Flood Risk Analysis on Terrains

MATHIAS RAV, Aarhus University, Denmark

AARON LOWE and PANKAJ K. AGARWAL, Duke University

2

An important problem in terrain analysis is modeling how water flows across a terrain and creates floods by filling up depressions. In this article, we study the *flooding query* problem: Preprocess a given terrain Σ , represented as a triangulated xy -monotone surface with n vertices, into a data structure so that for a query rain region R and a query point q on Σ , one can quickly determine how much rain has to fall in R so that q is flooded. Available terrain data is often subject to uncertainty, which must be incorporated into the terrain analysis. For instance, the digital elevation models of terrains have to be refined to incorporate underground pipes, tunnels, and waterways under bridges, but there is often uncertainty in their existence. By representing the uncertainty in the terrain data explicitly, we can develop methods for flood risk analysis that properly incorporate terrain uncertainty when reporting what areas are at risk of flooding.

We present two results. First, we present an $O(n \log n)$ -time algorithm for preprocessing Σ with a linear-size data structure that can answer a flooding query in $O(|R| + m \log n)$ time, where $|R|$ is the number of vertices in R , m is the number of so-called tributaries of q at which rain is falling, and n is the number of vertices of the terrain. Next, we extend this data structure to handle “uncertain” terrains using a standard Monte Carlo method. Given a probability distribution on terrain data, our data structure returns the probability of a query point being flooded if a specified amount of rain falls on a query region. We implement our data structure and test it on real terrains, showing that a small number of samples suffice to accurately estimate the flood risk.

CCS Concepts: • **Information systems** → **Geographic information systems**; *Uncertainty*;

Additional Key Words and Phrases: Terrains, geographical information systems, contour trees, stochastic process, data uncertainty, Monte Carlo method

ACM Reference format:

Mathias Rav, Aaron Lowe, and Pankaj K. Agarwal. 2019. Flood Risk Analysis on Terrains. *ACM Trans. Spatial Algorithms Syst.* 5, 1, Article 2 (May 2019), 31 pages.

<https://doi.org/10.1145/3295459>

Work by Rav is supported by the Danish National Research Foundation and Innovation Fund Denmark; work was done while visiting Duke University. 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.

Authors' addresses: M. Rav, Aarhus University, MADALGO, Department of Computer Science, Åbogade 34, Aarhus, Denmark; email: rav@cs.au.dk; A. Lowe and P. K. Agarwal, Department of Computer Science, Duke University, 308 Research Dr, Durham, NC 27705, USA; emails: {aaron, pankaj}@cs.duke.edu.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

© 2019 Copyright held by the owner/author(s). Publication rights licensed to ACM.

2374-0353/2019/05-ART2 \$15.00

<https://doi.org/10.1145/3295459>

1 INTRODUCTION

An important problem in terrain analysis is modeling how water flows across a terrain and creates floods by filling up depressions. When rain 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. In this article, we study the *flooding query* problem: Preprocess a given terrain Σ , represented as a triangulated xy -monotone surface¹ with n vertices, into a data structure so that for a query rain region R and a query point q on Σ , one can quickly determine how much rain has to fall in R so that q is flooded.

Current geographic information systems (GIS) algorithms often assume that the height of each vertex is specified precisely. In reality, however, terrain models approximate the ground truth and are subject to a degree of uncertainty. Large high-resolution terrain datasets such as the ones acquired by LiDAR and other similar technology contain measurement error; for the publicly available 1.6m terrain model of Denmark, the root mean square elevation error is 10cm [13]. Furthermore, raw data is noisy as it includes heights of various natural and anthropological features (e.g., bridges, trees, electric wires, cars, and mailboxes), and one has to clean the data to get the height of the ground. The data cleaning is performed manually, automatically, or often using a hybrid approach. The output after data cleaning (whether manual or automatic) is not perfect, and this causes uncertainty in the resulting terrain model. By representing the uncertainty in terrain data explicitly, we develop methods for flood risk analysis that compute the probability of a point q on Σ being flooded when a specified volume of rain falls on a given region R . See Figure 1.

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 [3, 21, 25]. 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, 6, 12, 14]. 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 remove them by “flooding” them.

Liu and Snoeyink [23] (see also Ref. [7]) proposed an $O(n \log n)$ time algorithm 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. [4] described an algorithm to compute the set of flooded vertices when a given volume of rain $\psi \geq 0$ falls on a given region R . We are not aware of any algorithm that can answer a flooding query in sub-linear time in the worst case.

As the resolution in available terrain models increases, there is an increasing demand for methods to handle terrain models that do not fit into main memory. However, the size of the terrain data is only one aspect, and it is equally important to be able to handle uncertainty in the terrain data. To this end, Agarwal et al. [2] proposed data structures for answering a few queries on uncertain terrains such as the *sea-level rise flood risk problem*, that is, they compute the flooding probability

¹An xy -monotone surface is represented by an equation $z = h(x, y)$ for some continuous height function $h : \mathbb{R}^2 \rightarrow \mathbb{R}$.

Fig. 1. When rain falls on p , the existence of each underground pipe (white segment) determines which areas are flooded. (a) Using the flooding query (without terrain uncertainty), the flooded areas for a given rain amount ψ on p can quickly be determined. (b) When each pipe exists with probability 0.5, our algorithm quickly estimates the flooding probability of each point of the terrain for a given rain amount ψ .

of a query point q given that the sea-level at a point p increases by ℓ unit of height. They model uncertain terrains by assuming that the height of each vertex is given by a probability distribution such that vertex heights are independent. They show that it is enough to evaluate the sea-level rise scenario on a small number of random terrains to guarantee that the estimate of the flooding probability is within ε of correct with high probability. They describe data structures for discrete distributions, in which each terrain vertex can take at most k distinct height values, and show that continuous distributions can be handled using approximations by discrete distributions.

Our Results. We present two main results. In Section 3, we present an $O(n \log n)$ -time algorithm for preprocessing Σ into a linear-size data structure that can answer the flooding query for a query rain region R and a query point q . We first describe a simple data structure using the so-called merge tree of Σ with $O(|R| + H \log n)$ query time, where $|R|$ is the number of vertices at which rain is falling, and H is the height of the merge tree, which is $\Theta(n)$ in the worst case. Roughly speaking, it computes the maximal depressions that need to be filled before q is flooded. We then present a fast data structure with $O(|R| + m \log n)$ query time, where m is the number of so-called tributaries of q at which rain is falling, where a *tributary* of q is a depression that spills into a depression containing q when full; the precise definition is given in Section 3. Since there is a one-to-one mapping between the tributaries of q and the ancestors of q in the merge tree, m is less or equal to H . Furthermore, m is bounded by $|R|$, which implies a query time of $O(\log n)$ when rain is falling at $O(1)$ vertices of the terrain. The fast data structure uses a heavy-path decomposition [27] of the merge tree to compress the set of “full depressions” to $O(\log n)$ sets, and to compute when they are filled.

Next, we consider computing the probability of flooding a query point q given that it has rained a volume of ψ on the rain region R on an uncertain terrain. Our model of uncertainty is similar to the one of Agarwal et al. [2]. We assume that the heights of terrain vertices are given by a probability distribution in which each vertex can take at most k distinct height values; contrary to Ref. [2], we do not assume that vertex heights are independent. The parameter k reflects how many different height values could be obtained during raw data acquisition for a given terrain location, so for terrain uncertainty arising from data cleaning, k is a small constant in practice. In Section 4, we show that computing the flooding probability exactly is #P-hard using a reduction from #KNAPSACK. Our hardness result works even in the more restricted model where vertex heights must be independent, as our reduction from #KNAPSACK uses an uncertain terrain in which vertex heights are independent. We extend our data structure (in Section 5) to estimate the flooding probability by following a standard Monte-Carlo approach: fix a parameter s , choose s random terrains from the uncertain terrain, and preprocess each of them using the data structure

Table 1. Overview of Notation

Symbol	Meaning
$\mathbb{M}$	Triangulation of $\mathbb{R}^2$
$h : \mathbb{M} \rightarrow \mathbb{R}$	Height function
$\mathbb{V}$	Vertex set of $\mathbb{M}$
$n = \mathbb{V} $	Number of vertices in the terrain
Σ_h	Triangulated terrain surface (graph of h)
T_h	Merge tree
$h_{<\ell}$	ℓ -sublevel set of h (union of depressions at level ℓ)
$h_{=\ell}$	ℓ -level set of h (union of contours at level ℓ)
$V_h(\beta)$	Volume of depression β
$\mu_h(x)$	Volume of depression delimited by point $x \in \mathbb{M}$
S_q	Tributary tree of q
P	Heavy-path tree of merge tree T
S_π	Tributary tree of heavy path π
ρ_β	Path in a tributary tree from β to the root
$R : \mathbb{V} \rightarrow \mathbb{R}_{\geq 0}$	Rain region assigning rainfall to each vertex
$ R $	Number of vertices with positive rainfall
m	Number of tributaries with positive rainfall
$A(R, q)$	Tributaries from R to q
$f_\beta(t)$	Fill rate of depression β at time t
$g_\beta(t)$	Spill rate of β at time t
$\mathbb{F}$	Set of all monotone piecewise constant functions $\mathbb{R}_{\geq 0} \rightarrow \mathbb{R}_{\geq 0}$
t_β	Fill time of β
H	Distribution over height functions
ψ	Volume of rainfall
k	Description complexity of uncertain terrain
M_h	Height-level map of h
$\pi(R, q, \psi)$	Probability that q is flooded when rain with volume ψ falls on R

Top: Notation related to a given terrain. Middle: Notation related to a given rain region. Bottom: Notation related to terrain uncertainty.

in Section 3 to determine whether q is flooded. The main technical challenge is to bound the value of s to ensure a desired level of accuracy. Since the number of different terrains can be k^n , a standard analysis based on Chernoff bounds suggests that one has to set $s = \Omega(n \log k)$ [2]. Using sophisticated techniques from combinatorial geometry and probabilistic methods, we show that only $O(\frac{1}{\varepsilon^2} \log \frac{nk}{\delta})$ samples are needed for all single-point queries, and only $O(\frac{k^3 \log n}{\varepsilon^2} \log \frac{nk}{\delta})$ samples are needed for a general rain region, to ensure that the answer is correct within error ε with probability at least $1 - \delta$.

We have conducted experiments with our algorithms on two realistic datasets, and in Section 6, we show that in practice the estimates converge to the true risk after a small number of samples are examined, and that the resulting output imparts interesting information about the flood risk over the terrains.

This study is an expanded version of Ref. [26]. Whereas the data structure for terrains without uncertainty in Ref. [26] had a query time of $O(|R| + m' \log^2 n)$, where m' is the number of sinks with positive rainfall, we have improved the query time to $O(|R| + m \log n)$ by restructuring the query algorithm significantly. Besides improving the query time, we provide a complete description of the

Fig. 2. Example terrain showing a depression that is not simply connected. The depression contains three sinks (minima), marked with squares, and a maximum is marked with a circle.

query algorithm including the most general case in which rain falls on multiple sinks of the terrain. This case was omitted in Ref. [26] because of lack of space. In Section 4, we prove that computing the flooding probability exactly is $\#P$ -hard. In Appendix A, we give the details of how to preprocess a merge tree in $O(n \log n)$ time into a data structure of linear size so that the depression volume of any point $x \in \mathbb{R}^2$ can be computed in $O(\log n)$ time. Although this is implied by the analysis of Carr et al. [11], to our knowledge, this result has not been written down explicitly before.

2 PRELIMINARIES

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 $h : \mathbb{M} \rightarrow \mathbb{R}$ 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 Σ_h , is an xy -monotone triangulated surface whose triangulation is induced by $\mathbb{M}$. If h is clear from the context, we denote Σ_h by Σ .

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 a *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*. Consider visiting the neighbors of v in clockwise order. If there exist four neighbors w_1, w_2, w_3, w_4 in clockwise order around v 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 ℓ -*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, but can be bounded by multiple contours, as maxima can cause holes to appear; refer to Figure 2 for an example.

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*. Refer to Figure 3. The

Fig. 3. 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. The maximal depressions α_1 , β_1 , α_3 , and β_3 are elementary. (b) Terrain seen from the side, showing the watershed of u_4 marked with a dotted line. The merge tree of the terrain shows how the maximal depressions nest within each other.

volume of a depression β of $h_{<\ell}$ is

$$V_h(\beta) = \int_{\beta} (\ell - h(x)) dx, \quad (1)$$

where the integral is taken over the region $\beta \subset \mathbb{R}^2$. For a point x that is not a sink or a negative saddle, we define the *depression volume* of x to be $\mu_h(x) = V_h(\beta_x)$, where β_x is the depression delimited by x .

Merge Tree. The maximal depressions of a terrain form a hierarchy that is easily represented using a rooted tree, called the *merge tree* [11, 22] 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, and (u, v) is an edge in T_h if v delimits a maximal depression β containing u and u is higher than all other sinks and saddles in β [11]. 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 β delimited by u and containing v . Each point on an edge of T_h corresponds to a depression. The depressions corresponding to the points on an edge contain the same set of sinks, namely, those in the subtree of T_h below the edge (u, v) . Refer to Figure 3(b). 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 [15].

Formally, T_h is the quotient space in which each depression is represented by a point and connectivity is defined in terms of the quotient topology. Let $\rho_h : \mathbb{M} \rightarrow T_h$ be the associated quotient map, which maps all points that delimit a depression to a single point on an edge of T_h . Fix a point x in $\mathbb{M}$. If x is not a sink or negative saddle, $\rho_h(x)$ lies in the relative interior of an edge in T_h ; if x is a sink, $\rho_h(x)$ is a leaf node of T_h ; and if x is a negative saddle, then $\rho_h(x)$ is a non-leaf node of T_h . We use h to denote the height function on the points of T_h as well.

Van Kreveld et al. [22] gave an $O(n \log n)$ -time algorithm for constructing the merge tree. The algorithm was later extended to 3D by Tarasov and Vyalys [28], and to arbitrary dimensions by Carr et al. [11]. The algorithm in Ref. [11] can be extended to preprocess T in $O(n \log n)$ time into

a linear-size data structure so that for a query point $q \in \mathbb{R}^2$, $\mu_h(x)$, the volume of the depression β_x can be computed in $O(\log n)$ time. Refer to Appendix A for the details.

Uncertainty Model. In our uncertainty model, $\mathbb{M}$ is fixed but the height function h is drawn from a distribution H . We assume that the height of vertex v_i is drawn from a discrete set $H_i = \{h_i^1, \dots, h_i^k\}$, and we say that H has *description complexity* k . We use h to denote a random height function drawn from H . H induces distributions Σ_H and T_H over terrains and merge trees. Σ_h and T_h are random terrain and merge tree, drawn from Σ_H and T_H , respectively. We do not model uncertainty in the location or amount of rain, but instead let these be inputs to the query.

Note that we do not require the vertex heights to be independent. When a terrain model is cleaned to remove features such as bridges and trees, the identification itself may be faulty or noisy, and this noise can be embedded as height uncertainty. In this case, the vertex heights are not independent, as the existence of the feature and the height of the ground under the feature causes correlated changes to the heights of the covered terrain vertices.

We require a discrete distribution with at most k height values for each vertex. Previously, Agarwal et al. [2] showed that certain queries on uncertain terrains with continuous height distributions can be ε -approximated by discrete height distributions with $k = O((n^2/\varepsilon^2) \log(n/\delta))$ with probability $1 - \delta$ and used this result to extend their work to continuous height distribution. However, their analysis requires vertex heights to be independent; since we do not wish to require independent vertex heights, in this article, we restrict our attention to discrete height distributions.

3 FLOODING QUERIES

In this section, we give a precise definition of the flooding query, and then we present two data structures to compute the flooding query. The first data structure (Section 3.2) traverses the merge tree directly and has a query time that depends on the height of the merge tree. The second data structure (Section 3.3) speeds up the worst-case query time by using a heavy-path decomposition of the merge tree.

3.1 Flooding Model

To model water flow on a terrain, each non-sink vertex v is assigned a *flow direction* that indicates the neighbor to which water will flow from v . The flow direction of v is the lowest neighbor v' of v . Each sink u defines a unique *watershed*, consisting of all vertices v that are reachable to u by following the flow directions. Note that each vertex belongs to exactly one watershed, and we assume that each vertex has a pointer to the sink defining the watershed.

For a negative saddle v , the lowest vertices in the two sets of adjacent downslope neighbors of v define two directions, of which one is the flow direction of v ; the other is the so-called *secondary flow direction*. By following flow directions from v , we reach a sink u_1 containing v in its watershed. Furthermore, by following flow directions from the secondary flow direction of v , we reach another sink u_2 . We say that u_1 and u_2 are the *spill sinks* of v . Let β_1 be the maximal depression delimited by v and containing u_1 . Then, the *spill sink* of β_1 is u_2 . Similarly, we define the spill sink of β_2 to be u_1 .

We let R denote a *rain region*, which is specified as a vector on the vertices of Σ such that for each vertex $v \in \mathbb{V}$, $R(v) \geq 0$ indicates the rate at which rain falls on v ; we require $\sum_v R(v) = 1$. We denote by $|R|$ the number of vertices with positive rainfall in R , and we assume that R is represented as a list of $|R|$ pairs $(v, R(v))$.

Our flooding model follows the depression filling model of Liu and Snoeyink [23]. When rain falls on a region R of Σ , water follows flow directions and accumulates in depressions of Σ . When rain falls on the watershed of a sink u , the rate at which the elementary depression β containing u

fills up is equal to the sum of $R(v)$ over all vertices v in its watershed. When a maximal depression β containing u fills up, water on the watershed of u spills over the saddle delimiting β into a neighboring watershed of a sink u' . We refer to this event as a *spill event*. At this event, the watershed of u is merged into the watershed of u' , which increases the rate at which u' is filling up.

The above process defines a sequence of spill events, each event marking a sink u as full and merging the watershed of u into a neighboring watershed. 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. The *R-fill time* (or *fill time* for brevity) of a point $q \in \mathbb{M}$ is the time when the rain water (on R) reaches q . Given a height function h and a rain region R , let $\sigma_h(R, q)$ denote the total volume it must rain on R before the water surface reaches q . For simplicity, we assume throughout the article that the point q is not a saddle. Our definitions and algorithms can easily be extended to handle saddles. Since we assume that rain falls at the rate of unit volume, the fill time of q is the same as $\sigma_h(R, q)$.

When rain falls on a vertex v , water follows flow directions toward a sink u . Given a rain region R , we can define an equivalent rain region R' in which $R'(u)$ for a sink u is the sum of $R(v)$ over vertices v in its watershed, and $R'(v)$ is zero when v is not a sink. Then, $\sigma_h(R, q) = \sigma_h(R', q)$ for all q , so we may always replace R by R' in $O(|R|)$ time and then assume that it is only raining on sinks. When all rain in R is contained in the watershed of a sink p , we denote $\sigma_h(R, q)$ by $\sigma_h(p, q)$.

Any given point $q \in \mathbb{M}$ is contained in a sequence of maximal depressions $\alpha_1 \supset \dots \supset \alpha_k \ni q$, each α_i delimited by a saddle v_i , which delimits another maximal depression β_i . Note that these saddles form a path in T_h from q to the root. We refer to the maximal depressions $\beta_1, \dots, \beta_k$ as the *q-tributaries*. The *q-tributaries* form the *q-tributary tree*, denoted by S_q and defined as follows. S_q is a directed tree, with each node pointing to its parent, rooted at α_k , the smallest maximal depression containing q , and the non-root nodes are the *q-tributaries* $\beta_1, \dots, \beta_k$.² We define the parent of β_i in S_q as the node containing the spill sink of β_i , which is either the root α_k or β_j for some $j > i$. For example, β_1 and β_3 are the children of β_4 in Figure 4.

Given a rain region R and a query point q , for each node $\beta \in S_q$ (either a *q-tributary* or α_k), we denote by $f_\beta(t)$ the *fill rate of β at time t* , which is the rate at which rain is falling directly on β plus the rate at which other *q-tributaries* are spilling into β . The fill rate $f_\beta : \mathbb{R}_{\geq 0} \rightarrow \mathbb{R}_{\geq 0}$ is a monotone piecewise constant function, and we denote the set of all such functions by $\mathbb{F}$. We denote by t_β the *fill time of β* and by $g_\beta(t)$ the *spill rate of β at time t* , and we define these formally as follows. For any function $f \in \mathbb{F}$ and $\psi \geq 0$, we let $\tau(f, \psi)$ be the time at which the integral of f is ψ , that is, $\tau(f, \psi)$ is t_0 such that $\int_0^{t_0} f(t)dt = \psi$. Refer to Figure 5. Then, $t_\beta = \tau(f_\beta, V_h(\beta))$, and we define the spill rate as

$$g_\beta(t) = \begin{cases} 0 & \text{for } t < t_\beta \\ f_\beta(t) & \text{for } t \geq t_\beta. \end{cases}$$

Using S_q we can now recursively define the fill rate of β as follows: the fill rate of β is the rate at which rain falls directly into β , plus the spill rates of the children of β in S_q . Let $R(\beta) = \sum_v R(v)$ where the sum is taken over all the vertices v in watersheds of sinks in β , and let B be the set of children of β in S_q . We define

$$f_\beta(t) = R(\beta) + \sum_{\gamma \in B} g_\gamma(t).$$

In particular, when β is a leaf of S_q , f_β is the constant function $R(\beta)$.

²There is a natural bijection between the saddles $v_1, \dots, v_k$ and the *q-tributaries* $\beta_1, \dots, \beta_k$, and thus, we sometimes refer to a saddle v_i as a node of S_q .

Fig. 4. Example terrain and query (p, q) . Sinks are marked with squares, and saddles are marked with labels 1–8 indicating the saddle elevation. Dotted lines indicate the two spill sinks of each saddle. **Top:** Merge tree. **Middle:** Terrain seen from the side. **Bottom:** Terrain seen from above. Blue arrows are the edges in the q -tributary tree S_q ; the shaded depressions form $A(p, q)$.

Fig. 5. The fill rate f_γ is computed from the spill rates g_{β_1} and g_{β_2} , and then the spill rate g_γ is computed from f_γ .

Using the fill rate of α , the root of S_q , we can obtain the time at which α is full, i.e., the fill time q , as $\tau(f_\alpha, \mu_h(q))$. Since we assume that rain falls at the rate of unit volume, the fill time of q is the same as the total volume it must rain on R before q is flooded, so

$$\sigma_h(R, q) = \tau(f_\alpha, \mu_h(q)).$$

We define the *tributaries from R to q* , denoted by $A(R, q)$, as the subtree of S_q induced by the vertices that have non-zero fill-rate functions.

Remark. Note that S_q depends on the smallest maximal depression, which contains q . For all points $p, q \in \Sigma$ on the same edge of T_h , we have $S_p = S_q$, and it follows that $A(R, p) = A(R, q)$.

Single-source Spilling. If R is contained in the watershed of a single sink p , $A(p, q)$ is a path of tributaries $(\gamma_1, \dots, \gamma_\ell)$, and we observe that $t_{\gamma_1} = V_h(\gamma_1)$, and $t_{\gamma_{i+1}} = t_{\gamma_i} + V_h(\gamma_{i+1})$. Thus, we have

$$\sigma_h(p, q) = \sum_{i=1}^{\ell} V_h(\gamma_i) + \mu_h(q). \quad (2)$$

3.2 A Simple Data Structure

Now, we describe a data structure for computing $\sigma_h(R, q)$ for a rain region R and query point q . Since the terrain Σ will be fixed, we omit the subscript h for the rest of this section. Our data structure consists of T , and stores the following information: for each saddle v in T , it stores

- pointers to the two spill sinks of v ,
- the volumes of the two maximal depressions delimited by v .

We augment the merge tree, using methods from Carr et al. [12], so that for any point $x \in \Sigma$, $\mu(x)$ can be computed in $O(\log n)$ time. Refer to Appendix A for the details. We compute a depth-first numbering of the leaves of T and store at each node of T the DFS (depth-first search) interval associated with its subtree. Using the DFS intervals, it is possible to check in constant time whether a sink u is contained in a subtree of T . The construction time is $O(n \log n)$, and the data structure has size $O(n)$. We also preprocess T for answering least-common ancestors (lca) queries, using the data structure by Harel and Tarjan [18]. Its preprocessing time, query time, and space are $O(n)$, $O(1)$, and $O(n)$, respectively, so we apply their technique without increasing the complexity of our data structure.

Now, we describe the query algorithm to compute $\sigma(R, q)$, first in the *single-point case* when R is restricted to the watershed of a single sink p , and then in the case when R is any region.

Single-point source. Suppose all rain in R falls on the watershed of a single sink p . Our algorithm to compute $\sigma(p, q)$ traverses the path $(v_1, \dots, v_k)$ in T from the root to the edge containing q and along the way identifies the path $A(p, q) = (\gamma_1, \dots, \gamma_\ell)$ in the q -tributary tree S_q as follows: For each $i \leq \ell$, let p_i be the sink in γ_i at which water initially collects when γ_i starts to fill up. Then p_1 is p , and p_{i+1} is the spill sink of γ_i . For $j = 1, \dots, k$, we visit the saddle v_j while maintaining the invariant that we have identified the tributaries that are delimited by the first $j - 1$ saddles $v_1, \dots, v_{j-1}$, and we have furthermore identified the next spill sink p_{j+1} . After we have visited v_k , we have identified all the tributaries between p and q . We return the rain volume $\sigma(p, q)$ computed using Equation (2), where each term $V(\gamma_i)$ is computed by looking up the volume stored at the merge tree node for the saddle delimiting γ_i .

Computing $\sigma(p, q)$ in this way requires $O(k)$ time, where k is the number of maximal depressions that contain q .

Region source. We now extend this algorithm to handle rain on the watersheds of multiple sinks of the terrain as specified by the rain region R . As in the single-point case, the algorithm visits the saddles $(v_1, \dots, v_k)$ on the path in T from the root to the edge containing q while incrementally identifying $A(R, q)$, the tributaries between R and q , and computing the fill and spill rates of the corresponding depressions as follows:

First, we add the rain rate $R(v)$ of each non-sink vertex v to the rain rate of the sink u containing v in its watershed. Next, we add the rain rate of each sink u to the node β_i containing u in the q -tributary tree S_q ; β_i is found by computing the lca of u and q in T . For each $i \leq k$, let ρ_i denote the resulting rain rate of β_i .

We now visit $v_1, \dots, v_k$ in this order. We maintain the following invariants when we are visiting v_i : The children of v_i have been computed, and for each edge $(\beta_r, \beta_s) \in S_q$ with $r < i \leq s$, the spill function g_{β_r} is stored at the node $\beta_s \in S_q$. We compute the fill-rate function

$$f_{\beta_i}(t) = \sum_j g_{\beta_j}(t) + \rho_i,$$

where the sum is taken over all children β_j of β_i in S_q . Given $f_{\beta_i}(t)$, g_{β_i} can be computed by computing the fill time t_{β_i} . Finally, the parent of β_i in S_q is computed in $O(1)$ time by computing the lca of the spill sink of β_i and q in T .

At the end, α , the root of S_q , stores the spill rate functions of the children of α as well as the rain rate of α , so we compute the fill-rate function f_α and return $\sigma(p, q) = \tau(f_\alpha, \mu(q))$. The only thing left is to show how f_{β_i} and g_{β_i} are computed efficiently because they are not constant-size functions.

Let g, g' be two spill rate functions. Using the representation described below, we support the following two operations:

Add. An ADD operation replaces g and g' by $g + g'$.

Truncate. The TRUNCATE operation takes a volume ψ , computes $t_0 = \tau(g, \psi)$ and sets $g(t) := 0$ for $t < t_0$.

Since each spill rate function g is piecewise constant, we represent g as a linked list of pairs (t_i, Δ_i) , where t_i is a time at which g changes value and Δ_i is the increase of g at time t_i . The collection of pairs is stored in a strict Fibonacci heap [10] keyed on time. ADD is then performed as a merge of two heaps, and TRUNCATE is computed by iteratively removing the top pair (t_i, Δ_i) and maintaining the total volume of rain at time t_i until it exceeds ψ . ADD takes constant time, and TRUNCATE takes time $O((r+1) \log n)$, where r is the number of pairs (t_i, Δ_i) removed by TRUNCATE. Since ADD does not change the total number of spill rate function point-pairs (t_i, Δ_i) and the initial number of point-pairs is $O(k)$, the time required in total for the TRUNCATE operations is $O(k \log n)$. We thus obtain the following:

THEOREM 3.1. *Given a triangulation $\mathbb{M}$ of $\mathbb{R}^2$ with vertex set $\mathbb{V}$ of size n and a height function $h : \mathbb{M} \rightarrow \mathbb{R}$ that is linear on each face of $\mathbb{M}$, a data structure of size $O(n)$ can be constructed in time $O(n \log n)$ that for any rain region R and query point $q \in \mathbb{R}^2$ returns $\sigma(R, q)$ in $O(|R| + k \log n)$ time, where $|R|$ is the number of vertices with positive rainfall, and k is the number of maximal depressions in the terrain containing q .*

3.3 The Fast Data Structure

In this section, we present a fast data structure for computing $\sigma(R, q)$. First, we describe the data structure, which consists of a number of trees, and then analyze its size and construction time. Finally, we describe the query algorithm to compute $\sigma(R, q)$.

3.3.1 Data Structure. We define the *heavy-path decomposition* [27] of T , as follows. For each internal node v in T , let w be the child of v with the largest subtree, with ties broken arbitrarily. We call the edge (v, w) *heavy*, and for all other children w' of v , we say the edge (v, w') is *light*. Then we say a path $\pi = (v_1, v_2, \dots, v_k)$ in T is a *heavy path* if the head v_1 is either the root of T , or the edge to its parent is light, the tail v_k is a leaf of T , and every edge (v_i, v_{i+1}) for $1 \leq i < k$ is heavy. Every parent has exactly one heavy edge to a child, so the set of heavy paths $\{\pi_i\}$ partitions the nodes of T (where some heavy paths may consist of a single sink). Refer to Figure 6(a).

Fig. 6. Overview of the fast data structure for computing $\sigma(R, q)$. (a) First, the merge tree T of Σ is computed along with its heavy-path decomposition. Light edges are dotted, and heavy paths are circled in purple. (b) Heavy-path tree P of T of height $O(\log n)$. (c) For each heavy path π of T , the tributary tree S_π is computed along with its heavy-path decomposition, only pictured for non-singleton paths π_1 – π_3 . Note that we use a heavy-path decomposition both of T and of each tributary tree S_π .

Next, we define the *heavy-path tree* P as follows: the nodes of P are the heavy paths of T , and P contains the edge (π_i, π_j) whenever the parent of the head of π_j in T is a node in π_i . There is a natural bijection between the light edges of T and the edges of P . We observe that for any light edge (v, w) in T , corresponding to an edge in P , the subtree rooted at w contains at most half as many vertices as the subtree rooted at v . Therefore, the height of P is at most $\lceil \log_2(n) \rceil$. Refer to Figure 6(b).

Fix a heavy path $\pi = (v_1, \dots, v_k, u)$, where for $i < k$, v_i is the parent of v_{i+1} in T , and v_k is the parent of the sink u . We define the *tributary tree of π* , denoted by S_π , as the subtree of the u -tributary tree containing only the tributaries delimited by the saddles $v_1, \dots, v_k$. The root of S_π is α_k , the (smallest) maximal depression containing u . Refer to Figure 6(c). We define the height of a tributary $\beta \in S_\pi$ to be the height of the saddle delimiting β . For a given node $\beta \in S_\pi$, let ρ_β be the path in S_π from β to the root α_u . Each tributary tree node $\beta \in S_\pi$ stores the volume of β , as well as the *suffix sum*, defined to be the sum of volumes of the tributaries in ρ_β . We compute a depth-first numbering of each tributary tree S_π and store with each node of S_π the DFS interval associated with its subtree. Each saddle v stores pointers to the two spill sinks of v , to the heavy path $\pi \in P$ containing v , and to its node in S_π . After constructing T in time $O(n \log n)$ [22], the additional structures can be built in time $O(n \log n)$ using standard techniques. Each node stores $O(1)$ information, so the total space is $O(n)$. Finally, we store each S_π as a heavy-path tree. As in the simple data structure, we use the data structure by Harel and Tarjan [18] to implement lca queries for both T and S_π in $O(1)$ time, which uses linear storage and preprocessing time.

Fix a tributary tree S of a heavy path of T . By storing S as a heavy-path tree, we can perform the following *tributary-sum* operation in $O(\log n)$ time: Given a node $\beta \in S$ and a height z such that $h(\beta) > z \geq h(\alpha_u)$, return the edge (w, w') of ρ_β with $h(w) > z \geq h(w')$ along with the sum of tributary volumes on the path from β to w in S .

The tributary-sum operation is implemented as follows. Suppose that, as we traverse the path ρ_β of S from α_u to β , we encounter the k heavy paths $(\pi_1, \pi_2, \dots, \pi_k)$ of S in order. In other words, the root α_u is the head of π_1 , and the query node β lies in π_k (refer to Figure 7 for an example). For each heavy path π_j , we say that its *lower height* x_j and *upper height* y_j are the heights of, respectively, the first and last tributary of π_j encountered along ρ_β . Formally, we define x_j to be the height of the head of π_j , and for each edge (π_j, π_{j+1}) for $1 \leq j < k$, corresponding to a light edge (v, v') connecting the two heavy paths, with v' being the head of π_{j+1} , we define y_j to be $h(v)$. Finally,

Fig. 7. Example of a β -tributary sum operation on a tributary tree S returning the desired edge (w, w') . Note that the root of S , having the lowest height, is on the bottom of the tree. (a) The tributary tree S with the light edges dotted, height threshold z marked with a horizontal line, and the path from $\beta \in \pi_4$ to the root highlighted. (b) The heavy-path tree representation of the tributary tree, with the path from the node π_4 containing β to the root marked.

we define y_k to be $h(\beta)$. As we traverse the heights in order $x_1 \leq y_1 < x_2 \leq y_2 < \dots < x_k \leq y_k$, there are two possible cases.

Case (i): If $x_j \leq z < y_j$, then (w, w') is a heavy edge in π_j , which can be found using binary search.

Case (ii): If $y_j \leq z < x_{j+1}$, then (w, w') is the light edge connecting π_j and π_{j+1} .

Once we have found the edge (w, w') , we compute the sum of tributary volumes from β to w in ρ_π as the difference between the suffix sums stored at β and w' .

We can find the heavy paths $(\pi_1, \pi_2, \dots, \pi_k)$ by storing with each node β the heavy path π_k that contains it. Then we follow parent pointers in the heavy-path tree to determine the $k = O(\log n)$ heavy paths. While we follow parent pointers to find the k heavy paths, we can determine the values x_j and y_j in constant time. Either we find out that (w, w') is a light edge, in which case we return it, or we find the heavy path π_j that contains the edge (w, w') , and take an additional $O(\log n)$ time to search for the edge. Finally, we can compute the sum of the volumes in constant time using the stored suffix sums. This gives an overall running time of $O(\log n)$.

3.3.2 Query Procedure for a Single Point Source. We now describe our procedure for computing $\sigma(p, q)$, the volume of the rain that must fall on a single vertex p before q gets flooded. First, we describe the case when the path in T from the root to the edge containing q consists solely of heavy edges, meaning that only a single heavy path π is involved when computing $\sigma(R, q)$. Then, we describe the general case when multiple heavy paths are involved in the query. The overall query time is $O(\log^2 n)$.

Single Heavy Path. If the path in T from the root to the edge containing q is contained in a single heavy path π and the rain region R consists of a single sink p , we start by computing the lca of p and q in T to find the tributary $\beta \in S_\pi$ containing p . Then, we perform a tributary-sum operation on S_π with β and $h(q)$ as input, resulting in an edge (w, w') with $h(w) > h(q) \geq h(w')$ and a sum V of tributary volumes from β to w . The query procedure returns $\sigma(p, q) = V + \mu(q)$, where $\mu(q)$ is computed by querying the merge tree as described in Appendix A.

Multiple Heavy Paths. Now suppose the path in T from the root to the edge containing q is split among $O(\log n)$ heavy paths. Let $A(p, q) = (\gamma_1, \dots, \gamma_\ell)$ be the sequence of tributaries between p and q . For each $i \leq \ell$, let v_i be the saddle that delimits γ_i . Then, v_1 is the lca of p and q in T , which can be computed in $O(1)$ time. Suppose q lies on the edge (u, v) of T . Let $(\pi_1, \dots, \pi_k)$ be the path in the heavy path tree P of T from the heavy path containing v_1 to the heavy path containing

Fig. 8. Given the heavy-path decomposition, $A(p, q) = (\beta_1, \beta_3, \beta_4, \beta_6, \beta_8)$ is partitioned into $A_1 = (\beta_1, \beta_3)$, $A_2 = (\beta_4)$, $A_3 = (\beta_6, \beta_8)$.

Fig. 9. The cases identified by our algorithm when processing a heavy path.

v , $k = O(\log n)$. Our query procedure performs a tributary-sum operation on each of the paths $\pi_1, \dots, \pi_k$ to determine the contribution of each heavy path to $\sigma(p, q)$, as follows.

The sequence of saddles $(v_1, \dots, v_\ell)$ delimiting the maximal depressions $A(p, q)$ form a subsequence of the path in T from the root to q , so we may partition $A(p, q)$ into contiguous subsequences $(A_1, \dots, A_k)$, each A_i consisting of the tributaries delimited by saddles in π_i (Figure 8). Fix $i \leq k$, and let $A_i = (\gamma_{r_i}, \gamma_{r_i+1}, \dots, \gamma_{r_{i+1}-1})$. We denote by $\sigma(A_i)$ the contribution of A_i to the sum (2), that is,

$$\sigma(A_i) = \sum_{j=r_i}^{r_{i+1}-1} V(\gamma_j).$$

Let $p(A_i)$ be the sink in γ_{r_i} on which water initially collects when γ_{r_i} starts to fill up. For the first heavy path π_1 , $p(A_1)$ is the source p , and for each subsequent path π_i , $p(A_i)$ is the spill sink of the predecessor of γ_{r_i} in $A(p, q)$. Our query algorithm computes the contribution $\sigma(A_i)$ using a tributary-sum operation on π_i for each $i \leq k$, and it also computes the starting sink $p(A_{i+1})$ for $i < k$. Finally, our query algorithm returns $\sigma(p, q)$ as $\sum_{i=1}^k \sigma(A_i) + \mu(q)$.

We now describe the procedure to compute $\sigma(A_i)$ and $p(A_{i+1})$ from $p(A_i)$. Fix $i \leq k$. For now, we assume $i < k$, which implies that q is contained in a tributary of π_i ; the last iteration $i = k$, when q lies on the heavy path π_i (Figure 9(d)), is handled in the same manner as the single heavy path case described previously. Let u be the sink at the tail of π_i , let v_p (resp. v_q) be the lowest ancestor

Fig. 10. Example of (B, z) -subtree operation on a tributary tree S . Note that the root of S , having the lowest height, is on the bottom of the tree. (a) The tributary tree S with the six nodes in B marked, height threshold z marked with a horizontal line, and each node labeled with its tributary volume. (b) G initialized as the union of ρ_β for $\beta \in B$. (c) G after removing all nodes $v \in G$ with $h(v) \leq z$. (d) The result of the (B, z) -subtree operation is a forest G consisting of nodes in B and nodes that are the lca of two nodes in B . Each node in G is labeled with its tributary-sum value.

of $p(A_i)$ (resp. q) in the merge tree which lies in π_i , and let α be the maximal depression delimited by v_q and containing u . We compute v_p (resp. v_q) in $O(1)$ time by using the data structure for constant time lca queries to find the lca between u and v_p (resp. v_q). There are three possible cases depending on the relative height of v_p and v_q .

- Case (i): If $h(v_p) = h(v_q)$, then A_i is empty, and $p(A_{i+1}) = p(A_i)$.
- Case (ii): If $h(v_p) < h(v_q)$, then A_i consists of the single maximal depression α (Figure 9(a)).
- Case (iii): If $h(v_p) > h(v_q)$, consider the tributaries $\gamma \in S_{\pi_i}$ along the path from v_p toward the sink u with $h(\gamma) \geq h(v_q)$. If v_q lies in this path, then the sequence of tributaries, excluding the last depression delimited by v_q , is equal to the sequence A_i (Figure 9(b)). Then, $\sigma(A_i)$ is the sum of tributary volumes from v_p to the predecessor of v_q along this path. If v_q does not lie on this path, this sequence of tributaries followed by the depression α is equal to A_i (Figure 9(c)). Then, $\sigma(A_i)$ is the sum of tributary volumes along the path, plus $V(\alpha)$. In both of these cases, $p(A_{i+1})$ is computed to be the spill sink of the last tributary in A_i , so $\sigma(A_i) = V(\alpha)$, and $p(A_{i+1})$ is the spill sink of α .

We visit $k = O(\log n)$ vertices of P and spend $O(\log n)$ time on each of them, so the total query time is $O(\log^2 n)$. We show below that the query time can be improved to $O(\log n)$.

3.3.3 Query Procedure for General Rain Regions. We now describe our procedure for computing $\sigma(R, q)$, for general rain regions R , with non-zero rain rate on m q -tributaries. As in the single point source case, we first describe the case when the path in T from the root to the edge containing q consists solely of heavy edges, and then give the query procedure when multiple heavy paths are involved.

Before we describe the query procedure, we need the following generalization of the tributary-sum operation on S_π . Given a height $z \geq h(\alpha_u)$ and a set of nodes $B \subseteq S_\pi$, $|B| = m$, such that all $\beta \in B$ have $h(\beta) > z$, the (B, z) -subtree operation returns a subtree G (actually, a forest) of S_π in time $O(m \log m)$, where G is defined by the following construction (refer to Figure 10 for an example):

- (1) G is initialized as the union of ρ_β for $\beta \in B$.
- (2) Remove all nodes $v \in G$ with $h(v) \leq z$.

- (3) Remove all nodes $v \in G$ that are not in B and have only a single child by splicing out v , that is, removing v and connecting its only child to the parent of v , if any.

The resulting forest G consists solely of nodes in B and nodes that are the lca of two nodes in B , so it has at most $2m - 1$ nodes. G can be constructed in $O(m \log m)$ time as follows. We sort B by increasing DFS discover time, resulting in a sequence $(\beta_1, \dots, \beta_m)$, and initialize G to the forest consisting of β_1 . For $i = 2$ to m , we compute $\alpha_i = \text{lca}(\beta_{i-1}, \beta_i)$ and add β_i to G according to the following two cases.

- Case (i): If $h(\alpha_i) \leq z$, then β_i is not connected to the previous tree, and we add β_i as a single node in G .
- Case (ii): If $h(\alpha_i) > z$, we follow parent pointers in G from β_{i-1} until we either find the edge $(\beta, \beta') \in G$ spanning α_i such that $h(\beta') < h(\alpha_i) < h(\beta)$, or find that α_i is lower than the lowest ancestor β' of β_{i-1} . We add the edge (α_i, β') and insert β_i as a child of α_i . Finally, if α_i was spanned by the edge (β, β') , we replace the edge (β, β') by (β, α_i) .

Case (i) takes $O(1)$ time. To see that Case (ii) takes amortized $O(1)$ time, observe that each edge is visited at most once after it is inserted into G . Thus, constructing G takes $O(m)$ time after sorting B in $O(m \log m)$ time.

Finally, the (B, z) -subtree operation augments each node $\beta \in G$ with a *tributary-sum value*, defined as follows (Figure 10(d)): If β has a parent β' in G , the tributary-sum value of β is the sum of tributary volumes on the path S_β up to but excluding β' . This value is computed by subtracting the suffix sum of β' from the suffix sum of β . If β does not have a parent in G , the tributary-sum value of β is the result of a tributary-sum operation in S_π on β and z . Computing each tributary-sum value takes $O(\log n)$ time, so the total time for the (B, z) -subtree operation is $O(m \log n)$.

Single Heavy Path. We assume that R consists of r sinks $p_1, \dots, p_r$; as described above, we can ensure this by spending $O(|R|)$ time to collect rain on sinks. We compute f_q , the fill-rate function of the depression delimited by q , as follows. First, we compute $B = \{\text{lca}(p_i, q) \mid i = 1, \dots, r\}$ by performing r lca operations on T and adding the R -rain rate on each p_i to $\text{lca}(p_i, q)$. As B consists of the q -tributaries at which rain is falling, $|B| = m$. For each tributary $\beta \in B$ such that $h(\beta) \leq h(q)$, β is contained in the depression delimited by q , so we remove β from B and add its initial R -rain rate to f_q . We then perform a $(B, h(q))$ -subtree operation on S_π resulting in a forest G . We set the initial fill-rate function f_β for each node $\beta \in B$ to the constant function equal to the R -rain rate in β ; the initial fill-rate f_β for each $\beta \in G \setminus B$ is simply set to the constant zero function. Then, we perform a postorder traversal of G in which we compute the spill-rate function g_β of each $\beta \in G$ with a TRUNCATE operation on the fill-rate function f_β with the tributary-sum value stored for β in G , and we then perform an ADD operation to add g_β to the fill-rate function of the parent of β in G , or to f_q if β has no parent in G . After the postorder traversal we have computed the fill-rate function f_q for the depression delimited by q , and the query procedure returns $\tau(f_q, \mu(q))$ as $\sigma(R, q)$. The total time is $O(|R| + m \log n)$.

Multiple Heavy Paths. We finally extend to the case in which multiple heavy paths are queried. As in the single heavy path case, we begin by performing r lca operations on T to compute the set B of m tributaries on which rain is initially falling, $B = \{\text{lca}(p_i, q) \mid i = 1, \dots, r\}$. We add the R -rain rate on each p_i to $\text{lca}(p_i, q)$; if B contains any tributaries β with $h(\beta) \leq h(q)$, we remove them from B and add the sum of their initial R -rain rates to f_q , the fill-rate function of the depression delimited by q . Then, we set the initial fill-rate function f_β for each node $\beta \in B$ to the constant function equal to the R -rain rate in β .

Fig. 11. Illustration of S_q , the tributary tree for q , consisting of 20 nodes split into four sets corresponding to the four heavy paths π_1 – π_4 . (a) B consists of seven tributaries (shaded nodes) that have a positive R -rain rate. (b) The initial set of fill-rate functions F_1 consists of seven constant functions corresponding to the seven tributaries in R . (c) Fill-rate functions in F_2 after processing the first heavy path π_1 . (d) Fill-rate functions in F_3 after processing π_2 . (e) Fill-rate functions in F_4 after processing π_3 . (f) Fill-rate function for q after processing the last heavy path π_4 .

For each $i \leq k + 1$, let F_i be the set of fill-rate functions describing the rate at which the depression delimited by q and the tributaries in $\pi_i, \dots, \pi_k$ fill up when taking only initial R -fill rate and spilling from $\pi_1, \dots, \pi_{i-1}$ into account. Refer to Figure 11. Thus, F_1 consists of the initial fill-rate functions computed for B (Figure 11(b)), and F_{k+1} consists of the final fill-rate function f_q (Figure 11(f)). We compute F_{i+1} from F_i as follows. Fix $i \leq k$. Let B' be the set of q -tributaries that have non-zero fill-rate functions in F_i , let v_q be the first ancestor of q in T that lies in π_i , and let α be the q -tributary delimited by v_q . We perform a $(B', h(v_q))$ -subtree operation on π_i to compute a forest G and perform a postorder traversal of G . When processing a node $\beta \in G$ in the postorder traversal, we compute the spill-rate function g_β with a TRUNCATE operation on the fill-rate function f_β with the tributary-sum value stored for β in G . Then, if β has a parent in G , we perform an ADD operation to add g_β to the fill-rate function of the parent of β in G . Otherwise, β has no parent in G , and we compute the lca (in T) of q and the spill sink of β to find the q -tributary β' to which β spills. β' is either the depression delimited by q , in which case we add g_β to f_q , or β' is α or a q -tributary in one of $\pi_{i+1}, \dots, \pi_k$, in which case, we add g_β to $f_{\beta'}$. After processing G , we conclude the processing of π_i by computing g_α from f_α and adding g_α to the q -tributary α spills, too. Once we have processed the last heavy path π_k , we have computed the fill-rate function f_q for the depression delimited by q , and the query procedure returns $\tau(f_q, \mu(q))$ as $\sigma(R, q)$.

The time spent in each heavy path π_i is $O(m \log n)$, so the total time for the query procedure is $O(|R| + m \log^2 n)$.

3.3.4 Improving the Query Time to $O(|R| + m \log n)$. The query procedure described above spends $O(m \log n)$ time in each heavy path:

- Computing v_q takes $O(1)$ time to find the lca between q and the root of the heavy path.
- Computing the (B, z) -subtree query requires
 - $O(m)$ lca queries, each of which takes $O(1)$ time using the fast data structure
 - $O(m \log m)$ time to construct the forest G
 - $O(m)$ tributary-sum queries in S_{π_i} , each of which takes $O(\log n)$ time using the heavy-path tree structure to implement the (B, z) -subtree query.

We first describe how to improve the time for computing each tributary-sum query to $O(\log(n_i/n_{i+1}))$, where n_i is the size of the subtree of T rooted at the head of π_i . Then we describe how each forest G can be constructed in time $O(m \log(n_i/n_{i+1}))$, plus an additional cost of $O(m \log m)$ amortized over all the heavy paths.

We wish to modify the tributary-sum operation so that it takes time $O(\log(n_i/n_{i+1}))$ where n_{i+1} is the size of the sub-problem corresponding to the edge returned. For this, we store the edges of S_π in a *persistent biased search tree*, which we define as follows. A *biased search tree* is a binary search tree where each node has a weight in addition to its search key. Bent, Sleator, and Tarjan [8] showed how to maintain a biased search tree with updates in $O(\log n)$ time and searches in time $O(\log(n/w))$, where n is the total weight of items in the search tree and w is the weight of the returned item. A *persistent biased search tree* is a biased search tree in which any previous version may be searched. Since the in-degree and out-degree of nodes in a biased search tree have bounded degree, the biased search tree may be made persistent without affecting the asymptotic query and update times, with the total space usage of the persistent structure being proportional to the number of updates to the structure [9].

Given a heavy path $\pi = (v_1, \dots, v_\ell, u)$ in P , for each $i \leq \ell$, let β_i be the u -tributary delimited by v_i and $|\beta_i|$ denote the number of merge tree nodes contained in the depression. We store the edges of the tributary tree S_π of π in a persistent biased search tree, where the *search key* of (v_i, v_j) is $h(v_j)$ and the *weight* of (v_i, v_j) is $\max_{i < k \leq j} |\beta_k|$. Besides the search key and weight, the edge (v_i, v_j) also stores the suffix sum of v_j , that is, the sum of tributary volumes from v_j to u in S_π .

Starting with the empty search tree, we perform a DFS traversal of S_π in which we insert an edge into the search tree when we first encounter it (on the way down), and delete the edge from the search tree when we encounter it again (on the way up). The first time we visit a node β , we store a pointer to the current version of the biased search tree, which will at this time contain exactly the edges in ρ_β , the path from β to the root α_u in S_π . At any point in the DFS traversal of S_π , the weights stored in the biased search tree correspond to the sizes of disjoint sub-problems, so the total weight is at most the sub-problem size of π . When an edge (v_i, v_j) is inserted into the biased search tree, the suffix sum of v_j is computed by adding the tributary volume of v_j to the suffix sum stored with the successor of (v_i, v_j) in S_π , which can be found in $O(\log |\pi|)$ time, where $|\pi|$ is the number of edges in π . We note that the insertion or deletion of an edge of S_π does not affect the suffix sum stored at any other node of the biased search tree. Therefore, insertion and deletion of an edge in the biased search tree requires only amortized $O(1)$ changes to ranks and pointers stored in the tree. Each edge of S_π is inserted and deleted once. All in all, $O(|\pi|)$ information is changed in the biased search tree during the DFS traversal of S_π , so constructing the persistent biased search tree takes $O(|\pi| \log |\pi|)$ time and uses $O(|\pi|)$ space. Thus, this representation does not increase the asymptotic preprocessing time or space usage of the overall data structure.

Now, to perform a tributary-sum operation, given a node β and height z , we use the biased search tree stored at β to search for the node with the largest key no larger than z . This corresponds to the desired edge (v_r, v_s) , such that $h(v_r) > z \geq h(v_s)$. When we perform a tributary-sum operation

on the i -th heavy path π_i , the height for the query will be $h(v_q)$, recalling that v_q is the lowest ancestor of q which lies in π_i , and we recurse into the tributary β_q . For the edge (v_r, v_s) returned by the tributary-sum operation, we have that $\beta_q \in \{\beta_{r+1}, \dots, \beta_s\}$. This implies that $|\beta_q|$, the size of the next sub-problem, is at most the weight of the edge (v_r, v_s) . Thus, the time spent on the tributary-sum operation is $O(\log(n_i/n_{i+1}))$, as required.

To compute the forest G in one of the heavy paths for a set of nodes B' , we first perform a tributary-sum operation for each $\beta \in B'$, and store each β with the edge it returns. Two nodes β, β' belong to the same tree of the forest G if and only if the tributary-sum operations return the same edge. By identifying each node of B' with the edge returned by the tributary-sum operation, we group the nodes of B' into trees of G . We then proceed with constructing each tree as described previously by sorting the tree nodes by DFS discover time and performing lca queries. Constructing a tree of k nodes requires $O(k \log k)$ time. The forest G is then computed as the union of the constructed trees. The following charging argument shows that the total time to sort all trees in all heavy paths is $O(m \log m)$. After processing a tree in G consisting of $k > 1$ nodes, the total number of tributaries with non-zero fill-rate functions is reduced by $k - 1$, as the k nodes eventually spill to the same tributary of S_π , namely that which is the lca of the nodes. Picking an arbitrary representative of the k nodes, we say that the other $k - 1$ tributaries merge into the representative. We charge the $O(k \log k)$ time to sort the nodes in the tree to the $k - 1$ tributaries that are merged into the representative. Since every tributary is merged at most once, each of the initial m tributaries is charged at most $O(\log m)$ time, and the total time to sort all trees is $O(m \log m)$. Therefore, the total time computing the (B, z) -subtree queries will be at most $O(m \log m)$.

THEOREM 3.2. *Given a triangulation $\mathbb{M}$ of $\mathbb{R}^2$ with vertex set $\mathbb{V}$ of size n and a height function $h : \mathbb{M} \rightarrow \mathbb{R}$ that is linear on each face of $\mathbb{M}$, a data structure of size $O(n)$ can be constructed in time $O(n \log n)$ that for any rain region R and point $q \in \mathbb{R}^2$ in $O(|R| + m \log n)$ time, returns $\sigma(R, q)$, where $|R|$ is the number of vertices with positive rainfall.*

4 #P-HARDNESS OF COMPUTING FLOODING PROBABILITY

In this section, we show that computing the exact flooding probability on uncertain terrains is #P-hard, which motivates the use of an approximation algorithm in the next section.

Let H be a distribution on height functions of description complexity k . Given a rain region R , volume $\psi \geq 0$, and point $q \in \mathbb{M}$, the probability that q is flooded is equal to the probability of sampling a height function h from the distribution H for which $\sigma_h(R, q) \leq \psi$. The only source of uncertainty is the terrain; the rain region R and volume ψ are not random in our model. To be precise, we define the probability of flooding as

$$\pi(R, q, \psi) = \Pr_{h \sim H} [h \in H_{R, q, \psi}],$$

where $H_{R, q, \psi}$ is the set $\{h \in H \mid \sigma_h(R, q) \leq \psi\}$.

We show the hardness of computing $\pi(R, q, \psi)$ by reduction from #KNAPSACK, a well-known #P-hard problem. An instance of the #KNAPSACK problem consists of a set of n items $X = \{x_1, \dots, x_n\}$ with corresponding weights $w_1, \dots, w_n$ and capacity W . We say a subset $S \subset X$ is *feasible* if

$$\sum_{x_i \in S} w_i \leq W$$

and the solution to the #KNAPSACK instance is the number of feasible subsets.

Given an instance of the #KNAPSACK problem, we construct a planar triangulation $\mathbb{M}$ and distribution on height functions H as follows, referring to Figure 12. The exact details of $\mathbb{M}$ do not matter, so we only focus on the depressions and the heights of their saddle points. While,

Fig. 12. The construction of an uncertain terrain $\mathbb{M}$ to solve an instance of #KNAPSACK.

in general, some vertices in a terrain may be a saddle point for one height function, but not for another, we assume that our terrain is constructed such that the saddle points are saddle points over the distribution of height functions, and the saddles are the only vertices that have uncertainty on their height. We form one central depression β_0 surrounded by $n + 1$ depressions $\beta_1, \dots, \beta_{n+1}$. Let v_i denote the saddle point between β_0 and β_i , and p_i denote the sink of the depression β_i . We set the height of v_{n+1} to 1 with probability 1. For every other saddle point v_i , its height is $1 - \varepsilon$ with probability $1/2$ for some sufficiently small $\varepsilon > 0$, and height 2 with probability $1/2$. For simplicity, our construction uses a degenerate case where the height of many vertices can be the same, but one can fix this by perturbing the heights by some small amount. We assume that the height of each saddle point is chosen independently, so each of the 2^n possible assignments are equally likely in our distribution $\mathcal{H}$. Further, we construct each depression β_i so that it has volume w_i when the water is at height 1, with β_0 having volume 1.

We prove the following lemma that relates the solution to the #KNAPSACK problem with a flooding probability on the uncertain terrain.

LEMMA 4.1. *Given an instance of #KNAPSACK with k feasible subsets, for the uncertain terrain $(\mathbb{M}, \mathcal{H})$ constructed as above, letting p be the sink of β_0 , and q be the sink of β_{n+1} , we have*

$$\pi(p, p_{n+1}, W + 1) = \frac{k}{2^n}.$$

PROOF. Given a subset $S \subset X$, let h_S be the height function that assigns a height of $1 - \varepsilon$ to each saddle point corresponding to an element $x \in S$, and assigns a height of 2 to each saddle point corresponding to $x \in \bar{S}$. We note that there is a bijection between the possible height functions and subsets of S .

When rain falls on p , depression β_0 begins filling until the water reaches a height of $1 - \varepsilon$. At this point, water begins spilling into the depressions whose saddle points are lower, namely those corresponding to the set S . Once these all become filled, the water level finally rises to height 1, spilling into depression β_{n+1} , and q will be flooded. The total volume of rain at this point will be the volume of each of the depressions plus the volume of β_0 , which is 1, so we have that

$$\sigma_{h_S}(p, q) = 1 + \sum_{x_i \in S} w_i.$$

Thus, a height function satisfies $\sigma_{h_S}(p, q) \leq W + 1$ if and only if S is a feasible subset in the corresponding #KNAPSACK problem. Since each height function is chosen with probability $\frac{1}{2^n}$, the flooding probability is then $\frac{k}{2^n}$. $\square$

THEOREM 4.2. *Computing the flooding probability $\pi(R, q, \psi)$ on an uncertain terrain is #P-hard.*

5 MONTE CARLO ALGORITHM

In this section, we present a data structure to estimate the probability of flooding on an uncertain terrain. Our data structure to estimate $\pi(R, q, \psi)$ works as follows. During preprocessing, we sample s height functions $H = \{h_1, \dots, h_s\}$ i.i.d. from H , with s to be specified later. For each $i \leq s$, we construct in $O(n \log n)$ time the data structure $\mathcal{D}_i$, of size $O(n)$, described in Section 3.3, on h_i . The size of our data structure is $O(sn)$, and the preprocessing time is $O(sn \log n)$.

Given a query (R, q, ψ) , we query each $\mathcal{D}_i$ to determine the number c of height functions $h_i \in H$ for which $\sigma_{h_i}(R, q) \leq \psi$. We return c/s as the probability estimate $\hat{\pi}(R, q, \psi)$. The query time is $O(s|R| + M \log^2 n)$, where $M = \sum m_i$, and m_i is the number of sinks in R under h_i .

Now we bound s , the number of samples in our Monte Carlo scheme, to ensure that the approximation error does not exceed ε on any query with probability at least $1 - \delta$ for fixed parameters $\varepsilon, \delta \in (0, 1)$. First, we analyze the rain-volume function σ_h to show that it has a nice structure. Exploiting this structure, we apply ideas from statistical learning to bound the value of s .

Rain-volume Function. Recall that the ℓ -level set of h is the set $h_\ell = \{x \in \mathbb{R}^2 \mid h(x) = \ell\}$. We define the *height-level map* of h , denoted by M_h , to be the planar subdivision induced by $h_{=h(v)}$ for all $v \in \mathbb{V}$. Note that M_h can have $\Theta(n^2)$ vertices in the worst case. We fix the distribution H on the height functions, as defined in Section 2. M_H is the distribution of height-level maps induced by H . Let Ξ denote the triangulation of the overlay of $\mathbb{M}$ with all the height-level maps in M_H . Agarwal et al. [2] showed that Ξ has complexity $O(n^3 k^8)$ despite being the overlay of k^n different height-level maps in the worst case.

LEMMA 5.1. *The following properties hold for any $h \in H$ and for any face ϕ (of any dimension) of Ξ :*

- (i) *A level-set through a vertex of Σ is either disjoint from ϕ or contains all of ϕ (in which case ϕ is zero- or one-dimensional).*
- (ii) *For any two points $x, x' \in \phi$, the depression delimited by x contains the same set of sinks as the depression delimited by x' .*
- (iii) *For any fixed $p \in \mathbb{V}$ and for any $x, x' \in \phi$, $A(p, x) = A(p, x')$, where $A(p, x)$ is the set of tributaries between p and x .*

PROOF. (i) follows from the construction. (ii) follows from the fact that x and x' belong to the same edge of T_h . (iii) follows from the remark in Section 3.1, since x and x' belong to the same edge of T_h . $\square$

LEMMA 5.2. *For any height function $h \in H$ and any face ϕ (of any dimension) of Ξ , the depression volume function is a bivariate cubic function denoted by μ_h^ϕ , that is, for all $x \in \phi$,*

$$\mu_h(x) = \mu_h^\phi(x).$$

PROOF. We first split each triangle of $\mathbb{M}$ into two triangles as follows. For each triangle of $\mathbb{M}$ with vertices $A, B, C \in \mathbb{V}$ such that $h(A) < h(B) < h(C)$, let D be the point on AC such that $h(D) = h(B)$. We split the face by the line BD into a bottom half ABD and a top half BCD . The ℓ -level set, for $h(A) \leq \ell \leq h(C)$, intersects the face ABC in a line segment parallel to BD , which implies that no level set intersects both the bottom half and the top half of a face ABC .

Fig. 13. Example triangulation with triangle $\phi \in \Xi$, triangles δ_1 – $\delta_3 \in \Delta$, and the lowest point x_1 and highest point x_3 in ϕ . δ_1 is contained in β_{x_1} , δ_3 is disjoint from β_{x_3} , and δ_2 intersects the boundary of β_x for all $x \in \phi$.

Fig. 14. Measuring depression volume above a triangle ABC of Σ . (a) The triangle is completely contained in the depression. (b) The triangle with upper flat edge is intersected by the contour DE . (c) The triangle with lower flat edge is intersected by the contour DE .

Let Δ denote this resulting set of triangles in $\mathbb{M}$.

$$\mu_h(x) = \sum_{\delta \in \Delta} \int_{\delta \cap \beta_x} (h(x) - h(y)) dy. \quad (3)$$

Fix $\phi \in \Xi$. By Lemma 5.1(i), there are two cases. If ϕ is contained in a level-set through a vertex of Σ , then h is constant on ϕ and every point $x \in \phi$ delimits the same depression. This implies that $\mu_h(x)$ is a constant function on ϕ .

Otherwise, ϕ is disjoint from all level-sets through vertices of Σ . For now, we assume that ϕ is two-dimensional, that is, a triangle; our proof can easily be simplified to handle the case when ϕ is zero- or one-dimensional. Let x_1, x_2 , and x_3 be the vertices of ϕ such that $h(x_1) \leq h(x_2) \leq h(x_3)$ (Figure 13). For all $x \in \phi$, $\beta_{x_1} \subseteq \beta_x \subseteq \beta_{x_3}$, where β_{x_1}, β_x , and β_{x_3} are the depressions delimited by, respectively, x_1, x , and x_3 . By construction, the boundary of β_x crosses the same set of edges of $\mathbb{M}$ as β_{x_1} . Using this property, we argue that each term in the sum of Equation (3) is a cubic function of $x \in \phi$, from which the claim follows.

Fix a triangle $\delta \in \Delta$. Since no level-set through a vertex of δ intersects ϕ , δ is either contained in β_{x_1} , disjoint from β_{x_3} , or δ intersects the boundary of β_x for all $x \in \phi$. Thus, the term of μ_h^ϕ corresponding to the half-face δ falls into one of three cases.

Case (i). If δ is disjoint from β_{x_3} , then the term is an empty integral, which is zero.

Case (ii). If $\delta \subseteq \beta_{x_1}$ (refer to Figure 14(a)), then $\delta \cap \beta_x = \delta$ for all $x \in \phi$. The portion of β_x over δ is a triangular prism whose top face is a copy of δ at height $h(x)$ and the bottom face is the triangle of Σ corresponding to δ . Let h_δ be the height of the highest

vertex of δ . Then, the volume of this prism is a constant plus $\text{Area}(\delta) \cdot (h(x) - h_\delta)$, and thus a linear function.

Case (iii). If for all $x \in \phi$, $\delta \cap \beta_x \subset \delta$, meaning the contour through x intersects δ , then we argue that the integral is a cubic function. Let A, B, C be the vertices of δ such that either $h(A) < h(B) = h(C)$ (Figure 14(b)) or $h(A) = h(B) < h(C)$ (Figure 14(c)). The integral over $\delta \cap \beta_x$ equals the volume of the frustum Ω that lies between the plane $z = h(A)$ and the plane $z = h(x)$. Each vertex of Ω can be written as a linear function of $h(x)$ with coefficients depending on A, B , and C . By tetrahedralizing Ω , we can write its volume as the sum of volumes of tetrahedra. Since the volume of each tetrahedron is a cubic function of $h(x)$, it follows that the integral over $\delta \cap \beta_x$ is a cubic function of $h(x)$.

Finally, $h(x)$ is a linear function of x for $x \in \phi$. Hence, $\mu_h(x)$ is a bivariate cubic function of x . $\square$

LEMMA 5.3. *Let h be a height function and p a vertex of $\mathbb{M}$. Restricting the query point q to a face ϕ of Ξ , the rain-volume function is a univariate cubic function of height, denoted by $s_{h,p,\phi}$, that is, for all $q \in \phi$,*

$$\sigma_h(p, q) = s_{h,p,\phi}(h(q)).$$

PROOF. Fix h, p, ϕ , and consider points in ϕ . By Lemma 5.2, for all points $x \in \phi$, $\mu_h(x) = v_h^\phi(h(x))$, where v_h^ϕ is a univariate cubic function. By Lemma 5.1(iii), $A(p, x)$ is the same for all $x \in \phi$, and, furthermore, the volume of each tributary in $A(p, x)$ is independent of x . Therefore, the restriction of $\sigma_h(p, x)$ over ϕ is the cubic function

$$\sigma_{h,p,\phi}(x) = v_h^\phi(h(x)) + \alpha,$$

where α is a constant that depends on h, p , and ϕ . $\square$

Since $h(x)$ is a linear function of x for $x \in \phi$, we immediately obtain the following corollary.

COROLLARY 5.4. *Let h be a height function and p a vertex of $\mathbb{M}$. Restricting the query point q to a face ϕ of Ξ , the rain-volume function is a bivariate cubic function of q denoted by $\sigma_{h,p,\phi}$, that is, for all $q \in \phi$,*

$$\sigma_h(p, q) = \sigma_{h,p,\phi}(q).$$

COROLLARY 5.5. *For a fixed rain region R , for any height function $h \in \mathcal{H}$, and for any face $\phi \in \Xi$, the rain-volume function is a piecewise cubic function of height denoted by $\sigma_{h,R,\phi}$ with number of pieces being bounded by $|R|$.*

PROOF. By Lemma 5.1(ii), β_x has the same set of sinks for all $x \in \phi$. As long as the water from the same set of vertices of R is reaching these sinks, by Lemma 5.3, the rain-volume function is a cubic function of height. As more rain falls, water from more source points will reach these sinks, so the rain-volume function is piecewise cubic with the number of pieces being bounded by $|R|$. $\square$

Point-source Query. We now bound the number of samples needed in the case when the source is a single vertex $p \in \mathbb{V}$. For a point $q \in \mathbb{R}^2$ and for a volume $\psi \geq 0$, we use $\pi(p, q, \psi)$ to denote the flooding probability of q if ψ units of rain fall at p , and $\hat{\pi}(p, q, \psi)$ to denote the estimate of $\pi(p, q, \psi)$ returned by our data structure. Let

$$\text{err}(p, q, \psi) = |\pi(p, q, \psi) - \hat{\pi}(p, q, \psi)|.$$

For a face $\phi \in \Xi$ and a vertex $p \in \mathbb{V}$, we define

$$\text{err}(p, \phi) = \sup_{\substack{q \in \phi \\ \psi \geq 0}} \text{err}(p, q, \psi).$$

Finally, let err be the maximum value of $\text{err}(p, \phi)$ over all vertices $p \in \mathbb{V}$ and $\phi \in \Xi$.

First, we bound the probability of the event that $\text{err}(p, \phi) > \varepsilon$. We use the so-called *Veronese map* [19] to map a bivariate cubic function to a linear function with nine variables. In particular, each monomial $x^i y^j$ is mapped to a new variable z_{ij} for $1 \leq i + j \leq 3$, $i, j \geq 0$. That is, we define the map $v : \mathbb{R}^2 \rightarrow \mathbb{R}^9$ as

$$v(x, y) = \langle z_{ij} \mid i, j \geq 0, 1 \leq i + j \leq 3 \rangle.$$

Now, a cubic function

$$f(x, y) = \sum_{i=0}^3 \sum_{j=0}^{3-i} a_{ij} x^i y^j$$

is mapped to the linear function

$$\tilde{f}(z_{10}, \dots, z_{03}) = a_{00} + \sum_{1 \leq i+j \leq 3} a_{ij} z_{ij}.$$

LEMMA 5.6. *For any given $p \in \mathbb{V}$ and face ϕ of Ξ ,*

$$\Pr[\text{err}(p, \phi) > \varepsilon] = \exp(O(-s\varepsilon^2)). \quad (4)$$

PROOF. Fix a vertex $p \in \mathbb{V}$ and a face ϕ of Ξ . We consider the set system $(H, \Omega_{p, \phi})$, where

$$\Omega_{p, \phi} = \{H_{p, q, \psi} \mid q \in \phi, \psi \geq 0\},$$

and $H_{p, q, \psi}$ is the set $\{h \in H \mid \sigma_h(p, q) \leq \psi\}$. By Corollary 5.4, for any bivariate height function $h \in H$ and for a fixed source $p \in \mathbb{V}$, $\sigma_h(p, q)$ restricted to ϕ is a bivariate cubic function $\sigma_h^\phi(q)$. Let $\tilde{\sigma}_h^\phi(v(q))$ denote the linearization of $\sigma_h^\phi(q)$ as defined above. We now define a new set system (S, R) , where $S = \{\tilde{\sigma}_h^\phi \mid h \in H\}$ and $R = \{\{\tilde{\sigma}_h^\phi \mid \tilde{\sigma}_h^\phi(v(q)) \leq \psi\} \mid q \in \phi, \psi \geq 0\}$. Note that S is a set of linear functions, so a classical result in random sampling by Vapnik and Chervonenkis [29] (see also Ref. [17]) implies for a random sample of size $O(\frac{1}{\varepsilon^2} \log \frac{1}{\delta})$, $\text{err}(p, \phi) \leq \varepsilon$ with probability at least $1 - \delta$. This immediately implies the lemma. $\square$

Since $|\Xi| = (nk)^{O(1)}$, using a union bound, we obtain the following:

THEOREM 5.7. *Let $(\mathbb{M}, H)$ be an uncertain terrain with n vertices and description complexity k , and let $\varepsilon, \delta \in (0, 1)$. If the number of samples is $s = \Omega(\frac{1}{\varepsilon^2} \lg \frac{nk}{\delta})$, then the greatest approximation error of our data structure over all queries where R consists of a single vertex is at most ε with probability at least $1 - \delta$.*

Fixed Region Query. We now extend the above analysis to the case when the source is a fixed region R . For a point $q \in \mathbb{R}^2$ and for a volume $\psi \geq 0$, we use $\pi(R, q, \psi)$ to denote the flooding probability of q if ψ units of rain falls at R , and $\hat{\pi}(R, q, \psi)$ to denote the estimate of $\pi(R, q, \psi)$ returned by our data structure. Let

$$\text{err}(R, q, \psi) = |\pi(R, q, \psi) - \hat{\pi}(R, q, \psi)|.$$

For a face $\phi \in \Xi$, we define

$$\text{err}(R, \phi) = \sup_{\substack{q \in \phi \\ \psi \geq 0}} \text{err}(R, q, \psi).$$

Finally, let $\text{err}(R)$ be the maximum value of $\text{err}(R, \phi)$ over all $\phi \in \Xi$.

Let H_ϕ^* denote the set of all possible height functions for ϕ . Since ϕ is contained inside a triangle of $\mathbb{M}$, H_ϕ^* is a set of k^3 height functions $\{h_1^*, \dots, h_{k^3}^*\}$. For each $i \leq k^3$ we let H_i be the set of height functions equal to h_i^* on ϕ .

LEMMA 5.8. *For any fixed rain region R and any face ϕ of Ξ ,*

$$\Pr[\text{err}(R, \phi) > \varepsilon] = \exp(O(-s\varepsilon^2 k^3 \log n)). \quad (5)$$

PROOF. Fix a face ϕ of Ξ . We consider the set system (H, Ω_ϕ) , where

$$\Omega_\phi = \{H_{R,q,\psi} \mid q \in \phi, \psi \geq 0\}.$$

By Corollary 5.5, for any bivariate height function $h \in H$, $\sigma_h(R, q)$, restricted to ϕ is a piecewise cubic function $\sigma_h^\phi(q) = s_h^\phi(h_i^*(q))$ with at most n pieces, where $h_i^* \in H_\phi$ is the restriction of h to ϕ . For each $i \leq k^3$, we define a set system (S_i, R_i) where $S_i = \{s_h^\phi \mid h \in H_i\}$ and $R_i = \{\{s_h^\phi \in S_i \mid s_h^\phi(h_i^*(q)) \leq \psi\} \mid q \in \phi, \psi \geq 0\}$. We now define a new set system (S, R) , where $S = \bigcup_{i=1}^{k^3} S_i$ and $R = \bigcup_{i=1}^{k^3} R_i$. Note that each S_i is a set of functions that intersect pair-wise at most $6n$ times, so the analysis by Grünbaum [16] and Har-Peled [17] implies for a random sample of size $O(\frac{k^3 \log n}{\varepsilon^2} \log \frac{1}{\delta})$, $\text{err}(R, \phi) \leq \varepsilon$ with probability at least $1 - \delta$. $\square$

Since $|\Xi| = (nk)^{O(1)}$, using a union bound, we obtain the following:

THEOREM 5.9. *Let $(\mathbb{M}, H)$ be an uncertain terrain with n vertices and description complexity k , let R be a rain region, and let $\varepsilon, \delta \in (0, 1)$. If the number of samples is $s = \Omega(\frac{k^3 \log n}{\varepsilon^2} \lg(\frac{nk}{\delta}))$, then the greatest approximation error of our data structure over all queries with fixed rain region R is at most ε with probability at least $1 - \delta$.*

6 EXPERIMENTS

In this section, we present the experiments we have conducted with two real datasets to demonstrate the efficiency and usefulness of our method. Our experiments show that only a few samples are needed in practice, and by inspecting the output, we learn something interesting about the flood risk in an uncertain terrain.

We have implemented the simple data structure in Section 3.2 in C++, and we use a computer with a 3.5GHz Intel Xeon E5-1650 CPU, 128GB RAM, running Linux 4.4, for the experiments.

We did not conduct experiments with the fast data structure, as our experiments are about uncertainty, not speed.

Terrains. We study the performance of our algorithms on two terrain datasets. The *Indiana dataset* is a 0.4km² model of an area 1km northeast of Holland, Indiana, extracted from the publicly available 5ft resolution DEM of Indiana [20]. The *Denmark dataset* is a 33km² model of the city of Vejle, Denmark, extracted from the publicly available 1.6m resolution DEM of Denmark [13]. Both are high-resolution grid datasets containing respectively 190,000 and 13,000,000 vertices, and, for efficiency, we use an implementation of our algorithm tuned to grid datasets rather than general TINs (triangulated irregular networks). The datasets do not have uncertainty on the vertex heights, and we introduce uncertainty below.

Uncertainty. For the Indiana dataset, we introduce uncertainty on the vertex heights by choosing parameters k and Δ and generating data as follows. For each vertex v , we draw k numbers H_v^k from the uniform distribution $U(h(v) - \Delta/2, h(v) + \Delta/2)$. To instantiate a random terrain, we pick the height of a vertex v uniformly at random from H_v^k . In this uncertain terrain, vertex heights are independent. The vertex heights on the base dataset are between 150m and 170m above sea-level, and the median height distance between a vertex and its lowest neighbor is 12cm. Accordingly,

we used values of Δ between 2cm and 80cm in our experiments. As for the parameter k , we chose values from 2 up to 64; as discussed in the introduction, when terrain data uncertainty stems from multiple height values measured during raw data acquisition, we expect that the value of k is a small constant, smaller than 64 in practice.

The source of uncertainty in the Denmark dataset is the set of publicly available so-called *hydrological corrections* for the terrain model [13]. Each hydrological correction is a polygon with an associated height function and represents an underground pipe, a waterway under a bridge, or a similar terrain feature that permits water flow under the surface represented by the terrain model. Of the 114,000 corrections in the dataset for all of Denmark, 220 corrections are inside the Vejle region we use in our experiments. We assign existential probabilities to each of the 220 corrections by taking independent samples from a normal distribution with mean $\mu = 0.5$ and standard deviation $\sigma = 0.05$. To instantiate a random terrain, we update the base terrain model with the height functions of a random subset of the corrections, where each correction is included independently according to its probability. The vertex heights in this uncertain terrain are *not* independent, since the vertices contained in the same hydrological correction are all updated to the height function of the correction if the correction is included in the instantiated terrain.

Queries. We considered the following queries (R, q, ψ) to our data structure. For the rain region R , we considered convex regions with different sizes from the smallest region consisting of a single vertex, to the largest region consisting of all terrain vertices, and we set the same rain rate on all vertices on which rain is falling. For each dataset, we selected regions with large downstreams in the terrain to ensure that flooding queries would exercise flooding of areas far away from the rain region. Except where noted, our experiments use a specific small rain region of 4,000m² for the Indiana model and 0.24km² for the Denmark model. For the point q , we query all vertices of the terrain. We selected rain volumes ψ corresponding to between 1cm and 80cm of rain on the rain region. As an example of the output of our algorithm, Figure 1(b) shows a 1.7km $\times$ 0.7km portion of the output when 800m³ of rain falls on the marked point p of the Denmark dataset, and Figure 19 shows a 3.1km $\times$ 1.2km portion of the output when 40cm rain falls on the region R of the Denmark dataset.

Measuring the Convergence. For each query (R, q, ψ) and round j , we compute $|\hat{\pi}_j(R, q, \psi) - \pi(R, q, \psi)|$, where $\hat{\pi}_j(R, q, \psi)$ denotes the estimate of $\pi(R, q, \psi)$ computed on the first j instances of the uncertain terrain. Over all query points $q \in \mathbb{V}$, we report the 50th, 80th, 95th, or 100th percentile of these probability differences. Note that if $\pi(R, q, \psi) = 0$, then $\hat{\pi}(R, q, \psi) = 0$ and the error is 0; the error is also 0 when $\pi(R, q, \psi) = 1$. We discard such queries when we measure the errors; including them would only reduce the error. Figure 15(a)–(b) illustrates that the error decreases quickly, with 50 samples yielding a reasonably small error.

Since computing $\pi(R, q, \psi)$ is hard, we instead apply our Monte Carlo method with 3,000 samples and use the resulting estimates as the exact value of $\pi(R, q, \psi)$.

Size of Region. First we measured the convergence as we vary the size of the rain region. Figure 15(c) illustrates that the error decreases quickly for both the single-point case, the small region case, and the case when rain is falling on the entire Indiana terrain. The error is greatest when rain is falling on the entire terrain, which agrees with our theoretical analysis.

Amount of Rain. We measured the convergence as we vary ψ , the amount of rain. Figure 15(d) illustrates that the error increases as the amount of rain increases. This is to be expected since the number of vertices with a non-zero flood risk increases as one increases the amount of rain.

Description Complexity k . We considered varying the description complexity k , that is, the number of possible heights in each vertex on the Indiana model. We fix $\Delta = 40$ cm and choose

(a) Denmark.

(b) Indiana, $\Delta = 40$ cm, $k = 8$.

(c) Varying size of region.

(d) Varying amount of rain.

Fig. 15. (a–b) Error when 40cm rain falls on the rain region. (c) Max error when varying the size of the Indiana region. (d) 95th percentile error on the Denmark medium-sized region when the amount of rain varies.

(a) 80-th percentile.

(b) 95-th percentile.

Fig. 16. Varying k on Indiana with $\Delta = 40$ cm, 40cm rain.

k from $\{2, 8, 32, 64\}$. For each vertex v , we draw random sets $H_v^2 \subset H_v^8 \subset H_v^{32} \subset H_v^{64}$ from the uniform distribution $U_v = U(h(v) - \Delta/2, h(v) + \Delta/2)$, and we compare the probability estimates for $k = 2, 8, 32$ with the error probability $\pi(R, q, \psi)$ computed on 3,000 samples of the uncertain terrain with $k = 64$. We expect that smaller values of k will converge faster, but will converge at a higher error level. Figure 16 illustrates that the error does decrease as k increases, however, the effect of k on the rate of convergence is not as clear.

Fig. 17. Varying Δ on Indiana with $k = 8$, 40cm rain.

Fig. 18. Errors vs. exact values.

Height Variation Δ . We measured the convergence as the height variation Δ varies on the Indiana dataset. Figure 17 shows that the error increases as Δ increases from 10cm to 80cm, which is expected.

Distribution of Errors vs. Exact Values. We examined the distribution of errors as a function of the exact probability. Figure 18(a) and (b) illustrate that the estimation errors of probabilities are independent of the underlying exact probability. For the simulation in Figure 18(a), we used the Indiana dataset with $\Delta = 40$ cm, $k = 8$, 40cm rain. For the simulation in Figure 18(b), we used the Denmark dataset with 40cm rain.

7 CONCLUSION AND FUTURE WORK

In this article, we have presented efficient data structures for the flooding query problem on terrains. For terrain models without uncertainty, our data structure uses linear space and $O(n \log n)$ preprocessing time, and can determine how much rain has to fall on a region R before a query point q is flooded in $O(|R| + m \log n)$ time, where $|R|$ is the number of vertices in R , m is the number of tributaries of q at which rain is falling, and n is the number of vertices of the terrain. For the specific case when rain falls on a single vertex p , this implies a query time of $O(\log n)$. To our knowledge, this is the first algorithm that can answer a flooding query in sub-linear time in the worst case. A key ingredient in our data structure is the novel approach of computing fill rates, that is, functions indicating the rate at which a depression fills up over time, instead of simply

Fig. 19. Example output with 40cm rain on region R .

computing the fill time of each depression in the terrain. We believe that the fill rate approach can be used to implement efficient data structures for related flood risk analysis problems, such as the flood height problem of computing the water depth above a given terrain vertex v at a given time after v is flooded.

In this article, we assumed that water flows according to a *single-flow direction* (SFD) model, in which water flows from a vertex along one downslope edge. An interesting question is how to answer flooding queries under the *multi-flow direction* (MFD) model, in which water at a vertex can split and flow along multiple downslope edges. Lowe and Agarwal [24] recently presented algorithms to answer the flooding queries under the MFD model. We would like to investigate if ideas from this article can be used to improve those algorithms and answer flooding queries under the MFD model in sub-linear time using a near-linear size data structure. Another question is if our analysis for uncertain terrains under the SFD model can be extended to estimate the flooding probability under the MFD model.

For uncertain terrains, where a probability distribution on terrain data is given, we have proposed a data structure based on a Monte Carlo scheme to estimate the probability of a query point being flooded if a specified amount of rain falls on a query region. Our Monte Carlo scheme is efficient, that is, to obtain an ϵ -approximation with probability $1 - \delta$, the required number of samples from the distribution on terrains is a polynomial in $1/\epsilon$, $\log 1/\delta$, $\log n$, and k , where k is the number of different height values possible for each vertex. In future work, we would like to extend the analysis to handle continuous distributions; that is, we would like to remove the dependence on the parameter k .

In the future, we plan to develop better models for uncertain terrains. In this work, we have taken a very general model, in which the height function is given by some probability distribution H with the only restriction being that each vertex can have at most k distinct values. By refining the model to more precisely reflect real-world remote sensing, it should be possible to improve the performance of the data structure.

Our model of rain regions allows rain to be distributed arbitrarily among the vertices of the terrain without any restriction on the geographical structure of the rain region. In practice, the resolution of terrain models is typically much greater than the resolution of weather forecasts, meaning that a weather forecast will generally assign similar or equal predicted rainfall amount to adjacent vertices of the terrain. In this article, we analyzed the error probability of our Monte Carlo scheme in two different query sets: In the first set, a query is a triple consisting of a vertex p on which rain is falling, a volume ψ of rain, and a query point q for which the flooding probability

is sought. The second query set is parameterized by a rain region R , and queries consist of pairs (q, ψ) , where ψ is the volume of rain on the fixed region R . In the future, we plan to investigate query sets where the rain region R is given as part of the query. Since the space of rain regions in general is n -dimensional, we expect that we have to restrict the structure of allowed regions in order to approximate the flooding probability using a small number of samples. For instance, the terrain vertices could be partitioned into $F \ll n$ forecast regions, and R would be required to assign the same rainfall to vertices in the same forecast region, thus restricting the space of allowed regions to be F -dimensional. An approach similar to this has been investigated for sea-level flood risk by Arge et al. [5].

The analysis of our Monte Carlo scheme exploits the fact that the rain volume function behaves nicely inside each region of the overlay Ξ of height-level maps M_H . This points to several directions for future research into other functions that similarly behave nicely; we conjecture that a similar approach can be used to design an efficient Monte Carlo algorithm for estimating the geometric visibility probability, that is, the probability that the line segment between two query points $x, y \in \Sigma_h$ does not intersect Σ_h .

APPENDIX

A COMPUTING DEPRESSION VOLUMES

In this section, we describe how to extend the merge tree construction algorithm of Carr et al. [11] to compute the depression volumes of arbitrary points $x \in \mathbb{M}$.

The algorithm of Carr et al. [11] computes the *augmented merge tree*, denoted by $\hat{T}_h$, which extends T_h by inserting each vertex $v \in \mathbb{V}$ into the edge of T_h containing $\rho_h(v)$. $\hat{T}_h$ is computed by sweeping the vertices of $\mathbb{M}$ in increasing order of height. When a vertex $v \in \mathbb{V}$ is processed, a new node in $\hat{T}_h$ is created for v , and edges incident to v are added from the highest vertex in each depression delimited by v . The algorithm maintains the set of depressions of $\mathbb{M}_{<h(v)}$ in a union-find data structure, where each depression is represented by the highest vertex in the depression. Using the union-find data structure, each downslope neighbor of v is queried to determine which depressions are delimited by v .

We now describe how to extend the computation to compute the depression volume of every vertex of the terrain. Suppose (u, v) is an edge of $\hat{T}_h$ with $h(u) < h(v)$. Let A_u be the set of points contained in the depression delimited by v but not by u , that is, A_u is the set of points between the contour at u and the contour at v . Restricted to A_u , the depression volume function is a cubic function of height denoted by μ_u . By storing the coefficients of μ_u at u , the depression volume of any point in A_u can be computed. Furthermore, by using the set of triangles constructed in the proof of Lemma 5.2 and denoted by Δ , the coefficients of μ_v can be computed from the coefficients of μ_u and the triangles incident to v in Δ . If v is a negative saddle, then $\mu_{u_1} + \mu_{u_2}$ is used in place of μ_u , where u_1 and u_2 are the children of v in $\hat{T}_h$.

Thus, the function μ_v can be computed when v is processed during the construction of $\hat{T}_h$ without increasing the asymptotic construction time. The cubic function μ_v is stored in the node $v \in \hat{T}_h$ without increasing the asymptotic space usage.

To compute the depression volume of a point x in a face $\phi \in \mathbb{M}$, let v_1, v_2, v_3 be the corners of ϕ such that $h(v_1) < h(v_2) < h(v_3)$. Using standard techniques, we in $O(\log n)$ time query the node $v_1 \in \hat{T}_h$ to find the edge (u, v) of $\hat{T}_h$ containing x , and then we evaluate and return $\mu_u(h(x))$ as $\mu_h(x)$.

REFERENCES

- [1] Pankaj K. Agarwal, Lars Arge, and Ke Yi. 2010. I/O-efficient batched union-find and its applications to terrain analysis. *ACM Trans. Algorithms* 7, 1 (2010), 11.

- [2] Pankaj K. Agarwal, Sayan Mukherjee, and Wuzhou Zhang. 2015. Contour trees of uncertain terrains. In *Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems*. ACM, 43.
- [3] Lars Arge, Jeffrey S. Chase, Patrick Halpin, Laura Toma, Jeffrey S. Vitter, Dean Urban, and Rajiv Wickremesinghe. 2003. Efficient flow computation on massive grid terrain datasets. *GeoInf.* 7, 4 (2003), 283–313.
- [4] Lars Arge, Mathias Rav, Sarfraz Raza, and Morten Revsbæk. 2017. I/O-efficient event based depression flood risk. In *Proceedings of the 19th Workshop on Algorithm Engineering and Experiments*. SIAM, 259–269.
- [5] Lars Arge, Mathias Rav, Morten Revsbæk, Yujin Shin, and Jungwoo Yang. Manuscript. Sea-rise flood prediction on massive dynamic terrains.
- [6] Lars Arge and Morten Revsbæk. 2009. I/O-efficient contour tree simplification. In *Proceedings of the 20th International Symposium on Algorithms and Computation*. 1155–1165.
- [7] Lars Arge, Morten Revsbæk, and Norbert Zeh. 2010. I/O-efficient computation of water flow across a terrain. In *Proceedings of the 26th Annual Symposium on Computational Geometry*. ACM, 403–412.
- [8] Samuel W. Bent, Daniel D. Sleator, and Robert E. Tarjan. 1985. Biased search trees. *SIAM J. Comput.* 14, 3 (1985), 545–568.
- [9] Gerth S. Brodal. 1994. Partially persistent data structures of bounded degree with constant update time. *BRICS Report Series* 1, 35 (1994), 1–24.
- [10] Gerth S. Brodal, George Lagogiannis, and Robert E. Tarjan. 2012. Strict Fibonacci heaps. In *Proceedings of the 44th Annual ACM Symposium on Theory of Computing*. ACM, 1177–1184.
- [11] Hamish Carr, Jack Snoeyink, and Ulrike Axen. 2003. Computing contour trees in all dimensions. *Comput. Geom. Theory Appl.* 24, 2 (2003), 75–94.
- [12] Hamish Carr, Jack Snoeyink, and Michiel van de Panne. 2010. Flexible isosurfaces: Simplifying and displaying scalar topology using the contour tree. *Computational Geometry* 43, 1 (2010), 42–58.
- [13] Danish Geodata Agency. 2007. The Danish Elevation Model DHM-2007/Terræn_bro. Retrieved from <http://eng.gst.dk>.
- [14] Andrew Danner, Thomas Mølhave, Ke Yi, Pankaj K. Agarwal, Lars Arge, and Helena Mitášová. 2007. TerraStream: From elevation data to watershed hierarchies. In *Proceedings of the 15th Annual ACM International Symposium on Advances in Geographic Information Systems*. ACM, 28.
- [15] Edelsbrunner, Harer, and Zomorodian. 2003. Hierarchical Morse—Smale complexes for piecewise linear 2-manifolds. *Discrete Comput. Geom.* 30, 1 (May 2003), 87–107.
- [16] Branko Grünbaum. 1975. Venn diagrams and independent families of sets. *Math. Mag.* 48, 1 (1975), 12–23.
- [17] Sarel Har-Peled. *Geometric Approximation Algorithms*. American Mathematical Society, Boston. 2011.
- [18] Dov Harel and Robert E. Tarjan. 1984. Fast algorithms for finding nearest common ancestors. *SIAM J. Comput.* 13, 2 (1984), 338–355.
- [19] Joe Harris. *Algebraic Geometry: A First Course*. Springer. 1992.
- [20] Indiana Spatial Data Portal. 2013. Indiana Orthophotography (RGBI), LiDAR, and Elevation. Retrieved from http://gis.iu.edu/datasetInfo/statewide/in_2011.php.
- [21] Susan K. Jenson and Julia O. Domingue. 1988. Extracting topographic structure from digital elevation data for geographic information system analysis. *Photogramm. Eng. Remote Sens.* 54, 11 (1988), 1593–1600.
- [22] Marc van Kreveld, René van Oostrum, Chandrajit Bajaj, Valerio Pascucci, and Dan Schikore. 1997. Contour trees and small seed sets for isosurface traversal. In *Proceedings of the 13th Annual Symposium on Computational Geometry*. ACM, 212–220.
- [23] Yuanxin Liu and Jack Snoeyink. 2005. Flooding triangulated terrain. In *Proceedings of the 11th International Symposium on Spatial Data Handling*. Springer, 137–148.
- [24] Aaron Lowe and Pankaj K. Agarwal. 2018. Flood-risk analysis on terrains under the multiflow-direction model. In *Proceedings of the 26th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. ACM.
- [25] John F. O’Callaghan and David M. Mark. 1984. The extraction of drainage networks from digital elevation data. *Comput. Vision Graphics Image Process.* 28, 3 (1984), 323–344.
- [26] Mathias Rav, Aaron Lowe, and Pankaj K. Agarwal. 2017. Flood risk analysis on terrains. In *Proceedings of the 25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. ACM, 36.
- [27] Daniel D. Sleator and Robert E. Tarjan. 1983. A data structure for dynamic trees. *J. Comput. Syst. Sci.* 26, 3 (1983), 362–391.
- [28] Sergey P. Tarasov and Michael N. Vyalyi. 1998. Construction of contour trees in 3D in $O(n \log n)$ steps. In *Proceedings of the 14th Annual Symposium on Computational Geometry*. ACM, 68–75.
- [29] V.N. Vapnik and A. Ya Chervonenkis. 1971. On the uniform convergence of relative frequencies of events to their probabilities. *Theory Probab. Appl.* 16, 2 (1971), 264–280.

Received May 2018; revised September 2018; accepted November 2018