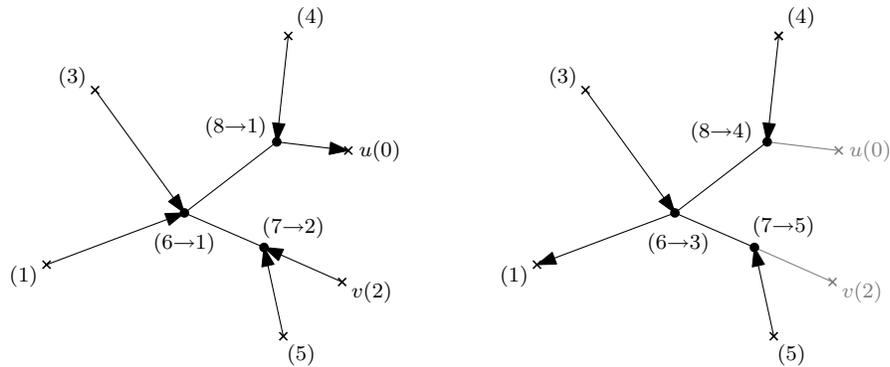


Fig. 7. Left: the tree of an optimal solution, Ψ , showing the original and final height of the vertices lowered. Right: Ψ_i , assuming that in the current terrain u and v have been already removed. As the example shows, the cost of Ψ_i (9) can be different from the one of Ψ (17), but it can never be higher.

paths that connected them to the rest of the tree—we keep the smallest connected subtree of Ψ of the current minima. If there is a removed local minimum in Ψ that is not a leaf, we treat that vertex as a ordinary one (non-local minimum).

Let $\text{Cost}(\Psi_i)$ be the cost of Ψ_i , in the same way as it is defined for breaching graphs: the cost of removing all local minima in Ψ_i by lowering only vertices of the tree. We first observe that $\text{Cost}(\Psi_i) \leq \text{OPT}$. This is because Ψ_i contains less leaves than Ψ , and because the lowest-minimum in Ψ_i must be higher than the one in Ψ , thus the total displacement can never increase. An example is shown in Figure 7.

Given a tree and a subset of its vertices M , $|M| \geq 2$, there is always a spider decomposition of M contained in the tree¹¹. Consider a spider decomposition of Ψ_i , where M are the minima in Ψ_i . Let c_1, \dots, c_r be the centers of the spiders in the decomposition. For a spider with only two leaves (a path), we pick any vertex in the path as its center. Let the cost of the spider S_j , centered at c_j , be $\text{Cost}(S_j)$, and let n_j be the number of minima that it connects. We define the *ratio* of S_j as $\text{Cost}(S_j)/n_j$.

Next we show that the ratio of the minima selected by the algorithm, at iteration i , is better or equal to the ratio of any spider S_j .

This can be seen as follows. Let S_j be any spider in the spider decomposition, with center c_j . During iteration i of the algorithm, vertex c_j will be considered as a possible center vertex to connect a subset of the local minima. For that particular center, Algorithm 1 will find a subset of minima to remove that minimizes Equation (1). The breaching graph associated with the local minima selected by the algorithm may not be a spider. This is because it is not guaranteed that the individual breaching paths that connect the local minima to the center vertex are disjoint. In fact, the algorithm considers a larger class of breaching graphs, which include all spiders. Therefore it will always find a subset with a ratio at least as good

16 *Rodrigo I. Silveira and René van Oostrum*

as the one of the spider centered at c_j . Then for each spider S_j in the decomposition we have

$$\frac{C_i}{\lambda_i} \leq \frac{\text{Cost}(S_j)}{n_j}$$

Rewriting and summing over all the spiders in the decomposition we get

$$\frac{C_i}{\lambda_i} \sum_j n_j \leq \sum_j \text{Cost}(S_j) \quad (3)$$

Now we argue that $\sum_j \text{Cost}(S_j) \leq \text{Cost}(\Psi_i)$. The cost of a spider S_i , $\text{Cost}(S_i)$, can be different from the cost of the associated subgraph in Ψ_i . Some vertices may have to be lowered in a different way, because all minima in S_i will have to be removed by connecting them to the lowest vertex in S_i . However, using the same arguments used to claim that $\text{Cost}(\Psi_i) \leq \text{OPT}$, it follows that the total cost needed to connect all the minima in S_i can never be more than what is paid to remove the same minima in the associated subgraph of Ψ_i . Hence $\sum_j \text{Cost}(S_j) \leq \text{Cost}(\Psi_i)$.

Going back to Equation (3), and using that $\text{Cost}(\Psi_i) \leq \text{OPT}$ and the fact that the sum $\sum_j n_j$ equals the number of minima at the beginning of the current iteration, k_{i-1} , we get $(C_i/\lambda_i)k_{i-1} \leq \text{OPT}$, which is (after rewriting) the result claimed. \square

To get the approximation factor, exactly the same arguments used by Klein and Ravi¹¹ can be used to conclude that if p is the total number of iterations of the algorithm, then $\sum_{j=1}^p C_j \leq 2 \ln k \cdot \text{OPT}$. An example showing that the approximation factor is nearly tight can be constructed in a way very similar to the one presented by Klein and Ravi¹¹.

Theorem 4. *Given a terrain T with n vertices, k of them minima, a lowering of the vertices that removes all the local minima, which minimizes the total displacement of the vertices, can be computed in $\mathcal{O}(k^3 n)$ time, where the total displacement is at most $2 \ln k$ times the minimum one.*

3.5. Other measures

Even though most of this paper considered the total displacement as the measure being optimized, there are several other measures that are also interesting. We comment briefly on some of them.

Maximum displacement. Minimizing the maximum displacement, that is, the maximum height that any vertex is moved, is an interesting criterion. The adaptations needed for the algorithm of Section 3.2 to work for this variant are straightforward. Moreover, for this measure the solution for k local minima can be obtained by simply merging the optimal breaching paths of each minimum. Hence

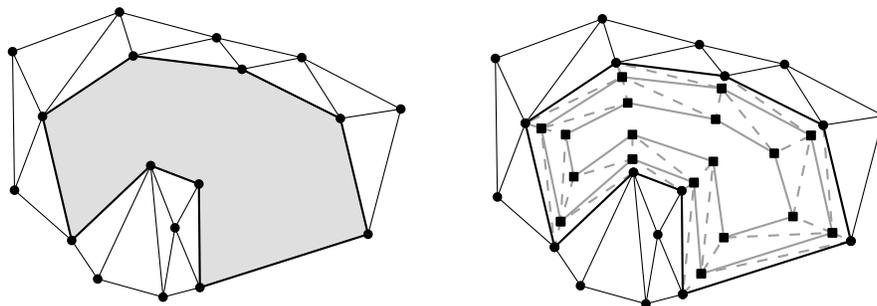


Fig. 8. Left: a face of the input graph (in gray) that needs to be triangulated to turn the graph into a valid terrain. Right: a series of smaller copies of the face, two in the example, are created and triangulated; the center hole still not triangulated can be triangulated in any way.

the problem of removing all minima (or more generally, a given set of k local minima) while minimizing the maximum displacement, is solvable in $O(kn \log n)$ time.

Number of vertices modified. It might be important to minimize not the amount of displacement but the number of vertices that are modified. This can be seen as a particular case of the problem studied before, where all the heights are either 0, for the local minima, or 1, for the other vertices. The NP-hardness proof of Section 3.3 can also be used to show that this problem is NP-hard. Only the triangulation part needs to be slightly different. Figure 8 shows a way to adapt it, by adding a series of smaller, concentric, copies of the face that needs to be triangulated, and connecting them by triangles in some simple way. For a face with r vertices, around $r/4$ inner copies are sufficient. The approximation algorithm can also be adjusted to this version of the problem.

Total volume reduction. Minimizing the total decrease in volume due to the lowering is also a criterion of interest, but it is in essence very similar to minimizing the total displacement, because the volume change related to the lowering of one vertex is independent from the other vertices. Both the NP-hardness proof and the algorithms presented above can be adapted to work for this version of the problem.

4. Conclusions and Future Work

This paper studied the removal of local minima from triangulated terrains by modifying the heights of the vertices. Two major techniques were analyzed, lifting and lowering, with the objective of removing all local minima while minimizing the total displacement of the vertices. For the lifting technique, we showed how to use contour trees to facilitate finding which vertices need to be lifted to remove each local minimum.

The lowering technique presented many interesting challenges. We showed that one local minimum can be removed efficiently, but as soon as the number of minima is not constant the problem becomes NP-hard. With that in mind, we proposed an

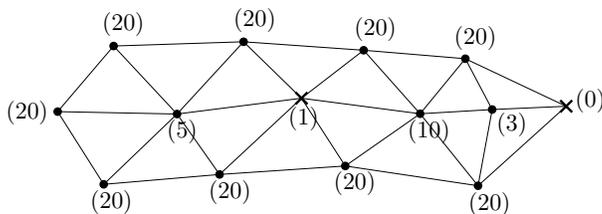


Fig. 9. Three approaches to remove the local minimum at height 1: (i) by lifting: (1→10) and (5→10) (cost 14), (ii) by lowering: (10→1) and (3→1) (cost 11), and (iii) by combining both: (1→5) and (10→5). The combined approach, with cost 9, outperforms the other two.

approximation algorithm with factor $2 \ln k$.

There are many directions for further research, specially for the lowering approach. Approximation algorithms with better factors are one of them. There are some better approximation algorithms for the Node-Weighted Steiner Tree problem, like the ones of Guha and Khuller⁹, which improve the $2 \ln k$ factor of Klein and Ravi¹¹ to $1.5 \ln k$ or even $(1.35 + \epsilon) \ln k$, but it is unclear how to adapt them to our problem.

In this paper the approaches of lifting and lowering were analyzed separately. However, there are cases where a combination of both can result in a smaller total displacement. A simple example is shown in Figure 9. Moreover, such an approach would make more sense since both the underestimation and overestimation error present in the terrain could be compensated.

The combined problem can be easily shown to be NP-hard, since it generalizes the lowering approach. Moreover, it seems to get considerably more complicated. To illustrate this, take the problem of removing one single given local minimum. For the lowering approach this can be solved rather easily by reducing it to a shortest path problem. In the combined version, where any vertex can be either lowered or lifted, we are no longer looking for an optimal path to remove the minimum. An optimal solution will consist of some vertices in the basin of the minimum—including the minimum itself—being lifted to some height h and then a minimum cost breaching path connecting the lifted vertices to a lower minimum. The height h can be anything between the height of the minimum and the height of the lowest saddle vertex of the basin (note that the extreme values correspond to the *pure* lowering and lifting approaches). Therefore in order to find the optimal way to remove a single minimum, we must find the value of h that minimizes the cost of the lifting plus the cost of the lowering. $\mathcal{O}(n)$ different values are possible. For the general case of removing all local minima, it is not clear how to integrate this two-step removal into the approximation algorithm here presented, while maintaining the approximation factor guarantee.

Finally, there are many other variants of the problem that can be worth studying in more detail, like removing not *all* but a *given set* of local minima (the approxima-

tion algorithm presented here can be adapted for that variant as well), constraining the vertical displacement of any vertex to be within some interval, and the use of different cost functions.

Acknowledgments. We thank Marc van Kreveld for proposing the problem and for many helpful suggestions.

References

1. P. K. Agarwal, L. Arge, and K. Yi. I/O-efficient batched union-find and its applications to terrain analysis. In *Symposium on Computational Geometry*, pages 167–176, 2006.
2. H. Carr, J. Snoeyink, and U. Axen. Computing contour trees in all dimensions. *Comput. Geom. Theory Appl.*, 24:75–94, 2003.
3. J. Charleux-Demargne and C. Puech. Quality assessment for drainage networks and watershed boundaries extraction from a digital elevation model (DEM). In *Proc. 8th Int. Symp. Advances Geographic Information Systems (ACM-GIS)*, pages 89–94, New York, 2000. ACM Press.
4. T. de Kok, M. van Kreveld, and M. Löffler. Generating realistic terrains with higher-order Delaunay triangulations. *Comput. Geom. Theory Appl.*, 36:52–65, 2007.
5. H. Freeman and S. P. Morse. On searching a contour map for a given terrain profile. *J. of the Franklin Institute*, 248:1–25, 1967.
6. M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, New York, NY, 1979.
7. C. Gold and S. Cormack. Spatially ordered networks and topographic reconstructions. In *Proc. 2nd Internat. Sympos. Spatial Data Handling*, pages 74–85, 1986.
8. J. Gudmundsson, M. Hammar, and M. van Kreveld. Higher order Delaunay triangulations. *Comput. Geom. Theory Appl.*, 23:85–98, 2002.
9. S. Guha and S. Khuller. Improved methods for approximating node weighted Steiner trees and connected dominating sets. *Information and Computation*, 150:57–74, 1999.
10. M. R. Henzinger, P. N. Klein, S. Rao, and S. Subramanian. Faster shortest-path algorithms for planar graphs. *J. Comput. Syst. Sci.*, 55(1):3–23, 1997.
11. P. Klein and R. Ravi. A nearly best-possible approximation algorithm for node-weighted Steiner trees. *J. Algorithms*, 19:104–115, 1995.
12. J. B. Lindsay and I. F. Creed. Removal of artifact depressions from digital elevation models: towards a minimum impact approach. *Hydrological Processes*, 19:3113–3126, 2005.
13. Y. Liu and J. Snoeyink. Flooding triangulated terrain. In *Proc. 11th Int. Symp. on Spatial Data Handling*, pages 137–148, 2004.
14. D. Mark. Network models in geomorphology. In M. G. Anderson, editor, *Modelling Geomorphological Systems*, chapter 4, pages 73–97. John Wiley & Sons, 1988.
15. L. W. Martz and J. Garbrecht. An outlet breaching algorithm for the treatment of closed depressions in a raster DEM. *Computers & Geosciences*, 25:835–844, 1999.
16. V. J. Rayward-Smith. The computation of nearly minimal Steiner trees in graphs. *Int. J. Math. Educ. Sci. Technol.*, 14:15–23, 1983.
17. G. Reeb. Sur les points singuliers d’une forme de pfaff complement integrable ou d’une fonction numerique. *Comptes Rendus Acad. Sciences Paris*, 222:847–849, 1946.
18. W. Rieger. A phenomenon-based approach to upslope contributing area and depressions in DEMs. *Hydrological Processes*, 12:857–872, May 1998.
19. Y. Shinagawa and T. L. Kunii. Constructing a Reeb graph automatically from cross sections. *IEEE Comput. Graph. Appl.*, 11:44–51, Nov. 1991.

20 *Rodrigo I. Silveira and René van Oostrum*

20. J. K. Sircar and J. A. Cerbrian. Application of image processing techniques to the automated labelling of raster digitized contours. In *Proc. 2nd Internat. Sympos. Spatial Data Handling*, pages 171–184, 1986.
21. S. Takahashi, T. Ikeda, Y. Shinagawa, T. L. Kunii, and M. Ueda. Algorithms for extracting correct critical points and constructing topological graphs from discrete geographical elevation data. In *Eurographics'95*, volume 14, pages C–181–C–192, 1995.
22. A. Temme, J. Schoorl, and A. Veldkamp. Algorithm for dealing with depressions in dynamic landscape evolution models. *Computers & Geosciences*, 32:452–461, 2006.
23. D. Theobald and M. Goodchild. Artifacts of TIN-based surface flow modeling. In *Proc. GIS/LIS90*, pages 955–964, 1990.
24. M. van Kreveld, R. van Oostrum, C. Bajaj, V. Pascucci, and D. Schikore. Contour trees and small seed sets for isosurface generation. In S. Rana, editor, *Topological Data Structures for Surfaces*, chapter 5, pages 71–85. Wiley, 2004.
25. E. Vivoni, V. Ivanov, R. Bras, and D. Entekhabi. Generation of triangulated irregular networks based on hydrological similarity. *J. Hydrologic Engineering*, 9:288–302, 2004.
26. Q. Zhu, Y. Tian, and J. Zhao. An efficient depression processing algorithm for hydrologic analysis. *Computers & Geosciences*, 32:615–623, 2006.