Flood-Risk Analysis on Terrains under the Multiflow-Direction Model

Aaron Lowe
Duke University

Pankaj K. Agarwal
Duke University

ABSTRACT

An important problem in terrain analysis is modeling how water flows across a terrain and creates floods by filling up depressions. In this paper we study a number of *flood-risk* related problems: Given a terrain Σ, represented as a triangulated xy-monotone surface with n vertices, a rain distribution $\mathcal{R}$ and a volume of rain ψ, determine which portions of Σ are flooded. We develop efficient algorithms for flood-risk analysis under the multiflow-directions (MFD) model, in which water at a point can flow along multiple downslope edges to more accurately represent flooding events.

We present three main results: First, we present an $O(nm)$-time algorithm to answer a *terrain-flood* query: if it rains a volume ψ according to a rain distribution $\mathcal{R}$, determine what regions of Σ will be flooded; here m is the number of sinks in Σ. Second, we present a $O(n \log n)$-time algorithm for preprocessing Σ into a linear-size data structure for answering *point-flood* queries: given a rain distribution $\mathcal{R}$, a volume of rain ψ falling according to $\mathcal{R}$, and point $q \in \Sigma$, determine whether q will be flooded. A point-flood query can be answered in $O(nk)$ time, where k is the number of maximal depressions in Σ containing the query point q. Alternately, we can preprocess Σ in $O(n \log n + nm)$ time into an $O(nm)$-size data structure so that a point-flood query can be answered in $O(|\mathcal{R}|k+k^2)$ time, where $|\mathcal{R}|$ is the number of vertices in $\mathcal{R}$ with positive rain fall. Finally, we present algorithms for answering a *flood-time query*: given a rain distribution $\mathcal{R}$ and a point $q \in \Sigma$, determine the volume of rain that must fall before q is flooded. Assuming that the product of two $k \times k$ matrices can be computed in $O(k^\omega)$ time, we show that a flood-time query can be answered in $O(nk + k^\omega)$ time. We also give an α-approximation algorithm, for $\alpha > 1$, that runs in $O(nk+k^2(\log(n+\log_\alpha)\rho))$-time, where ρ is a variable on the terrain which depends on the ratio between depression volumes.

We implemented our terrain-flooding algorithm and tested its efficacy and efficiency on real terrains

CCS CONCEPTS

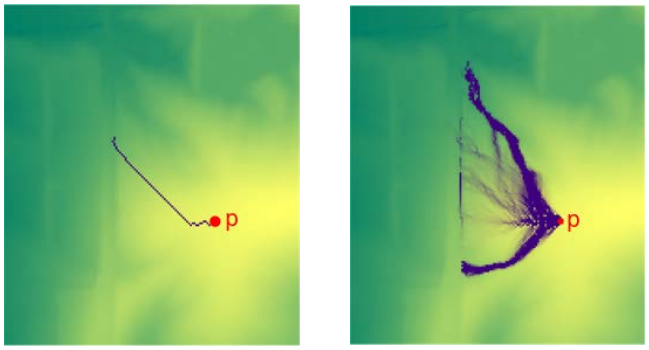
• **Information systems → Geographic information systems**;

KEYWORDS

Terrains, flood-risk analysis, merge trees

SIGSPATIAL '18, November 6–9, 2018, Seattle, WA, USA

ACM ISBN 978-1-4503-5889-7/18/11…$15.00
https://doi.org/10.1145/3274895.3274980



(a) (b)

Figure 1: Rain falls at point p on the terrain, with color indicating relative height, with darker areas being lower. (a) Under SFD it follows a single path, highlighted in blue, until it reaches a sink (b) Under MFD it splits and flows to many sinks, with darker shades of blue denoting more water flowing over that point.

ACM Reference Format:
Aaron Lowe and Pankaj K. Agarwal. 2018. Flood-Risk Analysis on Terrains under the Multiflow-Direction Model. In *26th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (SIGSPATIAL '18), November 6–9, 2018, Seattle, WA, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3274895.3274980

1 INTRODUCTION

An important problem in terrain analysis is modeling how water flows across a terrain and creates floods by filling up depressions. When rains falls, the rate at which a depression fills up depends not only on its shape and the size of its watershed, the area of the terrain that contributes water to the depression, but also on other depressions filling up. Water that falls on the watershed of a full depression will flow to a neighboring depression, effectively making its watershed larger and thus making it fill up faster. Modeling how depressions fill and spill into other depressions during a flash flood event is therefore an important problem.

The prior work on flood-risk analysis assumed that water only flowed along edges of the terrain, and further assumed a *single-flow direction* (SFD) model in which water flows from a point along one downward edge, say, the steepest descent one.[1] In this paper we consider the *multiflow direction* (MFD) model in which water at a point may split and flow to multiple downslope neighbors which models flooding on the terrain more accurately. (Like earlier papers, we also assume that water flows along the edges of the terrain.)

Flood-risk analysis under the MFD models raises many new challenges. Even when rain falls on a single point, as it flows across

[1] Since the modern data sets are high resolution and triangles are small, it is reasonable to assume that water flows along the edges.

Σ, the water can split and reach many sinks, see Figure 1 for an example. In turn, whenever a depression becomes full and starts spilling, this too can split and end up at many sinks across Σ. In contrast, under the SFD model, all water falling at a point flows to a single sink in Σ, and whenever a depression becomes full all the water spilling will again flow to another singular sink. As such the terrain can be partitioned into "watershed" regions. Each region associated with one sink, corresponds to the set of all points on the terrain from where water flows to that sink. Furthermore in the SFD model water flows from a point to a sink only along a single path while in the MFD model, water may reach from a point to the same sink along many paths. As such, flood queries on a terrain under the SFD model are answered by constructing a "flow tree" on the terrain, which is a tree with unweighted edges. In contrast, as we will see below, answering flood queries on a terrain under the MFD model requires constructing a weighted directed (acyclic) graph. Consequently, answering flooding queries under the MFD model is significantly more complicated and the approaches for the SFD model (e.g. as those in [3, 5, 16]) do not extend to the MFD model.

In this paper we study three related problems; for each we assume we are given a terrain Σ, represented as a triangulated xy-monotone surface with n vertices, and a rain distribution $\mathcal{R}$ describing the rate of rain over a region of Σ:

- ***Terrain-flood query:*** given a volume ψ of rain falling according to $\mathcal{R}$, which vertices of Σ will be flooded.
- ***Point-flood query:*** given a volume ψ of rain falling according to $\mathcal{R}$ and a point $q \in \Sigma$, determine whether q is flooded.
- ***Flood-time query:*** given a point $q \in \Sigma$, determine how much water ψ must fall before q is flooded.

Previous work. There has been extensive work in the GIS literature on modeling water flow on a terrain. Many flow modeling approaches first remove all depressions by flooding the terrain, that is, they conceptually pour water onto the terrain until all depressions are filled [2, 11, 14]. However, this approach often leads to unrealistic flow networks, since many important geographical features are removed. Therefore, several methods based on partial flooding algorithms that flood only "small" depressions (based on height, volume or area) have been proposed [1, 4, 7, 8]. Partial flooding methods provide a basis only for approximate solutions to flow modeling, as the underlying assumption is that all "small" depressions are flooded at a certain time, while all "big" depressions are not. To model the flow network at time t accurately, it is necessary to compute all depressions that have been filled by time t and "flood" them.

Quinn et al. [15] proposed a framework for assigning multiple flow directions from each vertex in a terrain Σ, so that the proportion of water which flows from a vertex to a downslope neighbor is proportional to the gradient of the edge connecting the two.

Liu and Snoeyink [13] (see also [5]) proposed an $O(n \log n)$ time algorithm under the SFD model that computes the fill times of all depressions assuming rain is falling at a constant rate on the entire terrain. In reality, localized extreme rainfall can affect downstream areas that do not receive heavy rainfall directly, so it is important to model non-uniform events as well.

Arge et al. [2] describe an I/O efficient algorithm to compute the related *flow routing* and *flow accumulation* problem under both the SFD and MFD models. The flow routing problem asks what direction will water flow from any point on a terrain, and flow accumulation asks if water falls uniformly on the terrain how water flows over a each point in a terrain. They optimized their algorithms to be efficient for extremely large datasets.

Arge et al. [3] described an algorithm under the SFD model to compute the set of flooded vertices when a given volume of rain $\psi \geq 0$ falls on a given region $\mathcal{R}$. The running time of their algorithm is $O(n \log n)$. Their work raises a question whether Σ can be preprocessed in a data structure so that a flooding query can be answered quickly, say, in $O(\log n)$ time. This question was answered positively by Rav et al. [16]. They described data structures to compute flooding queries under the SFD model in $O(|\mathcal{R}| \log n)$ time. They also present an algorithm for answering flood queries that can handle uncertainty in data. Their algorithms, however, do not extend to the MFD model, as they very strongly use the tree structure of the water flow in the SFD model.

Our results. We present three main results:

(i) We present an $O(nm)$-time algorithm to compute a terrain-flood query for a rain distribution $\mathcal{R}$ and volume ψ (Section 4). We do so by maintaining functions describing the rate at which each depression is filling, and processing in time order the *spill events*, when a depression becomes full and starts spilling into an adjacent depression.

(ii) We present an $O(n \log n)$-time algorithm for preprocessing Σ into a linear-size data structure for answering a point-flood query. Given a query point q, a rain distribution $\mathcal{R}$, and volume ψ, the data structure can answer a point-flood query in $O(nk)$ time, where k is the number of maximal depressions that contain the query point q (Section 5). We also present another data structure of size $O(nm)$ that can answer a point-flood query in time $O(|\mathcal{R}|k + k^2)$, where $|\mathcal{R}|$ is the number of vertices with positive rainfall. The data structure can be constructed in $O(n \log n + nm)$ time.

In both versions, we build a directed acyclic graph that describes how *tributaries* of the point q spill into downstream depressions (which we refer to as the *tributary DAG of* q.)

(iii) We present an algorithm for answering a flood-time query. If the product of two $k \times k$ matrices can be computed in $O(k^\omega)$ time, it answers a query in $O(nk + k^\omega)$ (resp. $O(|\mathcal{R}|k + k^\omega + k^2 \log n)$) time using $O(n)$ (resp. $O(nm)$) space. We do so by showing how we can compute the flow spilling from many tributaries to a set of their downstream neighbors at once, utilizing fast matrix multiplication (Section 6).

We also give an α-approximation algorithm, for any $\alpha > 1$ for flood-time query that runs in $O(nk + k^2 \log(n \log_\alpha \rho)$ time, (resp. $O(|\mathcal{R}|k + k^2 \log(n \log_\alpha \rho))$ using $O(n)$ (resp. $O(nm)$) space where ρ is a parameter depending on the terrain and point q relating the ratio between the volumes of depressions on the terrain. It basically performs a binary search and uses the point-flood query data structure at each step.

We have implemented our terrain-flooding algorithms and tested them on real terrains. We show that the algorithm is efficient in practice. We also show how the output differs under the MFD model compared to the SFD model, (Section 7).

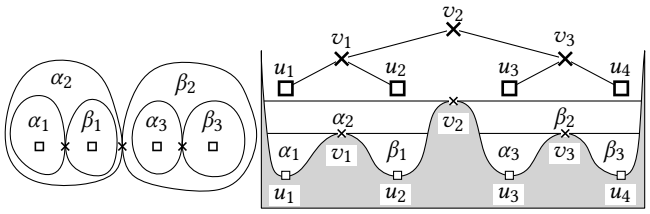


Figure 2: An example terrain with saddle vertices v_1-v_3. Each saddle v_i delimits two maximal depressions α_i and β_i. (a) Terrain seen from above. Sinks are marked with a square and saddles are marked with a cross. (b) Terrain seen from the side,

2 PRELIMINARIES

We begin with a number of preliminary definitions, some of which follow closely Rav et al. [16].

Terrains. Let $\mathbb{M}$ be a triangulation of $\mathbb{R}^2$, and let $\mathbb{V}$ be the set of vertices of $\mathbb{M}$; set $n = |\mathbb{V}|$. We assume that $\mathbb{V}$ contains a vertex v_∞ at infinity, and that each edge $\{u, v_\infty\}$ is a ray emanating from u; the triangles in $\mathbb{M}$ incident to v_∞ are unbounded. Let $h : \mathbb{M} \to \mathbb{R}$ be a height function. We assume that the restriction of h to each triangle of $\mathbb{M}$ is a linear map, that h approaches $+\infty$ at v_∞, and that the heights of all vertices are distinct. Given $\mathbb{M}$ and h, the graph of h, called a *terrain* and denoted by $\Sigma = (\mathbb{M}, h)$, is an xy-monotone triangulated surface whose triangulation is induced by $\mathbb{M}$.

Critical vertices. There is a natural cyclic order on the neighbor vertices of a vertex v of $\mathbb{M}$, and each such vertex is either an *upslope* or *downslope* neighbor. If v has no downslope (resp. upslope) neighbor, then v is a *minimum* (resp. *maximum*). We also refer to a minimum as a *sink*. If v has four neighbors w_1, w_2, w_3, w_4 in clockwise order such that $\max(h(w_1), h(w_3)) < h(v) < \min(h(w_2), h(w_4))$ then v is a *saddle* vertex.

Level sets, contours, depressions. Given $\ell \in \mathbb{R}$, the *ℓ-sublevel set of h* is the set $h_{<\ell} = \{x \in \mathbb{R}^2 \mid h(x) < \ell\}$, and the *$\ell$-level set of h* is the set $h_{=\ell} = \{x \in \mathbb{R}^2 \mid h(x) = \ell\}$. Each connected component of $h_{<\ell}$ is called a *depression*, and each connected component of $h_{=\ell}$ is called a *contour*. Note that a depression is not necessarily simply connected, as a maximum can cause a hole to appear;

For a point $x \in \mathbb{M}$, a depression β_x of $h_{<\ell}$ is said to be *delimited by the point x* if x lies on the boundary of β, which implies that $h(x) = \ell$. A depression β_1 is *maximal* if every depression $\beta_2 \supset \beta_1$ contains strictly more sinks than β_1. A maximal depression that contains exactly one sink is called an *elementary depression*. Note that each maximal depression is delimited by a saddle, and a saddle that delimits more than one maximal depression is called a *negative saddle*. For a maximal depression β, let $\mathrm{Sd}(\beta)$ denote the saddle delimiting β, and let $\mathrm{Sk}(\beta)$ denote the set of sinks in β. The *volume* of a depression β of $h_{<\ell}$ is

$$\mathrm{Vol}_h(\beta) = \int_\beta (\ell - h(v)) dv. \tag{1}$$

Merge tree. The maximal depressions of a terrain form a hierarchy that is easily represented using a rooted tree, called the *merge tree* [6, 12] and denoted by T_h. Suppose we sweep a horizontal plane from $-\infty$ to ∞. As we vary ℓ, the depressions in $h_{<\ell}$ vary continuously, but their structure changes only at sinks and negative saddles. If we increase ℓ, then a new depression appears at a sink, and two depressions merge at a negative saddle. The merge tree is a tree that tracks these changes. Its leaves are the sinks of the terrain, and its internal nodes are the negative saddles. The edges of T_h, the merge tree, are in one-to-one correspondence with the maximal depressions of Σ_h, that is, we associate each edge (u, v) with the maximal depression delimited by u and containing v. See figure 2 for an example. We assume that T_h has an edge from the root of T_h extending to $+\infty$, corresponding to the depression that extends to ∞. For simplicity we assume that T_h is binary, that is, each negative saddle delimits exactly two depressions. Non-simple saddles can be unfolded into a number of simple saddles [9].

Let u be a negative saddle, let (u, v_1) and (u, v_2) be two down edges in T_h from u, and let (w, u) be the up edge from u. We call the depression associated with (u, v_2) (resp. with (w, u)) as the *sibling* (resp. *parent*) (depression) of that associated with (u, v_1).

Any given point $q \in \mathbb{M}$ is contained in a sequence of maximal depressions $\alpha_1 \supset \cdots \supset \alpha_k \ni q$, each α_i delimited by a saddle v_i with sibling depression β_i. Note that these saddles form a path in T_h from q to the root. We refer to the maximal depressions $\beta_1, \ldots, \beta_k$ as the *tributaries of q* and denote them by $\mathcal{T}_q$. For a depression β delimited by a point q, we will also use $\mathcal{T}_\beta$ to denote $\mathcal{T}_q$.

Van Kreveld et al. [12] gave an $O(n \log n)$-time algorithm for constructing the merge tree. The algorithm was later extended to 3D by Tarasov and Vyalyi [17], and to arbitrary dimensions by Carr et al. [6].

T_h can be preprocessed in $O(n)$ additional time so that for a point $x \in \mathbb{R}^2$, $\mathrm{Vol}(\beta_x)$, the volume of the depression delimited by x can be computed in $O(\log n)$ time. In the following sections, we will be working with a fixed height function, so will drop the subscript h from T_h, Vol_h, etc.

3 FLOODING MODEL

Flow graph and flow functions. We transform $\mathbb{M}$ into a directed acyclic graph $\mathcal{M}$, referred to as the *flow graph*, by directing each edge $\{u, v\}$ of $\mathbb{M}$ from u to v if $h(u) > h(v)$, and from v to u otherwise, i.e., each edge is directed in the downward direction. For each (directed) edge (u, v), we define the local flow $\lambda(u, v)$ to be the portion of water arriving at u that flows along the edge (u, v) to v. If $\lambda(u, v) = 0$, we can delete the edge (u, v) from $\mathcal{M}$.

The value of $\lambda(u, v)$ is, in general, based on relative heights of the downslope neighbors of u. Note that the SFD model has $\lambda(u, v) > 0$ for exactly one downslope neighbor of u. We will not focus on the specifics of the flow function used, and only assume that it is specified and for a pair u, v can be retrieved in $O(1)$ time.

Following the edges of $\mathcal{M}$, water reaches a set of sinks of $\mathbb{M}$. We define a *flow function* $\phi : \mathbb{V}^2 \to [0, 1]$, which specifies the proportion of water that flows from a vertex u to another vertex v. Recall that unlike the SFD model, water can flow from u to v along many paths. The flow function is defined recursively as follows:

$$\phi(u,v) = \begin{cases} 1 & \text{if } u = v, \\ \sum_{(u,w)\in\mathbb{E}(\mathcal{M})} \lambda(u,w)\cdot\phi(w,v) & \text{otherwise.} \end{cases} \quad (2)$$

For a maximal depression β, we define

$$\phi(u,\beta) = \sum_{s\in \mathrm{Sk}(\beta)} \phi(u,s)$$

to be the portion of water that reaches from a vertex u to β. Recall that $\mathrm{Sk}(\beta)$ is the set of sinks in β.

Rain distribution. We let $\mathcal{R}$ denote a *rain distribution*, which is specified as a probability distribution on the vertices of the terrain, that is, for each vertex $v \in \mathbb{V}$, $\mathcal{R}(v) \geq 0$ indicates the rate at which it rains on v, and we require that $\sum_v \mathcal{R}(v) = 1$. We denote by $|\mathcal{R}|$ the number of vertices with positive rainfall in $\mathcal{R}$, and we assume that $\mathcal{R}$ is represented as a list of $|\mathcal{R}|$ pairs $(v, \mathcal{R}(v))$. In practice $|\mathcal{R}| \ll n$.

Flooding model. Our flooding model follows a similar depression filling model as Liu and Snoeyink [13]. When rain falls according to a distribution $\mathcal{R}$ on Σ, water flows according to the flow function and accumulates in depressions of Σ. When a maximal depression β_i fills up, water spills from the simple saddle v_i delimiting β_i towards sinks in the adjacent depression delimited by the saddle. We refer to this event as a *spill event*.

The above process defines a sequence of spill events, each event marking a sink u as full, and redistributing the rain falling on u to other sinks. In our model, the maximal depressions of Σ fill up at a constant rate between any two consecutive spill events. That is, after a spill event occurs at time t_1 and until the next occurs at time t_2, the volume of water in each elementary depression is a non-decreasing linear function of time.

For a maximal depression β, we define the *fill rate* $F_\beta : \mathbb{R}_{\geq 0} \to [0,1]$ as the rate at which water is arriving in the depression β. That is, the rate at which rain is falling directly in β plus the rate at which other depressions are spilling water into β. The fill rate F_β is a monotonically nondecreasing piecewise-constant function. Similarly we define the *spill rate* $S_\beta : \mathbb{R}_{\geq 0} \to [0,1]$ as the rate at which water spills from β through the saddle that delimits β. Let τ_β, called the *fill time*, be the time at which β becomes full. Let β' be the sibling depression of β. Then

$$S_\beta(t) = \begin{cases} 0 & \text{if } t < \tau_\beta \vee \tau_{\beta'} \leq \tau_\beta, \\ F_\beta(t) & \text{if } t > \tau_\beta \wedge \tau_{\beta'} > \tau_\beta. \end{cases}$$

By the definition of the fill rate, for any maximal depression β, its initial fill rate is

$$F_\beta(0) = \sum_{s\in \mathrm{Sk}(\beta)} \sum_v R(v)\phi(v,s), \quad (3)$$

which is how much rain water flows intially to the sinks of β. We can define $F_\beta(0)$ recursively using (2) as follows. Abusing the notation a little, let $F_v(0)$ denote the water reaching v at time 0. Then

$$F_v(0) = R(v) + \sum_{(w,v)\in\mathbb{E}(\mathcal{M})} F_w(0)\lambda(w,v) \quad (4)$$

$$F_\beta(0) = \sum_{s\in \mathrm{Sk}(\beta)} F_s(0). \quad (5)$$

For any $t \geq 0$

$$F_\beta(t) = F_\beta(0) + \sum_{\alpha\in\mathcal{T}_\beta} S_\alpha(t)\phi(\mathrm{Sd}(\alpha),\beta). \quad (6)$$

That is, the fill rate of β at time t is the direct rain reaching β plus the water spilling into β from its tributaries that are full.

4 TERRAIN-FLOOD QUERIES

In this section we describe an algorithm for answering a terrain-flood query. That is, given a rain distribution $\mathcal{R}$ and a volume ψ, determine which vertices of $\mathbb{M}$ will be flooded if a volume of ψ rain falls according to the distribution $\mathcal{R}$. Recall that $\sum_{v\in\mathbb{M}} \mathcal{R}(v) = 1$, so ψ is the same as saying that rain falls for ψ units of time on $\mathbb{M}$ with distribution $\mathcal{R}$.

Roughly speaking, for every maximal depression β, we compute its fill time τ_β as well as its fill rate $F_\beta(t)$ for $0 \leq t \leq \min\{\tau_\beta, \psi\}$. If a depression β is full, then all its vertices are flooded. For a partially filled depression β, its fill rate function $F_\beta(t)$, allows us to compute the water level in β at time ψ, from which we can determine which vertices in β are flooded.

We now describe the algorithm for computing τ_β and F_β for each maximal depression β. For a non-elementary depression β with its two children depressions β_1 and β_2, both β_1 and β_2 become full before β becomes full, so we track depressions β for when it gets full only after both β_1 and β_2 are full. Moreover,

$$F_\beta(t) = F_{\beta_1}(t) + F_{\beta_2}(t),$$

if neither β_1 nor β_2 is full. If one of them, say β_2 is full then $F_\beta = F_{\beta_1}(t)$. We can therefore compute the fill rate of β from those of β_1 and β_2.

We call a maximal depression *active* if it is not full, and furthermore,if it is a non-elementary depression, then both its children are full. Our algorithm maintains the set of active depressions and their estimated fill time based on the current fill rate. We observe that the next depression to become full (i.e. the next spill event) will be one of the active depressions, and the fill rate of all depressions will remain the same until the next spill event. Therefore the estimated fill time corresponding to the next spill event is the actual fill time.

Let $\mathcal{D}$ denote the set of active depressions. Abusing the notation a little, let τ_β denote the estimated fill time of a depression $\beta \in \mathcal{D}$. We store these fill times in a priority queue Q. At each step, the algorithm removes the next fill time, say τ_β, from Q. Then we mark β full at time τ_β and remove it from $\mathcal{D}$. If the sibling depression β' of β is not full, as in Figure 3b, then water starts spilling into various sinks of β' and we update the fill rates of active depressions that are descendants of β'. We also update the fill times of all active depressions whose fill rates have changed. On the other hand, if β' is full, as in Figure 3a, then we make the parent depression of β active, add it to $\mathcal{D}$, and set its current fill rate to be $F_\beta(\tau_\beta)$. We estimate the fill time of the parent depression. The algorithm stops when the time of the next event exceeds ψ. (Recall we assume $\mathbb{M}$ has a vertex with height ∞, so Q never becomes empty.) We now describe how to implement each step of the algorithm efficiently.

First, we can keep track of active depressions using the merge tree T. Namely, we delete all edges of T that correspond to full depressions. The non-full depressions form a top subtree of T. The edges connected to the leaves of the pruned merge tree correspond

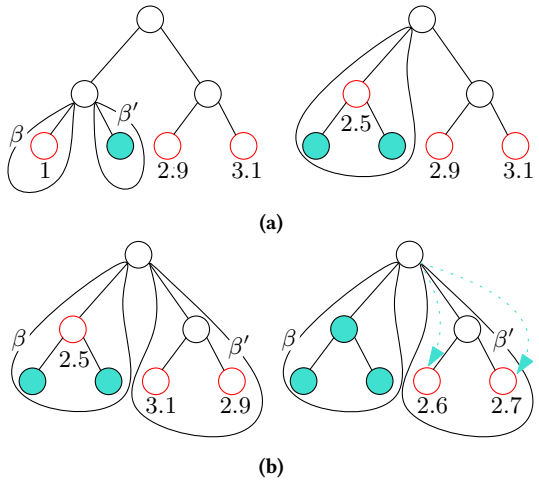


Figure 3: Two examples of before and after a spill event, where β is the next depression to fill. Active depressions are highlighted red and full depressions are shaded blue. Each active depression is labeled with its estimated fill times.

to active depressions. If a depression β becomes full at an event, then we remove the edge, say (u, v), corresponding to β from T. If the deletion of (u, v) creates a new leaf in T, then we have a new active depression and we add it to $\mathcal{D}$.

Next, we can update the fill rates of active depressions, as follows. Suppose β becomes full but its sibling depression β' is not full. Then we update the fill rates of active depressions that are descendants of β' in T, using (6). Note that it also requires computing spill rates of the depressions in the subtree of T rooted at β' that are full but whose siblings are not full. Computing fill rates using (6) is however expensive, so we describe a more efficient method that updates the flow graph dynamically.

We first describe how to compute the initial fill rates using the flow graph $\mathcal{M}$ and (4). We note that the initial set of active depressions will be elementary depressions, each containing a single sink. To find the initial fill rates, we compute $F_v(0)$ for all vertices v in descending height order. We note that $F_v(0)$ depends only on $\mathcal{R}$, and directed edges (w, v), which satisfy $h(w) > h(v)$ by construction. So for each vertex v, we can compute $F_v(0)$ in time proportional to the in-degree of v, plus a constant. If v is a sink with $F_v(0) > 0$, we add the elementary depression containing v to our set of active depressions. Letting β_v be the elementary depression containing v, we set the fill-rate of β_v to $F_v(0)$, and set the fill time to be $\mathrm{Vol}(\beta_v)/F_v(0)$, and insert β_v into the priority queue. The total time to find the initial active depressions and their fill rates is $O(n + |\mathbb{E}(\mathcal{M})|) = O(n)$ because $\mathbb{M}$ is a planar graph. Constructing the priority queue takes an additional $O(m \log m)$ time.

Next, we describe the general case, letting β be the next depression to become full. See Figure 4 for an overview of the process. Suppose that β is the i^{th} depression to become full, and it becomes full at time τ_i. Set $\mathcal{M}^0 = \mathcal{M}$. At each iteration when an active depression becomes full, we update the prior flow graph $\mathcal{M}^{i-1}$ to $\mathcal{M}^i$. To do so, we contract all vertices $z \in \beta$, with the exception of $\mathrm{Sd}(\beta)$, to a single super-node v_β. For each edge $(u, z) \in \mathbb{E}(\mathcal{M}^{i-1})$, with

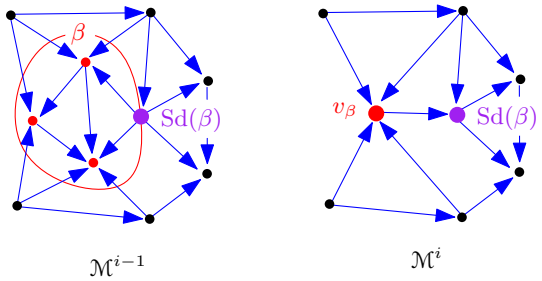


Figure 4: Left: $\mathcal{M}^{i-1}$ before the depression β is contracted; Right: $\mathcal{M}^i$ after the three vertices in β are contracted to v_β.

$u \notin \beta$, we add the edge (u, v_β) and update the weight as follows:

$$w(u, v_\beta) = \sum_{z \in \beta} w(u, z).$$

We also add the edge $(v_\beta, \mathrm{Sd}(\beta))$ with weight 1. Finally, some edges from $\mathrm{Sd}(\beta)$ may have been removed, so if any edges remain (i.e. the case when the neighbor of β, β' is not full), we rescale the weights of the remaining edges so that the out-degree of $\mathrm{Sd}(\beta)$ is 1. This contraction takes $O(n)$ time. Then we can recompute the fill rates of the depressions using the same method we used to compute the initial fill-rates, using $\mathcal{M}^i$ in place of $\mathcal{M}$.

Finally, we describe how to compute the new fill time of depressions whose fill rates have changed. For each active depression β, we maintain the volume of rain in β at time τ_{i-1}, denoted as $\mathcal{F}_\beta(\tau_{i-1})$. Then at step i, we set

$$\mathcal{F}_\beta(\tau_i) = \mathcal{F}_\beta(\tau_{i-1}) + (\tau_i - \tau_{i-1}) F_\beta(\tau_{i-1})$$

and compute the new estimated fill time to be

$$\frac{\mathrm{Vol}(\beta) - \mathcal{F}_\beta(\tau_i)}{F_\beta(\tau_i)}.$$

By maintaining the priority queue as a Fibonacci heap, we can update all the fill times in $O(m)$ time.

Putting everything together, we obtain the following:

Lemma 4.1. Given a triangulation $\mathbb{M}$ of $\mathbb{R}^2$ with n vertices, a height function $h : \mathbb{M} \to \mathbb{R}$ that is linear on each face of $\mathbb{M}$, and a rain distribution $\mathcal{R}$, the fill-times of all maximal depressions of $\Sigma = (\mathbb{M}, h)$ can be computed in $O(nm)$ time, where m is the number of sinks in Σ.

The procedure described above for computing the fill time of an active depression can be used to compute the water level in each active depression at time ψ. These water levels define a set of contours in $\mathbb{M}$, and these contours separate the filled portions of $\mathbb{M}$ from the unfilled portion. By scanning $\mathbb{M}$ once, we can compute all vertices that are flooded at time ψ. Hence we obtain the following:

Theorem 4.2. Given a triangulation $\mathbb{M}$ of $\mathbb{R}^2$ with n vertices, a height function $h : \mathbb{M} \to \mathbb{R}$ that is linear on each face of $\mathbb{M}$, and a rain distribution $\mathcal{R}$, the terrain-flood query which returns the vertices of $\mathbb{M}$ that are flooded, can be answered in $O(nm)$ time, where m is the number of sinks in $(\mathbb{M}, h)$.

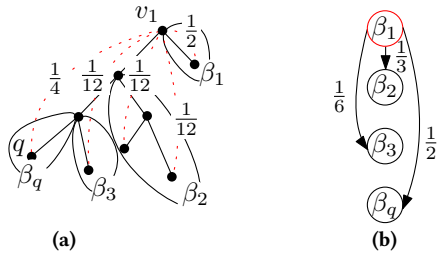


Figure 5: (a) A merge tree T, with tributaries $(\beta_1, \beta_2, \beta_3)$ of q marked and flow $\phi(v_1, s)$ to each sink s marked with dashed lines. (b) A subset of G[q], with the edge weights from β_1 marked.

5 POINT-FLOOD QUERY

Given a rain distribution $\mathcal{R}$, a query point $q \in \mathbb{M}$, and a rain volume ψ, the point-flood query asks whether the point q is flooded if a volume ψ rain falls with distribution $\mathcal{R}$. Of course, the terrain-flood query procedure described in Section 4 can answer this query, but our goal is to answer this query faster. We exploit two observations: first, we need not compute the fill rate and fill time of all maximal depressions. In particular, suppose q lies in a maximal depression β and there are two children depressions β_1 and β_2 of the sibling depression β' of β. Then we need not compute the fill times of β_1 and β_2. It suffices to compute when β' fills and at what rate water spills from β' to the depression β_q. In fact, it suffices to consider the tributaries of q, defined in Section 2. Second, we do not have to compute the fill rates of the tributaries of β_q for all values time. Instead it suffices to determine whether the volume of water in the depression β_q is less than Vol(β_q).

We first define the notion of tributary graph that describes where the water spills when a tributary of q fills. Next, we describe the functions that estimate how much water has arrived in a tributary of q and how much it has spilled. Roughly speaking, we compute these estimates assuming that a tributary of q fills before its sibling (which is a maximal depression containing q) fills. We prove that our estimates answer the point-flood query correctly.

We describe two data structures to compute these estimates. The first one is more space efficient — it uses $O(n)$ space and answers a query in $O(nk)$ time, where k is the number of maximal depressions containing q. The second one has a faster query time — it uses $O(mn)$ space and answers a query in $O(|\mathcal{R}|k + k^2)$ time.

Tributary graph. For a point $q \in \mathbb{M}$, the tributary graph G[q] = $(\mathsf{X}_q, \mathsf{E}_q)$ is a directed acyclic graph. X_q is the depression β_q plus its tributaries, i.e., $\mathsf{X}_q = \mathcal{T}_q \cup \{\beta_q\}$. For a pair of depressions $\alpha, \beta \in \mathsf{X}$, we add the directed edge (α, β) to E_q if $h(\mathrm{Sd}(\alpha)) > h(\mathrm{Sd}(\beta))$ (if $\beta = \beta_q$ then by Sd(β_q) we mean q) and $\phi(\mathrm{Sd}(\alpha), \beta) > 0$. We set the weights of the edge (α, β) to be

$$w(\alpha, \beta) = \frac{\phi(\mathrm{Sd}(\alpha), \beta)}{\sum_{(\alpha,\gamma)\in\mathsf{E}_q} \phi(\mathrm{Sd}(\alpha), \gamma)}, \tag{7}$$

i.e. the weights are normalized so that the weighted out-degree of each node in G[q] is 1. See Figure 5 for an example.

Fill and spill volumes. For a depression β, we define $\mathcal{F}_\beta, \mathcal{S}_\beta : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ as *fill- and spill-volume* functions of β, i.e. $\mathcal{F}_\beta(t)$ tells how much water has arrived in depression β by time t, and $\mathcal{S}_\beta(t)$ tells how much water has spilled from β by time t.

$$\mathcal{F}_\beta(t) = \int_0^t F_\beta(x)dx \quad \text{and} \quad \mathcal{S}_\beta(t) = \int_0^t S_\beta(x)dx.$$

By definition of the spill rate,

$$\mathcal{S}_\beta(t) = \max\{0, \mathcal{F}_\beta(t) - \mathrm{Vol}(\beta)\}.$$

It is expensive to compute $\mathcal{F}_\beta$ and $\mathcal{S}_\beta$ exactly for each tributary of q, so we define slightly different functions $\tilde{\mathcal{F}}_\beta, \tilde{\mathcal{S}}_\beta : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ for all $\beta \in \mathsf{X}_q$. They are fill, spill volume functions under the assumption that every tributary of β_q fills before its sibling, that is, water spills from a tributary β to various sinks in the sibling β' of β; note that β' is a depression containing q.

We define $\tilde{\mathcal{F}}_\beta, \tilde{\mathcal{S}}_\beta$ recursively using the tributary graph G[q] as follows:

$$\tilde{\mathcal{F}}_\beta(t) = F_\beta(0)t + \sum_{(\alpha,\beta)\in\mathsf{E}_q} \tilde{\mathcal{S}}_\alpha(t)w(\alpha, \beta),$$

$$\tilde{\mathcal{S}}_\beta(t) = \max\{0, \tilde{\mathcal{F}}_\beta(t) - \mathrm{Vol}(\beta)\}.$$

The following lemma is the key observation.

Lemma 5.1. Let $\tau_q := \tau_{\beta_q}$ be the fill time of the depression β_q. For any $\beta \in \mathsf{X}_q$,

(i) $\tilde{\mathcal{F}}_\beta, \tilde{\mathcal{S}}_\beta$ are monotonically non-decreasing functions.

(ii) For $t \leq \tau_q$, $\tilde{\mathcal{F}}_\beta(t) = \mathcal{F}_\beta(t)$ and $\tilde{\mathcal{S}}_\beta(t) = \mathcal{S}_\beta(t)$.

Proof. We begin by noting that if the lemma holds for $\tilde{\mathcal{F}}_\beta$, for all $\beta \in \mathsf{X}_q$, it also holds for $\tilde{\mathcal{S}}_\beta$, so we prove the lemma for $\tilde{\mathcal{F}}_\beta$.

(i) We prove the claim by induction on the topological ordering of X_q. For our base case, let α be the source of the tributary DAG, we have $\tilde{\mathcal{F}}_\alpha(t) = F_\alpha(0)t$, which is monotonically nondecreasing. By induction hypothesis, assume that $\tilde{\mathcal{F}}_\alpha(t)$ is monotonically nondecreasing, for all α appearing before β in the topological ordering of X_q. Then $\tilde{\mathcal{F}}_\beta$ is the sum of monotonically nondecreasing functions, so it also is monotonically nondecreasing.

(ii) By the definition of τ_q, the fill time of all maximal depressions containing q is greater than q. Therefore none of the sibling depressions of a tributary β of q (each of which is a maximal depression containing q) spills water into β, and therefore by definitions of $\mathcal{F}_\beta, \tilde{\mathcal{F}}_\beta, \mathcal{S}_\beta$, and $\tilde{\mathcal{S}}_\beta$, $\mathcal{F}_\beta(t) = \tilde{\mathcal{F}}_\beta(t)$ for all $t \leq \tau_q$. □

Corollary 5.2. $\tilde{\mathcal{F}}_{\beta_q}(t) < \mathrm{Vol}(\beta_q)$ if and only if $t < \tau_q$.

Proof. If $\psi < \tau_q$, then $\tilde{\mathcal{F}}_{\beta_q}(t) = \mathcal{F}_{\beta_q}(t) < \mathrm{Vol}(\beta_q)$. The inequalities follows by the definition of τ_q. On the other hand, if $t \geq \tau_q$, then by Lemma 5.1 (i) & (ii)

$$\tilde{\mathcal{F}}_{\beta_q}(t) \geq \tilde{\mathcal{F}}_{\beta_q}(\tau_q) = \mathcal{F}_{\beta_q}(\tau_q) = \mathrm{Vol}(\beta_q).$$

□

By Corollary 5.2, given ψ, we compute $\tilde{\mathcal{F}}_{\beta_q}(\psi)$ and return yes if this quantity is at least Vol(β_q). We now describe the two algorithms for computing $\tilde{\mathcal{F}}_{\beta_q}(\psi)$.

A space-efficient algorithm. In the preprocessing step, we construct the merge tree T and preprocess it so that for a point $q \in \mathbb{M}$, (i) the edge of T containing q and (ii) $\mathrm{Vol}(\beta_q)$ can be computed in $O(\log n)$ time.

This step takes $O(n \log n)$ time, and the size of the data structure is $O(n)$.

For a query rain distribution $\mathcal{R}$ and a query point, we first find the edge e of T containing q and $\mathrm{Vol}(\beta_q)$. Given e, we compute the set $\mathcal{T}_q$ of tributaries of q in $O(k)$ time by traversing T upward from e. We now construct the tributary graph $\mathsf{G}[q] = (\mathsf{X}_q, \mathsf{E}_q)$. For each pair of tributaries $\alpha, \beta \in \mathsf{X}_q := \mathcal{T}_q \cup \{\beta_q\}$, with $h(\mathrm{Sd}(\alpha)) > h(\mathrm{Sd}(\beta))$, we compute $\phi(\mathrm{Sd}(\alpha), \beta)$ as follows: We contract all vertices in each depression $\beta \in \mathsf{X}_q$ into a single vertex called a super node (recall that the saddle delimiting β is not a vertex of β_q). If there are two parallel edges e, e' from a vertex to a super node, we merge them into a single edge with local flow of the merged edge being $\lambda(e) + \lambda(e')$. For each saddle $\alpha \in \mathsf{X}_q$, we can now compute $\phi(\alpha, \beta)$ for all $(\alpha, \beta) \in \mathsf{E}_q$ in $O(n)$ time, using (6.) Hence, the total time spent in computing the weights of edges of E_q is $O(nk)$.

To compute $\tilde{\mathcal{F}}_q(\psi)$ for all depressions $\beta \in \mathsf{X}_q$, we first compute $F_\beta(0)$, for all $\beta \in \mathsf{X}_q$, in $O(n)$ time as described in Section 4. Next, we compute $\tilde{\mathcal{F}}_\beta(\psi)$ for every $\beta \in \mathsf{X}_q$, using the recurrence in a total of $|\mathsf{E}_q| = O(k^2)$ time.

Hence, the total time spent in answering a point-flood query is $O(nk)$.

A faster algorithm. We note that the two expensive steps of in the above algorithm are: (i) computing the weights of edges of E_q, and (ii) $F_\beta(0)$, the initial fill rate, for all depressions. We expedite these steps by storing more information at the nodes of T. In particular, for each vertex $v \in \mathbb{M}$ and for each maximal depression β, we store the value of $\phi(v, \beta)$. Recall that there are $O(m)$ maximal depressions, so we need $O(mn)$ storage. Actually we need to store only non-zero values, in practice, the number of such pairs is much smaller than mn. The total time spent in computing this additional information is $O(nm)$, by computing it for each sink in $O(n)$ time.

Given $\phi(v, \beta)$ for all pairs, we can compute the weights of all edgers of $\mathsf{G}[q]$ in $O(k^2)$ time using (7). Next, for each depression $\beta \in \mathsf{X}_q$, $F_\beta(0)$ can be computed in $O(|\mathcal{R}|)$ time using the formula

$$F_\beta(0) = \sum_{u : \mathcal{R}(u) > 0} \mathcal{R}(u)\phi(u, \beta).$$

Hence, a point-flood query can be answered in $O(|\mathcal{R}|k + k^2)$ time. Putting everything together, we obtain the following:

Theorem 5.3. Given a triangulation of $\mathbb{M}$ of $\mathbb{R}^2$ with n vertices, a height function $h : \mathbb{M} \to \mathbb{R}$ which is linear on each face of $\mathbb{M}$, a data structure of size $O(n)$ can be constructed in time $O(n \log n)$ so that a point-flood query can be answered in $O(nk)$ time, where k is the number of maximal depressions containing the query point. Alternately, a data structure of size $O(mn)$ can be constructed in $O(n \log n + mn)$ time so that a point-flood query can be answered in $O(|\mathcal{R}|k + k^2)$ time, where $|\mathcal{R}|$ is the complexity of the query rain distribution, and m is the number of sinks in the terrain $(\mathbb{M}, h)$.

6 FLOOD-TIME QUERIES

In this section we describe algorithms for answering flood-time queries: given a rain distribution $\mathcal{R}$ and a point $q \in \mathbb{M}$, how long rain must fall before q becomes flooded. We first describe an exact algorithm for answering a flood-time query and then present an approximation algorithm. For both the exact and approximation algorithms, we assume that given the query point q, we have computed the (directed) tributary graph $\mathsf{G}[q] = (\mathsf{X}_q, \mathsf{E}_q)$, as described in Section 5.

Exact flood-time query. Roughly speaking, we wish to compute the fill rate F_{β_q} of the depression β_q, which in turn requires computing the fill and spill rates of all tributaries of q. But computing them is expensive, so we use the same observation as in Section 5, namely, we compute "conditional" fill and spill rates of each $\beta \in \mathsf{X}_q$ under the assumption that every tributary of β_q fills before its sibling. Abusing notation slightly, we use F_β, S_β to denote these conditional fill and spill rates of β for all $\beta \in \mathsf{X}_q$. By Lemma 5.1 (ii), F_β, S_β are the exact fill rates for all $t \le \tau_q$, so they determine the value of τ_q exactly. By definition,

$$F_\beta = F_\beta(0) + \sum_{(\alpha,\beta)\in\mathsf{E}_q} S_\alpha(t) w(\alpha, \beta), \quad S_\beta = \begin{cases} 0 & \text{if } t < \tau_\beta, \\ F_\beta(t) & \text{if } t \ge \tau_\beta. \end{cases}$$

Both F_β and S_β are piecewise-constant non-decreasing functions, and their values change only at fill times of tributaries of β_q. Using this observation, we can rewrite F_β, S_β as follows: Let $\mathsf{E}_q = \langle \beta_1, \beta_2, \cdots, \beta_{k+1} = \beta_q \rangle$ be the sequence of depressions sorted in descending order of their heights. For $1 \le i \le k+1$, we use F_i, S_i, τ_i to denote $F_{\beta_i}, S_{\beta_i}, \tau_{\beta_i}$ respectively, and for $j \le i$, we set $w_{j,i} = w(\beta_j, \beta_i)$. Recall that the fill rate of β_i increases only when a β_j, for $j < i$, (i.e. β_j is higher than β_i) fills and spills water into β_i. For a pair i, j let $f_{i,j}$ (resp. $s_{i,j}$) denote the increase in the fill rate F_i (resp. spill rate S_i) at time τ_j. Note that $f_{i,j}, s_{i,j} = 0$ for $j \ge i$. Set $f_{i,0} = F_i(0)$ and $\tau_0 = 0$. Then

$$F_i(t) = \sum_{0 \le j \le i, \tau_j \le t} f_{i,j} \quad \text{and} \quad S_i(t) = \sum_{1 \le j \le i, \tau_j \le t} s_{i,j}. \tag{8}$$

Furthermore

$$f_{i,j} = \sum_{j \le \ell \le i} w_{\ell,i} s_{\ell,j} \quad \text{for } 1 \le j < i. \tag{9}$$

That is, the increase in fill rate of F_ℓ is the increase in the spill rate of β_ℓ for $\ell < i$, at time τ_j times the proportion of water that spills from β_ℓ into β_i summed over all tributaries higher than β_i. Note that the fill rate of β_ℓ does not increase at τ_j for $\ell < j$. Similarly, $s_{i,j}$ can be written as follows:

$$s_{i,j} = \begin{cases} 0 & \text{if } j > i \vee \tau_j < \tau_i, \\ \sum_{\ell < i, \tau_\ell < \tau_i} f_{i,j} & \text{if } j = i, \\ f_{i,j} & \text{if } j < i \wedge \tau_j \ge \tau_i. \end{cases} \tag{10}$$

Let $\boldsymbol{F} = (f_{i,j})_{\substack{1 \le i \le k+1 \\ 0 \le j \le k+1}}$ and $\boldsymbol{S} = (s_{i,j})_{1 \le i,j \le k+1}$ be *fill* and *spill* matrices; both of them are lower triangular matrices.

Let $\boldsymbol{F}_i$ (resp. $\boldsymbol{S}_i$), for $1 \le i \le k+1$, denote the i^{th} row of $\boldsymbol{F}$ (resp. $\boldsymbol{S}$). For a pair $1 \le a, b \le k+1$, let $\boldsymbol{F}_{a:b}$ (resp. $\boldsymbol{S}_{a:b}$) denote the submatrix of $\boldsymbol{F}$ (resp. $\boldsymbol{S}$) consisting of rows $a, a+1, \cdots, a+b-1$; $\boldsymbol{F}_a = \boldsymbol{F}_{a:1}$ and $\boldsymbol{S}_a = \boldsymbol{S}_{a:1}$.

After having computed $\boldsymbol{F}_i$, $\boldsymbol{S}_i$ can be computed in $O(i)$ time using (10). We refer to this procedure as Spill_Rate$(i, \boldsymbol{F}_i)$. Here we assume that we do not explicitly set $k-i+1$ entries of $\boldsymbol{S}_i$ as 0.

We now describe a recursive procedure for computing $\boldsymbol{F}$ and $\boldsymbol{S}$, outlined in Figure 1. It takes as input two values $1 \le a, b \le k+1$ and a $b \times (k+1)$ lower-triangular matrix $\boldsymbol{F}^{<}_{a:b} = (f^a_{ij})_{\substack{a \le i < a+b \\ 1 \le j < i}}$, where

$$f^a_{i,j} = \sum_{j \le \ell < a} w_{\ell,i} s_{\ell,j}$$

is a partial prefix sum of $f_{i,j}$ and the procedure returns $\boldsymbol{F}_{a:b}$ and $\boldsymbol{S}_{a:b}$. We note that for $b = 1$, $f_{a,j} = f^a_{a,j}$ for all $1 \le j < a$. Hence, for $b = 1$, given $\boldsymbol{F}^{<}_{a:1}$, $\boldsymbol{F}_a$ can be returns in $O(a)$ time by simply setting $f_{a,0} = F_a(0)$ and $f_{a,j} = f^a_{a,j}$ for $1 \le j < a$. We call this procedure Fill_Rate$(a, \boldsymbol{F}^{<}_{a:1})$.

Algorithm 1 Flood_Time$(a, b, \boldsymbol{F}^{<}_{a:b})$

if $b = 1$ **then**
  $\boldsymbol{F}_a$ = Fill_Rate$(a, \boldsymbol{F}^{<}_{a:1})$
  $\boldsymbol{S}_a$ = Spill_Rate$(\boldsymbol{F}_a)$
  return $(\boldsymbol{F}_a, \boldsymbol{S}_a)$
end if
$\boldsymbol{F}_{a:b/2}, \boldsymbol{S}_{a:b/2}$ = Flood_Time $\left(a, b/2, \boldsymbol{F}^{<}_{a:b/2}\right)$
$\boldsymbol{A} = (f^a_{i,j})_{\substack{a+b/2 \le i < a+b \\ 1 \le j < i}}$
$\boldsymbol{B} = \left(\sum_{\ell=a}^{a+\frac{b}{2}-1} w_{\ell,i} s_{\ell,j}\right)_{\substack{a+b/2 \le i < a+b \\ 1 \le j < i}}$
$\boldsymbol{F}^{<}_{a+b/2:b/2} = \boldsymbol{A} + \boldsymbol{B}$
$\boldsymbol{F}_{a+b/2:b/2}, \boldsymbol{S}_{a+b/2:b/2}$ = Flood_Time $\left(a+b/2, b/2, \boldsymbol{F}^{<}_{a+b/2:b/2}\right)$
$\boldsymbol{F}_{a:b} = \begin{pmatrix} \boldsymbol{F}_{a:b/2} \\ \boldsymbol{F}_{a+b/2:b/2} \end{pmatrix}$, $\boldsymbol{S}_{a:b} = \begin{pmatrix} \boldsymbol{S}_{a:b/2} \\ \boldsymbol{S}_{a+b/2:b/2} \end{pmatrix}$
return $(\boldsymbol{F}_{a:b}, \boldsymbol{S}_{a:b})$

The only nontrivial step in Flood_Time is the computation of matrix $\boldsymbol{B}$. A straightforward algorithm for computing $\boldsymbol{B}$ takes $O(b^3)$ time, but we can do better using a faster matrix multiplication algorithm, as follows.

Let

$$\boldsymbol{W} = \left(w_{\ell,i}\right)_{\substack{a \le \ell < a+b/2 \\ a+b/2 \le i < a+b}} \text{ and } \quad \boldsymbol{X} = \left(s_{\ell,j}\right)_{\substack{a \le \ell < a+b/2 \\ 1 \le j < a+b}}$$

Then $\boldsymbol{B} = \boldsymbol{W}^T \boldsymbol{X}$. Since $\boldsymbol{X}$ is not a square matrix, we partition $\boldsymbol{X}$ into $r = \lceil \frac{2a}{b} \rceil$ blocks of $b/2 \times b/2$ matrices, i.e.

$$\boldsymbol{X} = (\boldsymbol{X}_1, \cdots, \boldsymbol{X}_r).$$

Then $(\boldsymbol{W}^T \boldsymbol{X}_1 \cdots \boldsymbol{W}^T \boldsymbol{X}_r)$ has all entries of $\boldsymbol{B}$ plus some 0 entries. We can keep the extra entries or discard them. Suppose two $n \times n$ matrices can be multiplied in $O(n^\omega)$ time, then $\boldsymbol{B}$ can be computed in $O(rn^\omega) = O\left(\lceil a/b \rceil\, b^\omega\right)$ time. The rest of the steps in Flood_Time take $O(bk)$ time. Let $T(a, b)$ denote the maximum running time of the procedure for Flood_Time$(a, b, \boldsymbol{F}^{<}_{a:b})$. Then we obtain the following recurrence for $T(a, b)$:

$$T(a,b) = \begin{cases} O(a) & \text{for } b = 1, \\ T\left(a, \frac{b}{2}\right) + T\left(a + \frac{b}{2}, \frac{b}{2}\right) + O\left(\lceil \frac{a}{b} \rceil\, b^\omega + bk\right) & \text{for } b > 1. \end{cases}$$

The solution to the above recurrence is

$$T(a,b) = O\left(\left\lceil \frac{a}{b} \right\rceil b^\omega + bk \log k\right).$$

Since the procedure is initially called with $a = 1$, and $b = k+1$, the overall running time of the recursive procedure is $O\left(k^\omega + k^2 \log k\right)$. Combining this with the time spent in computing the tributary graph $\mathrm{G}[q]$, we obtain the following:

Theorem 6.1. Given a triangulation of $\mathbb{M}$ of $\mathbb{R}^2$ with n vertices, and a height function $h : \mathbb{M} \to \mathbb{R}$ which is linear on each face of $\mathbb{M}$, a data structure of size $O(n)$ (resp. $O(nm)$) can be constructed in time $O(n \log n)$ (resp. $O(nm + n \log n)$) that for a rain distribution $\mathcal{R}$ and a point q, can answer a flood-time query in time $O(nk + k^\omega + k^2 \log k)$ (resp. $O(|\mathcal{R}|k + k^\omega + k^2 \log k)$). Here k is the number of maximal depressions containing q, m is the number of sinks in the terrain $(\mathbb{M}, h)$, and k^ω is the time it takes to multiply two $k \times k$ matrices.

Approximate flood-time query. Given a rain distribution $\mathcal{R}$ and point $q \in \mathbb{M}$, let the answer to the exact flood-time query be τ_q. We say that an algorithm is an α-approximation, for parameter $\alpha > 1$, if it returns a value τ such that $\tau_q \le \tau \le \alpha \tau_q$.

Choosing indices j_i such that

$$\mathrm{Vol}(\beta_{j_1}) \le \mathrm{Vol}(\beta_{j_2}) \le \cdots \le \mathrm{Vol}(\beta_{j_k}),$$

let $V_i = \mathrm{Vol}(\beta_{j_i})$ for ease of notation, and let $\mathrm{Vol}_{h+1} = h\mathrm{Vol}_h$. We note that $\mathrm{Vol}_{h+1} \ge \sum_{i=1}^{h} \mathrm{Vol}_i$, so after a volume of Vol_{h+1} rain falls, all depressions in $\mathrm{G}[q]$ will be full. It follows that $\sigma(\mathcal{R}, q) \le \mathrm{Vol}_{h+1}$. Performing a binary search, using $O(\log h)$ point-flood queries, we can compute the index i such that q is not flooded at time V_i , but q is flooded at time V_{i+1}, in time $O(|\mathsf{E}_q| \log k)$.

Then we perform a binary search on τ over the set of values $\left\{\alpha^j V_i \mid 0 \le j \le \left\lceil \log_\alpha (V_{i+1}/V_i) \right\rceil\right\}$ to find value τ such that

$$\tau \le \tau_q \le \alpha\tau.$$

We note that $\alpha\tau$ is an α-approximation of the flood-time query.

Let $\rho = \max_{1 \le j < h} \frac{\mathrm{Vol}_{j+1}}{\mathrm{Vol}_j}$, be the *volume spread* of q. We can get an α-approximation of the fill time τ_q by performing $O\left(\log_\alpha \rho\right)$ point-flood queries. For example, if the largest ratio between two consecutively sized tributaries is at most 2^h then we can get a $2^{1/h}$-approximation in time $O(|\mathsf{E}_q| \cdot \log h)$.

Theorem 6.2. Given a triangulation of $\mathbb{M}$ of $\mathbb{R}^2$ with n vertices and a height function $h : \mathbb{M} \to \mathbb{R}$ which is linear on each face of $\mathbb{M}$, a data structure of size $O(n)$ (resp. $O(nm)$) can be constructed in time $O(n \log n)$ (resp. $O(n \log n + nm)$) that for a rain distribution $\mathcal{R}$ and a point $q \in \mathbb{M}$, returns an α-approximation to the flood-time query in time $O\left(nk + k^2 \log(n \log_\alpha \rho)\right)$ (resp. $O\left(|\mathcal{R}|k + k^2 \log(n \log_\alpha \rho)\right)$), where k is the number of maximal depressions containing q, and ρ is the volume spread of q.

7 EXPERIMENTS

In this section we present the experiments we have conducted on real terrains to demonstrate the efficiency of our algorithms and compare qualitatively the flooding under SFD and MFD models.

We have implemented the terrain-flood algorithm, described in Section 4, in C++, and we have run experiments on a machine with a 3.2 GHz Intel i5-6500 ®and 8 GB RAM running Windows 10. For

simplicity of coding and performance, the algorithm was implemented for grid DEMs rather than general TINs.

Data sets. We study the performance of our algorithm on two datasets, each consisting of 10^6 vertices.

(i) The *Indiana dataset*, a 0.89 mi^2 model of an area 0.5 mi northeast of Holland, Indiana, USA, extracted from the publicly available 5 ft resolution DEM of Indiana [10].

(ii) The *Philadelphia dataset* is a 9 km^2 model of an area in the northwest area of Philadelphia, extracted from the publicly available 3 m resolution DEM of Pennsylvania.

Flow models. For the SFD model, for a pair of neighboring vertices u, v, we set

$$\lambda(u,v) = \begin{cases} 1 & \text{if } v \text{ is lowest neighbor of } u, \\ 0 & \text{otherwise.} \end{cases}$$

For MFD model, we use a flow model similar to Quinn et al. [15]. That is, let $\Delta(u,v)$ be the gradient of the edge $(u,v) \in \mathbb{M}$. We define the flow to be proportional to the gradient, with the total (local) flow out of every non-sink vertex being 1:

$$\lambda(u,v) = \begin{cases} 0 & \text{if } u \text{ is a sink,} \\ \frac{\max\{\Delta(u,v),0\}}{\sum_{(u,v)\in\mathbb{E}} \max\{\Delta(u,v),0\}} & \text{otherwise.} \end{cases}$$

In Figure 1, we give an example of the two models on the Indiana dataset. The vertex marked p is a local maximum, and each vertex v with $\phi(p,v) > 0$ is marked in blue, with darker blue indicating a larger value of $\phi(p,v)$. In Figure 1 (a), under the SFD model flow is positive only along the path of steepest descent to a single sink. In Figure 1 (b), we note that under the MFD model water falling at p reaches multiple sinks.

Queries. We considered the rain distribution $\mathcal{R}$ to be rain falling: (i) on a single vertex, (ii) uniformly over a small rectangle, or (iii) uniformly over all vertices on the terrain.

For the case when rain falls at a single point, we computed the flooded areas with rain volume of 500ft^3. Figure 6 shows the terrain-flood query for two singe point rain distributions under both the SFD and MFD models on the Indiana data set. In Figure 6 (b) and (a), respectively, we see that under SFD the water follows a single path towards the lower region on the left (corresponding to a river) while under MFD a depression to the north west and one to the south west also become flooded. In Figure 6 (d) and (c), the point it is flooding on is near a ridge of the terrain, and the difference is more pronounced; under MFD a significant portion of the water flows south west, and a large region becomes flooded that receives no water under SFD.

For the case where rain is falling uniformly over a small square, we set the rain distribution to be uniform over a square of size 300m × 300m and set ψ = 400m^3 and computed the flooded area for the Philadelphia data set, under both SFD and MFD models. In Figure 7 we see that while similar areas become flooded, water is more dispersed across the region under the MFD model, while more water goes directly to the river on the left side under the SFD model.

For the case when rain falls uniformly over the entire Indiana data set, we computed the flooded area with ψ = 291ft^3. Figure 8 shows the flood area under both models. Under the MFD model, we see many spotted areas, showing the small depressions on the sink,

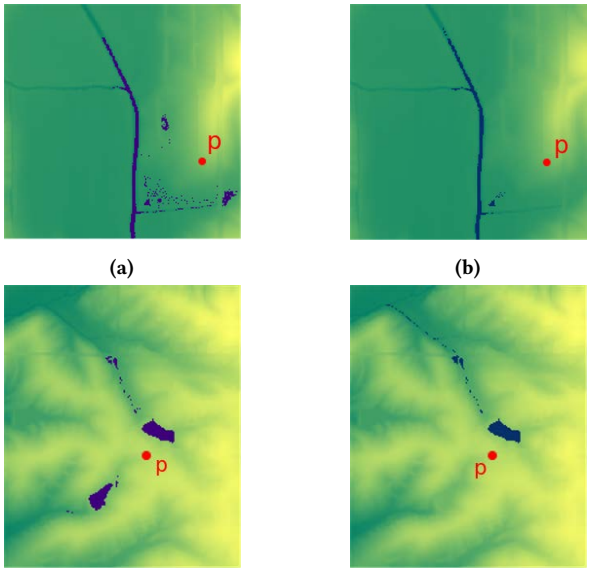


Figure 6: 500 ft^3 of rain falls at p, the flooded regions of Σ is marked in blue. Water flows according to MFD in (a) and (c) , and SFD in (b) and (d) .

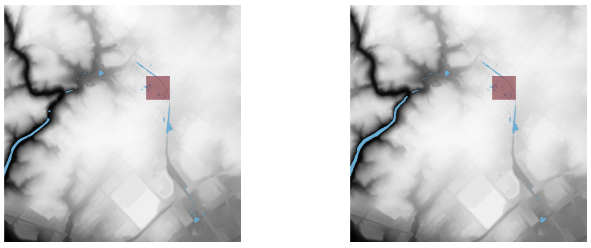


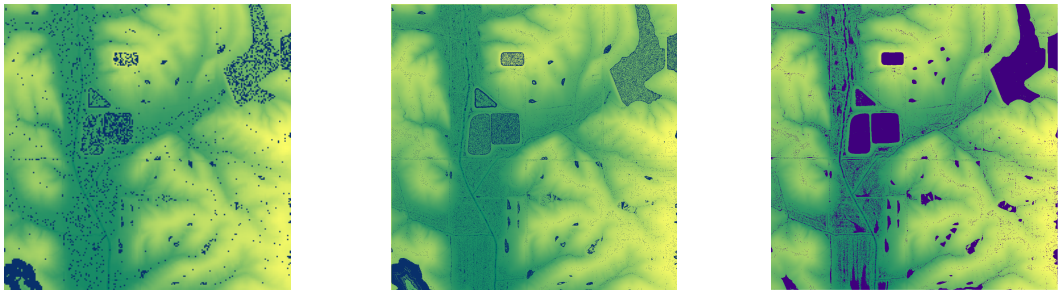
Figure 7: 4000 m^3 of rain falls uniformly over the red shaded rectangle. (a) Water flows according to MFD. (b) Water flows according to SFD.

mentioned above. There are some slight differences, but overall the flooded regions are similar.

Performance. Building the merge tree and preprocessing it to compute the depression volume of any point as well as to perform lca queries took an average of 4 seconds over 5 trials for both the Indiana data set and the Philadelphia data set.

Figure 9 gives a summary of the running time of the terrain flood query as we change the number of points with positive rain in $\mathcal{R}$ and the amount of rain ψ. In Figure 9 (a), we fix $\mathcal{R}$ to be uniform rain distribution over a 5 × 5 square of vertices and vary ψ, the volume of rain. In Figure 9 (b), the distribution $\mathcal{R}$ is uniform over a square with a fixed upper left corner but varying its side length from 1 to 20 vertices; rain volume is fixed to ψ = 100ft^3. In both figures, each line represents a different query location, and we tested 2 locations on each of the Indiana and Philadelphia datasets.

In Figure 9 (top) we see that as the volume of rain increases the running time increases as well, because more spill events occur and



(a) (b) (c)

Figure 8: Rain falls uniformly over the entire terrain, with the area flooded marked in blue. (a) SFD model with 291ft^3 rain. (b) MFD model with 291ft^3 rain. (c) MFD model with 240,000ft^3 rain.

must be processed as the rain volume increases. In Figure 9 (bottom) we see that the running time increases with $|\mathcal{R}|$. While the worst-case running time of our algorithm is independent of $|\mathcal{R}|$, in practice each iteration of our terrain-flooding algorithm does not take $O(n)$ time, but rather time proportional to the number of vertices over which water is flowing. As we increase the size of $|\mathcal{R}|$, rain flows over more vertices, and this increase may be nonuniform.

We also ran tests over the Philadelphia data set, expanding the area to regions with 9×10^6 and 16×10^6 vertices, each a superset of the smaller region(s). Building the merge tree and preprocessing it to compute the depression volume of any point as well as perform lca queries took an average of 47 and 121 seconds respectively over 5 tests. This is in line with the expected $O(n \log n)$ preprocessing time. We computed 5 queries with the same rain distribution over the three regions. The average running time was 5 seconds, 54 seconds and 103 second respectively. For our implementation, the query time depends on the size of the terrain. Further optimization of the code could have reduced this significantly.

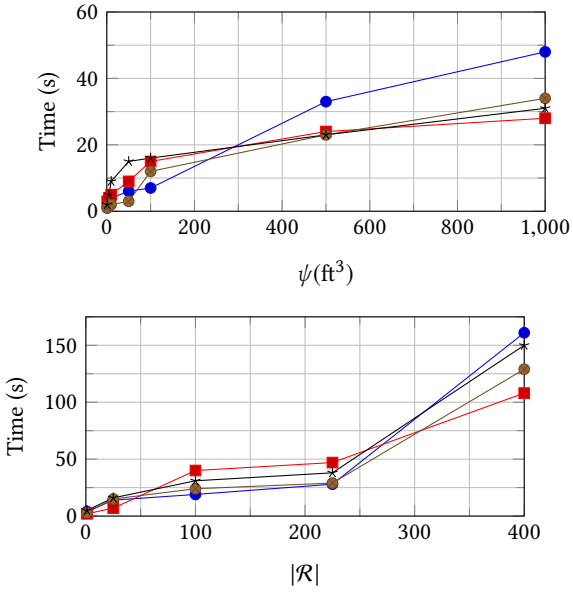


Figure 9: Query time as we vary: (top) the amount of rain, ψ and (bottom) the size of $|\mathcal{R}|$.

ACKNOWLEDGMENTS

Work by Lowe and Agarwal is supported by NSF under grants CCF-15-13816, CCF-15-46392, and IIS-14-08846, by ARO grant W911NF-15-1-0408, and by Grant 2012/229 from the U.S.-Israel Binational Science Foundation.

REFERENCES

[1] PK Agarwal, L Arge, and K Yi. 2006. I/O-efficient Batched Union-Find and its Applications to Terrain Analysis. In *Proc. 22nd Annu. Sympos. on Comp. Geom.* 167–176.
[2] L Arge, JS Chase, P Halpin, L Toma, JS Vitter, D Urban, and R Wickremesinghe. 2003. Efficient flow computation on massive grid terrain datasets. *GeoInformatica* 7, 4 (2003), 283–313.
[3] L Arge, M Rav, S Raza, and M Revsbæk. 2017. I/O-Efficient Event Based Depression Flood Risk. In *Proc. 19th Workshop on Algorithm Engineering and Experiments.* 259–269.
[4] L Arge and M Revsbæk. 2009. I/O-efficient Contour Tree Simplification. In *Intl. Sympos. on Algos. and Computation.* 1155–1165.
[5] L Arge, M Revsbæk, and N Zeh. 2010. I/O-efficient computation of water flow across a terrain. In *Proc. 26th Annu. Sympos. on Comp. Geom.* 403–412.
[6] H Carr, J Snoeyink, and U Axen. 2003. Computing contour trees in all dimensions. *Comp. Geom.* 24, 2 (2003), 75–94.
[7] H Carr, J Snoeyink, and M Panne. 2010. Flexible isosurfaces: Simplifying and displaying scalar topology using the contour tree. *Comp. Geom.* 43, 1 (2010), 42–58.
[8] A Danner, T Mølhave, K Yi, PK Agarwal, L Arge, and H Mitásová. 2007. TerraStream: from elevation data to watershed hierarchies. In *Proc. 15th Annu. ACM Intl. Sympos. on Advances in GIS.* 28.
[9] H Edelsbrunner, J Harer, and A Zomorodian. 2001. Hierarchical Morse complexes for piecewise linear 2-manifolds. In *Proc. 17th Annu. Sympos. Comp. Geom.* 70–79.
[10] Indiana Spatial Data Portal. 2013. Indiana Orthophotography (RGBI), LiDAR and Elevation. http://gis.iu.edu/datasetInfo/statewide/in_2011.php.
[11] SK Jenson and JO Domingue. 1988. Extracting topographic structure from digital elevation data for geographic information system analysis. *Photogrammetric Engineering and Remote Sensing* 54, 11 (1988), 1593–1600.
[12] M Kreveld, R Oostrum, C Bajaj, V Pascucci, and D Schikore. 1997. Contour trees and small seed sets for isosurface traversal. In *Proc. 13th Annu. Sympos. on Comp. Geom.* 212–220.
[13] Y Liu and J Snoeyink. 2005. Flooding triangulated terrain. In *Proc. 11th Intl. Sympos. on Spatial Data Handling.* 137–148.
[14] JF O'Callaghan and DM Mark. 1984. The extraction of drainage networks from digital elevation data. *Computer Vision, Graphics, and Image Processing* 28, 3 (1984), 323–344.
[15] PFBJ Quinn, K Beven, P Chevallier, and O Planchon. 1991. The prediction of hillslope flow paths for distributed hydrological modelling using digital terrain models. *Hydrological processes* 5, 1 (1991), 59–79.
[16] M Rav, A Lowe, and PK Agarwal. 2017. Flood Risk Analysis on Terrains. In *Proc. of the 25th ACM SIGSPATIAL Int. Conference on Advances in GIS.* ACM, 36.
[17] SP Tarasov and MN Vyalyi. 1998. Construction of contour trees in 3D in O(n log n) steps. In *Proc. 14th Annu. Sympos. on Comp. Geom.* 68–75.