Massively Parallel Algorithms for Computing TIN DEMs and Contour Trees for Large Terrains

Abhinandan Nath Kyle Fox Pankaj K. Agarwal
Kamesh Munagala
Duke University
{abhinath,kylefox,pankaj,kamesh}@cs.duke.edu

ABSTRACT

We propose parallel algorithms in the massively parallel communication (MPC) model (e.g. MapReduce) for processing large terrain elevation data (represented as a 3D point cloud) that are too big to fit on one machine. In particular, given a set S of 3D points that is distributed across multiple machines, we present a simple randomized algorithm to construct a TIN DEM of S by computing the Delaunay triangulation of the xy-projections of points in S, which is also stored across multiple machines. With high probability, the algorithm works in $O(1)$ rounds and the total work performed is $O(n \log n)$. Next, we describe an efficient algorithm in the MPC model for computing the contour tree of the resulting DEM. Under some assumptions on the input, the algorithm works in $O(1)$ rounds and the total work performed is $O(n \log n)$.

CCS Concepts

•Information systems → Geographic information systems; •Theory of computation → MapReduce algorithms;

Keywords

Delaunay triangulation; contour trees, massively parallel communication, MapReduce

1. INTRODUCTION

1.1 Terrain Analysis

The problem of modeling and analyzing massive terrain data sets has been studied extensively in many disciplines such as GIS, spatial databases, computational geometry, and environmental sciences. Formally, a terrain Σ is an xy-monotone surface in $\mathbb{R}^3$ and can be viewed as the graph of a height function $h : \mathbb{R}^2 \to \mathbb{R}$. It is often stored using a digital elevation model (DEM) such as a triangulated xy-monotone surface known as a *triangulated irregular network* (TIN). Here, the surface Σ can be regarded as a triangulation $\mathbb{M}$ of $\mathbb{R}^2$ and a piecewise linear (within each triangle of $\mathbb{M}$) height function $h : \mathbb{M} \to \mathbb{R}$. Because of various nice properties, $\mathbb{M}$ is often the Delaunay triangulation of the points in the plane at which the height of the terrain is observed.

Suppose we have computed a suitable TIN using triangulation $\mathbb{M}$ and height function h. Given a height value ℓ, the ℓ-level set of h is the set of all points on $\mathbb{M}$ whose height values are ℓ. As one continuously varies ℓ, the level sets of h deform, and their topology changes at certain heights. The *contour tree* of h encodes the changes to the levels sets. The contour tree is used in a variety of applications such as prediction of water flow including the computation of watershed hierarchies, the visualization of hydrothermal plumes, and the visualization of climate models [8, 11, 18, 30]. Therefore, it is natural to compute the contour tree of h as a second step to analyzing the terrain it represents.

Recent advances in sensing technology provide both opportunities and obstacles for the aforementioned analysis of terrain data. On the one hand, data like the point sets created by LiDAR are being generated at increasingly higher resolutions, expanding the types and quality of data analyses that are even possible. On the other hand, classical algorithms that run on individual machines cannot keep up with the amount of data generated.

In this paper, we study the problems of constructing the TIN DEM of such large elevation data (e.g., LiDAR data sets), a set of points in $\mathbb{R}^3$, and then constructing the contour tree of this TIN DEM.

1.2 Big Data Platforms

We require "big data" platforms and algorithms to take advantage of these massive data sets. These platforms include MapReduce [20], its open source implementation Hadoop [36], and more recently developed platforms such as Pregel [28], Spark [37], and Google Cloud Dataflow [6]. To simplify analysis of big data, these platforms help programmers focus on computation local to data stored on individual machines; the distribution of data and communication between machines is largely abstracted away.

Various theoretical models have been proposed that capture the salient features of the above programming platforms without being muddied by platform-specific details. In this paper, we focus on the basic *massively parallel communication* (MPC) model of Beame *et al.* [10] as refined by Andoni *et al.* [7] and Agarwal *et al.* [4]. Let n be the size of an input, and let $I = \{1, \ldots, m\}$ be a set of m machines. Let $s = n/m$ (we assume s is an integer for simplicity). Each machine is

SIGSPATIAL'16, October 31-November 03, 2016, Burlingame, CA, USA
© 2016 ACM. ISBN 978-1-4503-4589-7/16/10...$15.00
DOI: http://dx.doi.org/10.1145/2996913.2996952

given $O(s)$ memory (space). We assume $s \geq n^\alpha$ for some positive constant $\alpha < 1$. Machines perform sequential computation on their own data across rounds of computation. All communication between machines occurs between these rounds, and unlike some variants of the model, we allow data to persist on machines between rounds as long as the total space used by each machine never exceeds $O(s)$. In addition, each machine is limited to communicating $O(s)$ words between rounds of computation.

The efficiency of an algorithm in this model is measured using three metrics: the number of rounds of computation R, the total running time T, and the total work performed by the algorithm W. For a machine $\beta \in I$, let $t_{\beta r}$ denote the computation time of machine β during round r. Formally, the total *running time* of an algorithm is defined as

$$T = \sum_{r=1}^{R} \max_{\beta \in I} t_{\beta r},$$

and the total *work* is defined as

$$W = \sum_{r=1}^{R} \sum_{\beta \in I} t_{\beta r}.$$

Note that both of these values ignore the time spent communicating between machines. Generally, one considers R to be the most important metric to minimize in this model, because communication is often the most expensive part of performing distributed computations for big data.

1.3 Related Work

Computing the Delaunay triangulation (and its dual, the Voronoi diagram) is one of the most studied problems in computational geometry; see [9]. Many of the algorithms for computing the Delaunay triangulation in the RAM model run in $O(n \log n)$ time. To account for the challenges of dealing with massive data sets that do not fit in RAM, I/O efficient Delaunay triangulation algorithms have been developed in the two-level I/O-model; see e.g. [17]. There is extensive work on developing algorithms in the Parallel Random Access Machine (PRAM) model of computation; see [12] and the references therein. In addition, Goodrich [23] describes an algorithm for a bulk-synchronous parallel (BSP) computer that constructs convex hulls in 3D. Goodrich's algorithm can be adapted to construct Delaunay triangulations in $O(1)$ rounds in the MPC model of computation, although the algorithm itself is rather involved. Finally, there exist efficient algorithms for constructing the *constrained* Delaunay triangulation which is required to contain a given subset of edges [9]; however, these algorithms are not for the MPC model.

Efficient algorithms for constructing contour trees were discovered more recently. Van Kreveld *et al.* [35] gave the first $O(n \log n)$ time algorithm for our setting of piecewise linear height functions over $\mathbb{R}^2$. This algorithm was later generalized by Tarasov and Vyalyi [33] and Carr *et al.* [14]. Agarwal *et al.* [3] gave an I/O-efficient algorithm for computing the contour tree. There has been a great deal of work on computing contour trees and similar structures using parallel and distributed computing, although these algorithms do not have any theoretical guarantees for the MPC model. See [1, 31, 32] and the references therein. Of particular relevance to our work are the distributed contour and *merge* trees of Morozov and Weber [31, 32]. Suppose one has a TIN $\mathbb{M}$ and height function h stored in a distributed manner. Morozov and Weber describe how to compute trees for the data local to each processor, and then merge simplified versions of these trees so that the simplification is known to all of the processors.

Modern distributed programming frameworks have inspired a great deal of recent work in the MPC and closely related MapReduce models of computation (see, for example [7, 10, 25, 27]), although there is relatively little work done on solving geometric problems in these models. Goodrich *et al.* [25] relate these recently developed models to the classical BSP model [34], immediately yielding some geometric algorithms for MPC as mentioned above. Andoni *et al.* [7] describe geometric approximation algorithms for the MPC model. Agarwal *et al.* [4] describe algorithms to construct distributed variants of classical geometric data structures in the MPC model. Finally, there exist MapReduce implementations for analyzing and querying spatial and geometric data (see [21, 22] and references therein). In particular, SpatialHadoop [22] includes an algorithm to compute the Delaunay triangulation of a point set, though there are no provable bounds on its time-complexity.

1.4 Our Results

In this paper, we describe new algorithms in the MPC model for both parts of the terrain analysis pipeline mentioned above. Given a point set P in $\mathbb{R}^2$ with $|P| = n$, our first result is a randomized algorithm that computes the Delaunay triangulation of P using, with high probability, $O(1)$ rounds of computation in $O(s \log^2 n)$ time and $O(n \log n)$ work. While Goodrich's [23] convex hull algorithm for BSP computers can be adapted for a similar result, our approach is more direct, and is considerably simpler.

Our second result is a deterministic algorithm that computes the contour tree for a height function h over triangulation $\mathbb{M}$ using $O(1)$ rounds of computation, $O(s \log n)$ time, and $O(n \log n)$ work, provided that the following two assumptions hold:

1. $s = \Omega(n^{1/2+\varepsilon})$ for some constant $\varepsilon > 0$, and
2. every line in $\mathbb{R}^2$ intersects $O(\sqrt{n})$ edges of $\mathbb{M}$.

We believe these are reasonable assumptions, because the number of bytes of memory available to individual machines in datacenters should be much higher than the number of available machines. Our second assumption holds for sure when $\mathbb{M}$ is the Delaunay triangulation of points lying on a grid, and it holds with probability at least $1 - 1/n^{\Omega(1)}$, for n sufficiently large when the points are chosen independently and uniformally at random from the unit square [13]. Furthermore, large data sets generated by LiDAR and other technologies have points almost uniformly sampled.

Both of our algorithms rely on a similar divide-and-conquer strategy. Namely, we distribute the input points (triangles) into subsets with small intersection. These subsets are then copied to disjoint sets of machines where we recursively compute the Delaunay triangulation (contour tree) for each subset independently and in parallel. For our Delaunay triangulation algorithm, we begin by sampling a small subset of points S and building the Delaunay triangulation of the sample. The set of input points are then distributed based on how they conflict with edges in the triangulation of S. This idea was used before in, for example, the constrained

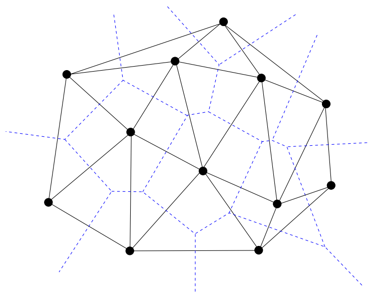

Figure 1. The Delaunay triangulation and Voronoi diagram (in blue dashed edges) of a set of points.

Delaunay triangulation algorithm of Agarwal *et al.* [2], but implementing the scheme in the MPC model requires additional ideas. We need high probability bounds instead of simple expectation bounds on the size of "conflict lists", and this involves adapting the *exponential decay lemma* [29] to our scenario using a two-level sampling procedure.

For our algorithm to build contour trees, we combine recursively constructed contour trees for each subset of triangles, taking advantage of the subsets' small intersection. Unlike the algorithm of Morozov and Weber [31, 32], we do not rely on an initial distribution of the input triangles to induce small regions of the triangulation with limited intersection. Instead, we compute our own recursive hierarchy of triangle subsets that contain few boundary vertices. Further, we combine multiple trees simultaneously to minimize communication instead of combining them in pairs.

2. PRELIMINARIES

2.1 Delaunay Triangulations

Given a set of n points $P \in \mathbb{R}^2$, a triangulation of P is a maximal subdivision of the plane into triangles having vertices from P. A triangle belonging to a given triangulation is said to be *Delaunay* if its circumcircle does not contain any point of P in its interior. A triangulation of P is called a *Delaunay triangulation*, denoted by $\mathcal{DT}(P)$, if all the triangles in $\mathcal{DT}(P)$ are Delaunay.

Given a planar point set P, the *Voronoi cell* of a point $p \in P$, $\mathrm{Vor}_P(p)$, is the set of all those points $q \in \mathbb{R}^2$ that are closer to p than any other point of P, i.e.,

$$\mathrm{Vor}_P(p) = \{q \in \mathbb{R}^2 \mid d(p,q) \le d(p',q)\ \forall p' \in P\}.$$

The *Voronoi diagram* of P, $\mathrm{Vor}(P)$, is the subdivision of the plane into the Voronoi cells of the points of P. A classical result states that $\mathcal{DT}(P)$ and $\mathrm{Vor}(P)$ are *duals* of each other, i.e., there exists an edge between two points $p, q \in P$ in $\mathcal{DT}(P)$ iff $\mathrm{Vor}_P(p)$ and $\mathrm{Vor}_P(q)$ share an edge in $\mathrm{Vor}(P)$. See Figure 1.

2.2 Terrain and Contour Trees

Terrains. Let $\mathbb{M} = (V, E, F)$ be a triangulation of $\mathbb{R}^2$, with vertex, edge, and face (triangle) sets V, E, and F, respectively, and let $n = |V|$. We assume that $\mathbb{M}$ is finite and contains one boundary component (our algorithm can be modified easily to remove the boundary by assuming V contains a vertex v_∞ at infinity). Let $h : \mathbb{M} \to \mathbb{R}$ be a *height function.* We assume that the restriction of h to each triangle of $\mathbb{M}$ is a linear map, 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 $\mathbb{M}$. See Figure 2. If h is clear from context, we denote Σ_h as Σ. The vertices, edges, and faces of Σ are in one-to-one correspondence with $\mathbb{M}$. With a slight abuse of terminology, we refer to V, E, and F as vertices, edges, and triangles of both Σ and $\mathbb{M}$.

Critical points. For a vertex v of $\mathbb{M}$, the *link* of v, denoted $\mathrm{Lk}(v)$, is the cycle or path formed by the edges of $\mathbb{M}$ that are not incident on v but belong to the triangles incident to v. The lower (resp. upper) link of v, $\mathrm{Lk}^-(v)$ (resp. $\mathrm{Lk}^+(v)$), is the subgraph of $\mathrm{Lk}(v)$ induced by vertices u with $h(u) < h(v)$ (resp. $h(u) > h(v)$). A *minimum* (resp. *maximum*) of $\mathbb{M}$ is a vertex v for which $\mathrm{Lk}^-(v)$ (resp. $\mathrm{Lk}^+(v)$) is empty. A maximum or a minimum vertex is called an *extremal* vertex. A non-extremal vertex v is *regular* if $\mathrm{Lk}^-(v)$ (and also $\mathrm{Lk}^+(v)$) is connected, and *saddle* otherwise. A vertex that is not regular is called a *critical* vertex.

Level sets and contours. Given any value $\ell \in \mathbb{R}$, the *ℓ-level set*, the *ℓ-sublevel set*, and the *ℓ-superlevel set* of $\mathbb{M}$, denoted as $\mathbb{M}_\ell$, $\mathbb{M}_{\le \ell}$, and $\mathbb{M}_{\ge \ell}$, respectively, consist of points $x \in \mathbb{R}^2$ with $h(x) = \ell$, $h(x) \le \ell$, $h(x) \ge \ell$, respectively. A connected component of $\mathbb{M}_\ell$ is called a *contour.* See Figure 2. Each point $v \in \mathbb{R}^2$ is contained in exactly one contour in $\mathbb{M}_{h(v)}$, which we call *the contour of v*. The contour of a noncritical point is a simple polygonal cycle with non-empty interior or a polygonal boundary-to-boundary arc. The contour of a local minimum or maximum v only consists of the single point v, and the contour of a saddle vertex v consists of two or more simple cycles and/or arcs with v being their only intersection point.

Contour trees. Consider raising ℓ from $-\infty$ to ∞. The contours continuously deform, but no changes happen to the topology of the level set as long as ℓ varies between two consecutive critical levels. A new contour appears as a single point at a minimum vertex, and an existing contour contracts into a single point and disappears at a maximum vertex. An existing contour splits into two new contours or two contours merge into one contour at a saddle vertex. The *contour tree* $\mathcal{T}_h$ of h is a tree on the critical vertices of $\mathbb{M}$ that encodes these topological changes of the level set. An edge (v, w) of $\mathcal{T}_h$ *represents* the contour that appears at v and disappears at w.

Formally, $\mathcal{T}_h$ is the quotient space in which each contour is represented by a point and connectivity is defined in terms of the quotient topology. Let $\rho : \mathbb{M} \to \mathcal{T}_h$ be the associated quotient map, which maps all points of a contour to a single point on an edge of $\mathcal{T}_h$. Fix a point p in $\mathbb{M}$. If p is not a critical vertex, $\rho(p)$ lies in the relative interior of an edge in $\mathcal{T}_h$; if p is an extremal vertex, $\rho(p)$ is a leaf node of $\mathcal{T}_h$; and if p is a saddle vertex then $\rho(p)$ is a non-leaf node of $\mathcal{T}_h$. See Figure 2. We use h to denote the height function on the points of $\mathcal{T}_h$ as well. The definition of contour trees as a quotient map actually applies to any simply connected topological space Σ with associated height function h.

Join trees and split trees. Analogous to the contour tree of Σ which encodes the topological changes to $\mathbb{M}_\ell$ as we increase ℓ from $-\infty$ to ∞, the *join tree* $\mathcal{J}_h$ (resp. *split tree* $\mathcal{S}_h$) encodes the topological changes in $\mathbb{M}_{\le \ell}$ (resp. $\mathbb{M}_{\ge \ell}$). Its leaves are minima (resp. maxima) of Σ, and its internal nodes are some of the saddle vertices of Σ. The contour

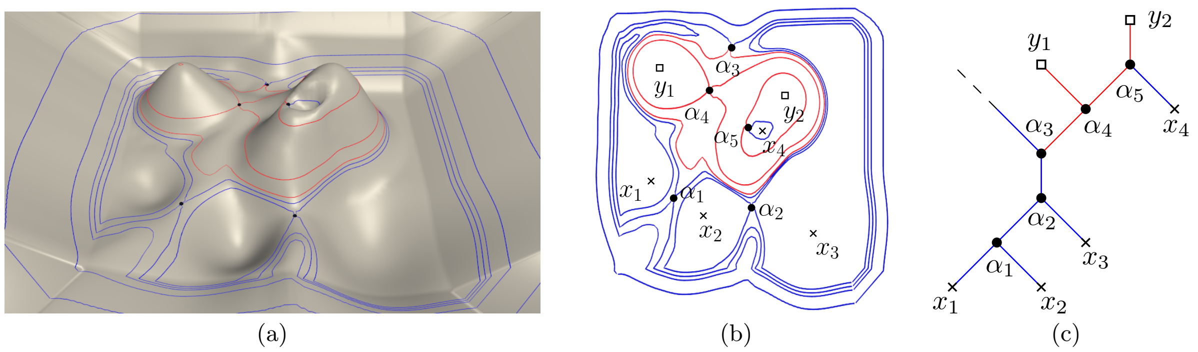

Figure 2. (a) (b) An example terrain depicted with contours through saddle vertices and showing the critical vertices of the terrain: $\alpha_1, \alpha_3, \alpha_4, \alpha_5$ are saddles. (c) The contour tree of the terrain in (a). This figure is taken from [5].

tree, join tree, and split tree can all be *augmented* with additional vertices of $\mathbb{M}$ by subdividing the edges representing the contours, sublevel set components, or superlevel set components containing these vertices. Carr *et al.* [14] describe how to build the contour tree $\mathcal{T}_h$ in time linear in its size given the join and split trees augmented with each other's nodes.

2.3 Range Spaces

A *range space* Y is a pair $(X, \mathcal{R})$, where X is a ground set and $\mathcal{R}$ is a family of subsets (*ranges*) of X. For example, X is a set of points in $\mathbb{R}^2$ and $\mathcal{R} = \{X \cap \mathcal{D} \mid \mathcal{D}$ is an open disc in $\mathbb{R}^2\}$. A subset $X' \subseteq X$ is *shattered* by Y if $\{X' \cap R \mid R \in \mathcal{R}\} = 2^{X'}$. The *VC dimension* of Y is the size of the largest subset of X shattered by Y. If there are arbitrarily large shattered subsets, the VC dimension of Y is set to ∞. The only ranges we consider in this paper are those induced by the union of two open discs in $\mathbb{R}^2$. Such range spaces have constant VC dimension.

Given a range space $Y = (X, \mathcal{R})$ and $0 \le \varepsilon \le 1$, a subset $X' \subseteq X$ is called an *ε-net* of Y if for any range $R \in \mathcal{R}$ with $|R| \ge \varepsilon|X|$, set X' intersects R. Similarly, given a range space $Y = (X, \mathcal{R})$ and $0 \le \varepsilon \le 1$, a subset $X' \subseteq X$ is called an *ε-approximation* of Y if for any range $R \in \mathcal{R}$, we have

$$\left| \frac{|X' \cap R|}{|X'|} - \frac{|R|}{|X|} \right| \le \varepsilon.$$

2.4 MPC Primitives

We mention here some primitive operations used by our algorithms. The implementations of these or similar primitives are described in [4].

`Broadcast`$(\mathcal{S}, I, \beta)$**:** Given a set of words $\mathcal{S}$, a contiguous set of machines $I = \{i_0, i_0+1, \dots\}$, and a machine β storing $\mathcal{S}$, copy $\mathcal{S}$ to all machines in I. Assuming $\mathcal{S}$ has size $O(s^{1/2})$, `Broadcast`$(\mathcal{S}, I, \beta)$ can be implemented by a deterministic algorithm using $O(1)$ rounds of computation, $O(s)$ time, and $O(n)$ total work.

`Distribute`(X, I, Π)**:** Given a function Π from a set of objects X to subsets of $\{1, \dots, k\}$, partition the machines of I into disjoint contiguous subsets $I_j \subseteq I$ for each $j \in \{1, \dots, k\}$. Copy each object $x \in X$ once to a machine in each subset I_j such that $j \in \Pi(x)$. We require I itself to be a contiguous subset of machines $\{i_0, i_0 + 1, \dots\}$. We also require that $k = O(s^{1/2})$, $k \le |I|$, each value $j \in \{1, \dots, k\}$ appear at most $O(|X|/k)$ times, and the total number of copies needed of objects lying on a single machine $i \in I$ is $O(s)$. `Distribute`(X, I, Π) can be implemented by a deterministic algorithm using $O(1)$ rounds of computation, $O(s \log s)$ time, and $O(n \log n)$ total work.

`Sample`(P, I, r, β)**:** Given a set of points P stored on a contiguous subset of machines $I = \{i_0, i_0 + 1, \dots\}$, compute a subset $S \subseteq P$ of size $O(r^2 \log n)$ and send it to machine β. Subset S should be a $(1/r)$-approximation[1] with probability at least $1 - 1/n^{\Omega(1)}$. Assuming $|S| = O(s^{1/2})$, `Sample`(P, I, r, β) can be implemented to always find some subset of the appropriate size using $O(1)$ rounds of computation, $O(s)$ time, and $O(|P|)$ total work, with probability at least $1 - 1/n^{\Omega(1)}$.

3. DELAUNAY TRIANGULATION

Let $P \subset \mathbb{R}^2$ be a set of n points. We now describe our algorithm to compute the Delaunay triangulation $\mathcal{DT}(P)$ of P. For simplicity of presentation, we assume that no four points in P are co-circular, in which case $\mathcal{DT}(P)$ is unique. The algorithm can be adopted to handle the case when four or more points are co-circular using standard techniques. We also add three points at infinity and every convex-hull vertex of P is connected to one of these points by an edge (which is a ray). These additional points and edges ensure that $\mathcal{DT}(P)$ is a triangulation of the entire plane and that each edge is adjacent to two triangles. We refer to this augmented structure as the *augmented Delaunay triangulation* of P, and we will not distiniguish between the standard and augmented Delaunay triangulation.

At a very high level, our algorithm works as follows: we randomly sample a small subset $S \subseteq P$ and compute $\mathcal{DT}(S)$ locally on one machine. For each edge $e \in \mathcal{DT}(S)$, we compute the points in P that are in *conflict* with e. We then compute the Delaunay triangulation of each such set, and discard some of the *invalid* triangles to get $\mathcal{DT}(P)$.

3.1 Conflict Lists and Kernels

Consider a subset $S \subseteq P$. Let $e = pq$ be an edge of $\mathcal{DT}(S)$, where $\triangle pqu$ and $\triangle pqv$ are the triangles adjacent

[1]The algorithm for computing a $(1/r)$-approximation depends on the underlying range space. In our applications, the range space associated with P is always the one induced by a finite union and intersection of discs in $\mathbb{R}^2$.

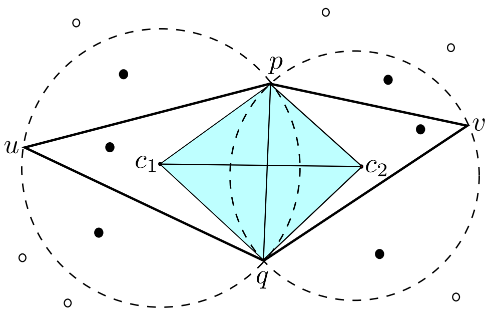

Figure 3. The kernel (in blue) for the edge pq. The points c_1 and c_2 are the circumcenters of triangles Δpqu and Δpqv resp., and c_1c_2 is the edge between $\mathrm{Vor}_S(p)$ and $\mathrm{Vor}_S(q)$. The solid points are in conflict with pq but the hollow points are not.

to e. (Recall that each edge is adjacent to two triangles in the augmented $\mathcal{DT}(S)$.) Abusing the notation a little, by a fixed edge e we will often mean e alongwith the two triangles adjacent to e. We identify the edge pq of $\mathcal{DT}(S)$ with the four tuple $\mathsf{D}(e) = \{p, q, u, v\}$, i.e., the two endpoints of e and the two other vertices of triangles adjacent to e. The *conflict list* of e, denoted by $P_{|e}$, is the set of all those points in P that lie in the circumcircle of $\triangle pqu$ or $\triangle pqv$. The following lemma is well known (see e.g. [19]):

Lemma 3.1. *An edge $e = pq$ along with triangles $\triangle pqu, pqv$ appears in $\mathcal{DT}(S)$ if and only if (i) $\mathsf{D}(e) \subseteq S$ and (ii) $P_{|e} \cap S = \emptyset$.*

The *kernel* of e, $\ker(e)$, is the set of points $x \in \mathrm{Vor}_S(p) \bigcup \mathrm{Vor}_S(q)$ such that the ray $\overrightarrow{px}$ or $\overrightarrow{qx}$ intersects the edge between $\mathrm{Vor}_S(p)$ and $\mathrm{Vor}_S(q)$. See Figure 3.

The following lemma from [2] is the basis of our algorithm.

Lemma 3.2. *Let $S \subseteq P$. Then the following hold.*

(a) The set $\{\ker(e) \mid e \in \mathcal{DT}(S)\}$ partitions the plane.

(b) A triangle Δ is in $\mathcal{DT}(P)$ iff there exists an edge $e \in \mathcal{DT}(S)$ such that $\Delta \in \mathcal{DT}(P_{|e})$ and the circumcenter of Δ lies in $\ker(e)$.

Lemma 3.2 suggests the following algorithm: compute the Delaunay triangulation of a small sample $S \subseteq P$ locally, compute the conflict list for each edge of $\mathcal{DT}(S)$, recursively build the Delaunay triangulation for each list in parallel, and discard the triangles that do not satisfy condition (b) of Lemma 3.2. We formalize this algorithm later.

3.2 Random Sampling

Consider the Delaunay triangulation of a uniformly random sample $R \subseteq P$ of size r. Consider the range space $Y = \{(P, \{P \cap \mathcal{D} \mid \mathcal{D} \text{ is an open disc}\}\}$. Y has constant VC dimension and hence R is an $O(\log(n)/r)$-net of Y with probability at least $1 - 1/n^{\Omega(1)}$ [26]. Any (open) circumcircle $\mathcal{D}$ of a triangle of $\mathcal{DT}(R)$ does not contain any points of R. Hence, by definition of an $O(\log(n)/r)$-net, we have $|P \cap \mathcal{D}| = O(n\log(n)/r)$. Thus, the size of all conflict lists is $O(n\log(n)/r)$ with probability at least $1 - 1/n^{\Omega(1)}$. To get rid of the extra $O(\log n)$ factor in the size, we do a *second level of sampling* as follows. For each edge $e \in \mathcal{DT}(R)$ with $|P_{|e}| = t_e \cdot n/r$, we sample a subset $R_{|e}$ of $O(t_e \ln t_e)$ points from $P_{|e}$ uniformly at random. We repeat both levels of sampling until $|P_{|e|e'}| = O(|P|/r)$ for every $e \in \mathcal{DT}(R), e' \in \mathcal{DT}(R_{|e})$ and $\sum_{e\in\mathcal{DT}(R)} |R_{|e}| = O(r)$. Applying Lemma 3.2 again gives us that a triangle $\Delta \in \mathcal{DT}(P_{|e|e'})$ is in $\mathcal{DT}(P)$ if and only if its circumcenter lies in $\ker(e) \cap \ker(e')$.

We will now show that the procedure only requires $O(\log n)$ repetitions of the two-level sampling with high probability. For an integer t, let

$$\mathcal{F}_t(R) = \{e \in \mathcal{DT}(R) \mid t \cdot n/r \le |P_{|e}| < (t+1) \cdot n/r\}.$$

Lemma 3.3. *For all $e \in \mathcal{DT}(R)$ and all $e' \in \mathcal{DT}(R_e)$, we have $|P_{|e|e'}| = O(n/r)$ with probability $\Omega(1)$.*

Proof. For an edge $e \in \mathcal{F}_{t_e}(R)$, we sample $O(t_e \ln t_e)$ points $R_{|e}$ from $P_{|e}$. Again, $R_{|e}$ is a $O(1/t_e)$-net of $P_{|e}$ with probability $1 - 1/t_e^{\Omega(1)}$. Applying the union bound, we see that each edge in $\mathcal{DT}(R_{|e})$ has $O(|P_{|e}|/t_e) = O(n/r)$ points of $P_{|e}$ in conflict with it, with probability at least $1 - \sum_{e\in\mathcal{DT}(R)} 1/t_e^{\Omega(1)}$ which is a positive constant. □

We now show that we sample only a small number of additional points in each attempt at two-level sampling. The following exponential decay lemma, originally proved in [15, 29], is crucial for the analysis. For a parameter $r \le n$, let $\phi_t(r) = \mathbb{E}[|\mathcal{F}_t(R)|]$, where R is a random sample of size r. We define $\phi_{\ge t}(r) = \sum_{t'\ge t} \phi_t(r)$. We set $\phi(r) = \phi_{\ge 0}(r)$.

Lemma 3.4. *For any $t \ge 1$,*

$$\phi_{\ge t}(r) \le t^{O(1)} \exp(-(t-2)) \cdot \phi(r/t).$$

Proof. The proof is adapted from [29] to our context. Let X be the set of all edges (alongwith the two adjacent triangles) that can appear in the Delaunay triangulation of a subset of P, and let $\mathsf{X}_t = \{e \in \mathsf{X} \mid |P_{|e}| \ge tn/r\}$.

For simplicity, we prove the lemma under a slightly different probability model, where R is not a random subset of size r but selected by choosing each point of P independently with probability $p = r/n$. The bound holds for the desired probabilistic model by a straightforward modification.

For an edge $e \in \mathsf{X}_t$, $\pi(e) = \Pr[e \in \mathcal{DT}(R)]$. By Lemma 3.1, $\pi(e) = p^4(1-p)^{\kappa_e}$, where $\kappa_e = |P_{|e}|$. Therefore,

$$\phi_{\ge t}(r) = \sum_{e\in\mathsf{X}_t} \pi(e) = \sum_{e\in\mathsf{X}_t} p^4(1-p)^{\kappa_e}.$$

We choose another random sample R' by choosing each point with probabibility $p' = p/t = \frac{r}{tn}$. Then

$$\begin{aligned}\phi(r/t) &= \sum_{e\in\mathsf{X}} p'^4(1-p')^{\kappa_e} \\ &\ge \sum_{e\in\mathsf{X}_t} t^{-4}p^4(1-p)^{\kappa_e}\left(\frac{1-p/t}{1-p}\right)^{\kappa_e} \\ &\ge \sum_{e\in\mathsf{X}_t} p^4(1-p)^{\kappa_e}\psi(e),\end{aligned}$$

where

$$\psi(e) = t^{-4}\left(\frac{1-p/t}{1-p}\right)^{\kappa_e}.$$

Assuming $p \le 1/2$ and using the fact $1 - x \le e^{-x}$, $1 - x \ge e^{-2x}$, and $\kappa_e \ge tn/r$ for $e \in \mathsf{X}_t$, we obtain

$$\psi(e) \ge t^{-4}\exp\left(\kappa_e(\frac{-2p}{t} + p)\right) \ge t^{-4}\exp(t-2).$$

Hence,

$$\phi_{\geq t}(r) \leq t^4 \exp(-(t-2))\phi(r/t).$$

□

We show that the Lemma 3.4 immediately implies that the expected additional number of points sampled is $O(r)$.

Lemma 3.5. *We have* $\mathbb{E}\left[\sum_{e\in\mathcal{DT}(R)} |R_{|e}|\right] = O(r)$.

Proof. We have,

$$\begin{aligned}\mathbb{E}\left[\sum_{e\in\mathcal{DT}(R)} |R_{|e}|\right] &\leq \sum_{t\geq 0}(t+1)\ln(t+1)\cdot\phi_t(r)\\ &\leq \sum_{t\geq 0}(t+1)\ln(t+1)\cdot\phi_{\geq t}(r)\\ &\leq \phi(r)\sum_{t\geq 0} t^{O(1)}\ln(t+1)\exp(-t+2)\\ &\leq O(r).\end{aligned}$$

□

Finally, we prove our claim.

Lemma 3.6. *The above procedure repeats two-level sampling at most $O(\log n)$ times with probability at least $1 - 1/n^{\Omega(1)}$.*

Proof. Let c a constant lower bound of the probability given in Lemma 3.3. By Markov's inequality, we have $\sum_{e\in\mathcal{DT}(R)} |R_{|e}| = O(r)$ with probability at least $c' > 1 - c$ by allowing the constant in the big-O to be sufficiently high. Therefore, the conditions of both Lemmas 3.3 and 3.5 are satisfied simultaneously with some constant probability. Within $O(\log n)$ repetitions of the two-level sampling, both conditions will hold with probability $1 - 1/n^{\Omega(1)}$. □

3.3 Algorithm

Sequential algorithm. We first briefly describe a sequential algorithm to compute the conflict lists of a sample. Given a subset $R \subseteq P$ with $|P| = n$ and $|R| = r$, we construct $\mathcal{DT}(R)$ and a point-location data structure on the triangles of $\mathcal{DT}(R)$. Using this data structure, for each $p \in P$, we determine the triangle $\Delta_p \in \mathcal{DT}(R)$ containing p. We say that a triangle Δ is in conflict with p if p lies in the interior of the circumcircle of Δ. The proof of the following well-known lemma can be found in [19].

Lemma 3.7. *The triangles of $\mathcal{DT}(R)$ that are in conflict with a point p form a connected subgraph in the dual graph (in the planar graph duality sense) of $\mathcal{DT}(R)$.*

The triangles in conflict with p can be found by performing a breadth-first search from Δ_p. Once these triangles are found, the edges that conflict with p can be found since they are the edges of the triangles. Point p is then added to the corresponding conflict lists.

Lemma 3.8. *Given a subset $R \subseteq P$ with $|P| = n$ and $|R| = r$, we can compute the conflict lists for the triangles of $\mathcal{DT}(R)$ using the above algorithm in time $O(n \log r + k)$, where k is the total size of the conflict lists.*

Proof. Computing $\mathcal{DT}(R)$ takes time $O(r \log r)$. Since there are $O(r)$ triangles in $\mathcal{DT}(R)$, the point location data structure can be constructed in $O(r \log r)$ time, and it answers a point-location query in $O(\log r)$ time [19]. Thus, doing a point location and a subsequent breadth-first search for a point p takes time $O(\log r + k_p)$ time, where k_p is the number of edges that p is in conflict with. Summing over all points, we get the total running time to be $O(n \log r + k)$. □

Two-level sampling in the MPC model. We describe an MPC procedure called `Two-Level-Sample` (P, I), where P is a set of points distributed evenly across machines I. We need to compute the value of t_e as defined in the previous section for every edge e in $\mathcal{DT}(R)$ of a random sample $R \subseteq P$. Computing the exact values of t_e for the set $P_{|e}$ will require sending the sample R to all the machines, since P is distributed across all machines. However, the conditions of Lemmas 3.3 and 3.5 can be guaranteed even if we estimate the value of t_e to within a constant factor of the actual value.

In particular, to reduce the amount of communication, we first compute a $(1/r)$-approximation S of P using the `Sample` primitive with high probability, send it to a machine $\beta \in I$ and perform the sequential two-level sampling procedure described above on the set S. This produces $\mathcal{DT}(R)$ for a random subset $R \subseteq S$ of size r, and a second level triangulation $\mathcal{DT}(S_{|e})$ for each edge $e \in \mathcal{DT}(R)$. We send the second level triangulations to all the machines containing the set P, using the `Broadcast` primitive.

Computing the conflict lists. Each machine then does the following in parallel – for each point $p \in P$ residing on that machine, it computes the number of edges in the second level triangulations that the point p is in conflict with. Let this number be denoted by k_p. Value k_p can be computed for all p on the machine by modifying the algorithm of Lemma 3.8 slightly to report only the number of lists containing point p, within the same time bound. We will show later that $\sum_{\beta\in I}\sum_{p\in\beta} k_p = O(|P|)$ with high probability. Thus all the conflict lists will fit on the machines I. Let K denote an array of the values k_p, indexed by the point p. Each machine divides the portion of array K indexed by that machine's points into contiguous subarrays K_j, each summing up to cs, for some suitable constant c. For each such subarray K_j, the machine sends the points P_j corresponding to the indices of the subarray to a machine i_j. Note that $i_{j_1} \neq i_{j_2}$ for any $j_1 \neq j_2$. The machine i_j is chosen in a manner similar to the implementation of the `Partition` primitive of [4].

Once the points have been distributed, we compute the set of all edges in the second level triangulations that the points are in conflict with, using Lemma 3.8. Since for a machine i_j we have $\sum_{p\in i_j} k_p \leq cs$, all the edges computed for points in i_j fit on i_j. The algorithm determines if there exists a set $P_{|e|e'}$ such that $|P_{e|e'}| > c'|P|/r$ for some sufficiently large constant c'. If so, the entire algorithm restarts. We then use the `Distribute` primitive so that the set $P_{|e|e'}$ lies on a disjoint set of machines $I_{|e|e'}$. We show later that $|P_{|e|e'}| \leq c'|P|/r$ with high probability.

Overall MPC algorithm. We compute $\mathcal{DT}(P)$ using a recursive procedure we call `DelaunayTriangulation`, which takes as input a set of points P, machines I and a set of edges E along with their adjoining triangles, and computes $\mathcal{DT}(P)$ but retains only those triangles whose circumcenters lie in $\mathbb{R}^2 \bigcap \left(\bigcap_{e\in E} \ker(e)\right)$.

If $|I| = 1$, all of the above is done sequentially. If $|I| > 1$, we first shuffle the points of P randomly across machines in I and then run `Two-Level-Sample` (P, I) to get sets $P_{|e|e'}$ on machines $I_{|e|e'}$. For each such set $P_{|e|e'}$, we recursively call `DelaunayTriangulation`$(P_{|e|e'}, I_{|e|e'}, E \cup e, e')$. The pseudocode is described in Algorithm 1.

Algorithm 1 `DelaunayTriangulation`(P, I, E)

```
1: If $|I| = 1$, compute $\mathcal{DT}(P)$ locally and retain those triangles whose circumcenters lie in $\cap_{e \in E}\ker(e)$.
2: Randomly shuffle the points of $P$ across machines in $I$.
3: $\{(P_{|e|e'}, I_{|e|e'})\} \leftarrow$ Two-Level-Sample $(P, I)$.
4: for all $P_{|e|e'}$ in parallel do
5:     DelaunayTriangulation $(P_{|e|e'}, I_{|e|e'}, E \cup \{e, e'\})$.
6: end for
```

3.4 Analysis

We first show that the $O(r)$ subproblems produced are of almost equal sizes.

LEMMA 3.9. *For any pair of edges e and e' produced in the first and second levels of sampling and triangulation on the sets S and $S_{|e}$, respectively, $|P_{|e|e'}| = O(|P|/r)$ with probability at least $1 - 1/n^{\Omega(1)}$.*

PROOF. The set $P_{e|e'}$ (resp. $S_{e|e'}$) is obtained by taking all those points of P (resp. S) which lie in the conflict lists of e and e'. The region defined by these conflict lists is obtained by a finite union and intersection of disks (in particular, the union of two disks each for the conflict lists of e and e', followed by an intersection of these two regions). The VC dimension of the range space defined by such regions is a constant and so the set S is a $(1/r)$-approximation of the range space defined on the set P with probability at least $1 - 1/n^{\Omega(1)}$. Thus, since $|S_{|e|e'}|/|S| = O(1/r)$ after two-level sampling, we have that $|P_{|e|e'}|/|P| = O(1/r)$ with probability at least $1 - 1/n^{\Omega(1)}$. $\square$

Since there are $O(r)$ subproblems, the total size of the conflict lists is $O(r) \cdot O(|P|/r) = O(|P|)$, and hence the subproblems fit on the machines I.

LEMMA 3.10. *Procedure* `Two-level-sample` (P, I) *takes $O(1)$ rounds of computation, $O(s \log^2 n)$ time, and $O(|P| \log |P|)$ work, with probability at least $1 - 1/n^{\Omega(1)}$.*

PROOF. Each of the time bounds below either hold deterministically or with probability at least $1 - 1/n^{\Omega(1)}$. Computing the set S takes $O(1)$ rounds of computation, $O(s \log s)$ time, and $O(|P| \log |P|)$ work. Two level sampling on the set S takes expected time and work $O(|S| \log r \log n) = O(s \log^2 n)$, by Lemma 3.8. Sending the second level of triangulations to all the machines takes one round of computation, $O(s \log s)$ time, and $O(|P| \log |P|)$ work. Computing the array K takes $O(|P| \log r) = O(|P| \log |P|)$ work, by Lemma 3.8 and the fact that the total size of the conflict lists is $O(|P|)$. The time taken is $O(s \log r + \max_\beta \sum_{p \in \beta} k_p)$. The value of $\max_\beta \sum_{p \in \beta} k_p$ can in the worst case be $O(sr)$. However, if we randomly shuffle the points of P in the beginning across all machines, a simple Chernoff bound argument shows that $\max_\beta \sum_{p \in \beta} k_p = O(s \log |P|)$ with probability at least $1 - 1/n^{\Omega(1)}$.
Goodrich et al. [25] show that random shuffling can be done in $O(1)$ rounds, $O(s \log s)$ time, and $O(|P| \log |P|)$ work. Distributing the points using the array K takes $O(1)$ rounds, $O(s \log s)$ time, and $O(|P| \log |P|)$ work. After distribution, each machine's points have conflict list size $O(s)$ and thus computing the conflict lists takes $O(s \log s)$ time and $O(|P| \log |P|)$ work, by Lemma 3.8. Finally, distributing the sets $P_{|e|e'}$ using the `Distribute` primitive takes $O(1)$ rounds, $O(s \log s)$ time and $O(|P| \log |P|)$ work.

Since $P_{|e|e'} = O(|P|/r)$ with probability at least $1 - 1/n^{\Omega(1)}$, the number of levels of recursion is $O(\log_r |P|) = O(1)$. By induction, each recursive level takes $O(1)$ rounds, $O(s \log^2 n)$ time, and $O(|P| \log |P|)$ work. The algorithm avoids ever restarting due to overly large conflict lists with high probability. The bounds in the statement of the lemma follow. $\square$

The correctness of our algorithm follows from Lemma 3.2 and the subsections following it. We thus have the following.

THEOREM 3.11. *Given a set P of n points in $\mathbb{R}^2$, the Delaunay triangulation of P can be built in the MPC model using $O(1)$ computation rounds, $O(s \log^2 n)$ time, and $O(n \log n)$ work, with probability at least $1 - 1/n^{\Omega(1)}$.*

4. CONTOUR TREE

We now describe our algorithm to compute a contour tree of a terrain. Let $\mathbb{M} = (V, E, F)$ be a triangulation in $\mathbb{R}^2$, and let $n = |V|$. Let $h : \mathbb{M} \to \mathbb{R}$ be a piecewise linear height function over $\mathbb{M}$. We assume $s = \Omega(n^{1/2+\varepsilon})$ for some constant $\varepsilon > 0$ and that every line in $\mathbb{R}^2$ intersects $O(\sqrt{n})$ edges of $\mathbb{M}$. Let Σ be the terrain described by $\mathbb{M}$ and h.

4.1 Distributed Contour Trees

The contour tree $\mathcal{T}$ constructed by our algorithm will be stored in a distributed fashion, similar to the data structures of Agarwal *et al.* [4]. Let $\beta \in I$ be an arbitrary machine. Given a degree-2 vertex of a tree, we say we *splice* the vertex by removing the vertex and adding an edge between its two neighbors. On β, we store a tree $\mathcal{T}_0$ formed by taking a subtree of $\mathcal{T}$ and iteratively splicing degree-2 vertices. For each vertex v of $\mathcal{T}$ that is a leaf in $\mathcal{T}_0$, let e_v be the edge incident to v in $\mathcal{T}$ separating v from other vertices in $\mathcal{T}$ that are also in $\mathcal{T}_0$. We define the *rooted subtree* of v as the connected component of $\mathcal{T} \setminus e_v$ that includes v. Let u, w be a pair of vertices in $\mathcal{T}$, and let e_u and e_w be the edges incident to u and w respectively along the unique path between u and w in $\mathcal{T}$. We define the *vine* of uw to be the connected component containing the path between u and w after removing every edge incident to u and w except e_u and e_w. Now, for each leaf v of $\mathcal{T}_0$, we recursively store the subtree $\mathcal{T}_v$ rooted at v on a subset of machines I_v (this subtree may contain only v). In addition, for each edge uw of $\mathcal{T}_0$, we recursively store the vine of uw on a subset of machines I_{uw}. See Figure 4. These subsets of machines are not necessarily disjoint. The subtrees are stored in such a way that for any vertex r in $\mathcal{T}_0$, number of recursively stored subtrees encountered on any r to leaf path in $\mathcal{T}$ is $O(1)$. In other words, r to leaf traversals can be performed in $O(1)$ rounds of computation and time linear in the number of vertices along the path.

Our high-level idea for computing the tree $\mathcal{T}$ is as follows: We split $\mathbb{M}$ into pieces with approximately equal numbers of vertices. We then recursively compute distributed contour trees within each piece independently and in parallel.

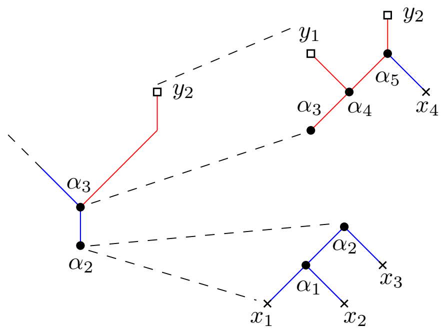

Figure 4. A possible top level subtree $\mathcal{T}_0$ for the contour tree given in Figure 2 is given on the left. Emanating to the right are the rooted subtree of α_2 and the vine of $\alpha_3 y_2$. The rooted subtree and vine may be stored on separate machines from $\mathcal{T}_0$.

Merging multiple contour trees directly is difficult, so we combine the contour trees by essentially sewing their join and split trees together along piece boundaries and then running a procedure of Carr *et al.* [14]. Within a recursive computation, we *trim* excess vertices from the contour tree before sending them to their recursive parent call in order to minimize the amount of communication necessary between rounds of computation.

4.2 Recursive Construction

We now describe our recursive algorithm in more detail. Throughout our explanation, let n be the number of triangles in the top level triangulation $\mathbb{M}$. The input to a recursive call of our algorithm is a rectangle $\square$, a triangulation $\mathbb{M}_\square = (V_\square, E_\square, F_\square)$ with one boundary component such that $\square \cap \mathbb{M}$ fits entirely within $\mathbb{M}_\square$, a piecewise linear height function $h_\square : \mathbb{M}_\square \to \mathbb{R}$, and a contiguous set of machines $I_\square = \{i_0, i_0+1, \dots\}$ storing $\mathbb{M}_\square$. Our algorithm is summarized as Algorithm 2. We call vertex v of $\mathbb{M}_\square$ a boundary vertex of $\square$ if v lies on any triangle properly intersecting $\square$. Our goal is to compute the distributed contour tree $\mathcal{T}_\square$ on machines $I_\square$. We will also create tree $\mathcal{T}'$ which will be sent as the 'return value' of a recursive call. Tree $\mathcal{T}'$ contains all boundary vertices of $\mathbb{M}_\square$, and all saddle vertices v for which there exists a boundary vertex in at least three of the connected components of $\mathbb{M}_\square$ separated by the contour through v.

Algorithm 2 $\texttt{ContourTree}(\square, \mathbb{M}_\square, h_\square, I_\square)$

```
1: if $|I_\square| = 1$ then
2:     Compute contour tree $\mathcal{T}$ locally.
3: else
4:     Divide $\square$ into $k = \min\{n^\varepsilon, |I_\square|\}$ rectangles $\{\square_1, \dots, \square_k\}$.
5:     for all $\square_i$ in parallel do
6:         $\mathbb{M}_{\square_i}$ := triangles of $\mathbb{M}_\square$ intersecting $\square_i$.
7:         $h_{\square_i}$ := restriction of $h_\square$ to $\mathbb{M}_{\square_i}$.
8:         $I_{\square_i}$ := a set of $|I|/k$ machines.
9:         $\mathcal{T}'_{\square_i}$ := ContourTree($\square_i, \mathbb{M}_{\square_i}, h_{\square_i}, I_{\square_i}$).
10:    end for
11:    Combine all trees $\mathcal{T}'_{\square_i}$ to make contour tree $\mathcal{T}$.
12: end if
13: Trim $\mathcal{T}$ to make tree $\mathcal{T}'$, and return $\mathcal{T}'$.
```

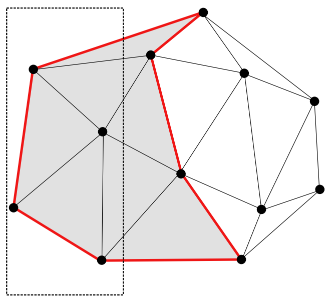

Figure 5. One rectangle $\square_i$ and its associated triangles $\mathbb{M}_i$.

Tree $\mathcal{T}'$ is created from the top level structure of the distributed contour tree $\mathcal{T}_\square$ by augmenting the top level structure with any boundary vertices that are not already present, iteratively removing leaves that are not boundary vertices, and iteratively splicing degree-2 vertices that are not boundary vertices, in that order. The top level subtree given in Figure 4 shows the result of this procedure if α_2, α_3, and y_2 are boundary vertices. Let $\Sigma_\square$ denote the terrain described by $\mathbb{M}_\square$ and $h_\square$. The creation of $\mathcal{T}'$ mirrors the changes to $\mathcal{T}_\square$ that take place as one iteratively sets the height of all points on $\mathbb{M}_\square$ mapping to a single leaf edge to be equal to the height of the edge's non-leaf vertex. Let $\Sigma'_\square$ be the terrain created by this iterative procedure. Observe that the triangles of $\mathbb{M}_\square$ properly intersecting $\square$ have their height values unchanged between $\Sigma_\square$ and $\Sigma'_\square$. The portions of $\mathcal{T}_\square$ removed from $\mathcal{T}'$ can be represented as rooted subtrees and vines. The top level structures for these rooted subtrees and vines are permanently stored on machines in $I_\square$. Note that $\mathcal{T}'$ is still augmented with every boundary vertex, although these boundary vertices may be degree-2.

LEMMA 4.1. *The number of vertices in $\mathcal{T}'$ is $O(\sqrt{n})$.*

PROOF. By our assumption that each line in $\mathbb{R}^2$ crosses $O(\sqrt{n})$ edges, the boundary of $\mathbb{M}_\square$ crosses $O(\sqrt{n})$ triangles and there are therefore $O(\sqrt{n})$ boundary vertices. Every degree-1 or degree-2 vertex in $\mathcal{T}'$ is a boundary vertex. The lemma follows. $\square$

We now describe how to compute the distributed contour tree $\mathcal{T}_\square$. If $|I_\square| = 1$, then we compute $\mathcal{T}_\square$ sequentially using the algorithm of Carr *et al.* [14]. Otherwise, we do so recursively. Let n' be the number of vertices of $\mathbb{M}$ strictly interior to $\square$. We begin by dividing $\square$ along vertical lines into $k = \min\{n^\varepsilon, |I_\square|\}$ rectangles $\{\square_1, \dots, \square_k\}$. These lines are chosen so that there exist at most $\lceil n'/k \rceil$ vertices of $\mathbb{M}$ in each rectangle $\square_i$. Finding the lines can be done in $O(1)$ rounds, $O(s \log s)$ time, and $O(n' \log n')$ work by sorting the vertices inside $\square$ by their x-coordinate using a procedure of Goodrich [24] and looking at the $i \cdot \lceil n'/k \rceil$-th vertex for each i in this order. Now, let Π be a function from $F_\square$ to subsets of $\{1, \dots, k\}$ where for each triangle Δ of $\mathbb{M}_\square$, $\Pi(\Delta)$ is the subset of rectangles $\square_i$ that intersect Δ. We run $\texttt{Distribute}(F_\square, I_\square, \Pi)$ to distribute copies of the triangles to disjoint sets of machines.

Let $I_{\square_i}$ for $i \in \{1, \dots, k\}$ denote the disjoint subsets of machines that receive triangles from the $\texttt{Distribute}$ primitive. Let $\mathbb{M}_{\square_i}$ be the set of triangles sent to machines $I_{\square_i}$. See Figure 5. We recursively compute distributed contour trees of each triangulation $\mathbb{M}_{\square_i}$ using the natural restrictions of h to those triangulations. In the following lemma, we prove that the distribution step above is doable in the MPC model. In the sequel, we describe how to combine the trees returned by each recursive call to create the top level subtree for $\mathcal{T}_\square$.

LEMMA 4.2. *Our algorithm meets the preconditions of the* `Distribute` *primitive.*

PROOF. By assumption, $I_\square$ is a contiguous set of machines as required. We meet the requirement that $k = O(s^{1/2})$ as $s = \Omega(n^{1/2+\varepsilon})$. Similarly, we meet the requirement that $k \leq |I|$. Each rectangle $\square_i$ contains $O(n'/k)$ points, so it receives $O(n'/k)$ *non*-boundary triangles. Further, by the assumption that each line in $\mathbb{R}^2$ intersects at most $O(\sqrt{n})$ edges, there are at most $O(\sqrt{n}) = O(n'/k)$ boundary triangles for each $\square_i$. Together, these facts imply that for each $i \in \{1, \dots, k\}$, there are at most $O(n'/k)$ triangles Δ for which $\Pi(\Delta) = i$ as required. Finally, each triangle can only be copied at most $n^\varepsilon = O(s)$ times as required. $\square$

4.3 Combining Multiple Contour Trees

Let $\mathcal{T}_{\square_i}$ be the contour tree for $\mathbb{M}_{\square_i}$, and let $\mathcal{T}'_{\square_i}$ be the tree returned by the recursive procedure computing $\mathcal{T}_{\square_i}$. We send all of these trees to a single machine $\beta \in I_\square$ in $O(1)$ rounds of computation. By Lemma 4.1, each such tree has size $O(\sqrt{n})$, so they all fit on machine β simultaneously. The rest of the computation occurs sequentially on machine β.

Recall, $\mathcal{T}'_{\square_i}$ is the contour tree for a surface $\Sigma'_{\square_i}$. We create the top level structure for $\mathcal{T}_\square$ using the following procedure: Let $\Sigma''_\square$ be the terrain that results from replacing the portion of $\Sigma_\square$ within each rectangle $\square_i$ with the portion of $\Sigma'_{\square_i}$ inside $\square_i$. We say vertex v is *important* if it appears as a vertex of any tree $\mathcal{T}'_{\square_i}$.

We use the following procedure to compute the join tree $\mathcal{J}''_\square$ of $\Sigma''_\square$. We compute the join tree $\mathcal{J}'_{\square_i}$ for each contour tree $\mathcal{T}'_{\square_i}$. Next, we sort the entire collection of important vertices in increasing order of height. We then use a set union-find data structure [16] to track the sublevel sets as we consider vertices of increasing height values. The sets of the union-find data structure will be collections of important vertices, and a find operation on an important vertex v will return the highest important vertex v' in the same sublevel set component as v. For each important vertex v in increasing order by height, we look at its down neighbors from each of the join trees $\mathcal{J}'_{\square_i}$ that contain v and do a find operation. The down neighbors of v in $\mathcal{J}''_\square$ will be all the vertices returned by the find operations.

LEMMA 4.3. *The previous algorithm correctly computes the join tree $\mathcal{J}''_\square$ of $\Sigma''_\square$.*

PROOF. We prove the lemma by induction on the increasing heights of important vertices. Let v be an important vertex, and consider any sublevel set component C in $\Sigma''_\square$ of height $h(v) - \delta$ for an arbitrarily small δ such that C lies within the sublevel set component of v (assuming C exists). Vertex v contains a neighbor u that lies in C. Let $\mathbb{M}_{\square_i}$ be a set of triangles containing edge (v, u). Let v' be the highest important vertex of $\Sigma'_{\square_i}$ with height strictly below v that shares a sublevel component of $\Sigma'_{\square_i}$ with v and u. Vertex v' is a down neighbor of v in $\mathcal{J}'_{\square_i}$ that lies in component C. By induction, the find operation on v' will return the highest important vertex of C, which the algorithm will correctly assign as a down neighbor of v in $\mathcal{J}''_\square$. $\square$

The split tree $\mathcal{S}''_\square$ of $\Sigma''_\square$ can be computed using a similar procedure. Finally, we use the procedure of Carr *et al.* [14] to create the contour tree $\mathcal{T}''_\square$ of $\Sigma''_\square$. Tree $\mathcal{T}''_\square$ is the top level structure used to store $\mathcal{T}_\square$ in a distributed manner.

LEMMA 4.4. *Procedure* `ContourTree` *takes $O(1)$ rounds of computation, $O(s \log s)$ time, and $O(n \log n)$ work.*

PROOF. The number of levels of recursion of the procedure is $O(\log_{n^\varepsilon} n) = O(1)$, and each level of recursion uses only $O(1)$ rounds of computation. For time and work, the most expensive operation is building the join and split trees to compute a contour tree $\mathcal{T}''_\square$. These operations require sorting $O(s)$ vertices in $O(s \log s)$ time. Summing over all the vertices stored on all machines, the total amount of work is $O(n \log n)$. $\square$

We thus have the following.

THEOREM 4.5. *Let $\mathbb{M} = (V, E, F)$ be a triangulation in $\mathbb{R}^2$ such that every line in $\mathbb{R}^2$ intersects $O(\sqrt{n})$ edges of $\mathbb{M}$ where $n = |V|$. Let $h : \mathbb{M} \to \mathbb{R}$ be a piecewise linear height function on $\mathbb{M}$. The distributed contour tree $\mathcal{T}$ of h can be built in the MPC model in $O(1)$ computation rounds, $O(s \log s)$ time, and $O(n \log n)$ work, assuming $s = \Omega(n^{1/2+\varepsilon})$ for some constant $\varepsilon > 0$.*

5. CONCLUSION

We described two algorithms for the MPC model that aid in terrain analysis. The first was a Las Vegas algorithm to compute a Delaunay triangulation of an arbitrary point set in $\mathbb{R}^2$. With high probability, it runs in $O(1)$ rounds of computation, and it uses only $O(s \log^2 n)$ time and $O(n \log n)$ work. The second algorithm deterministically constructs a contour tree of a triangulation $\mathbb{M}$, assuming the machines have sufficient memory and $\mathbb{M}$ is well-formed. This algorithm required only $O(1)$ rounds of computation, $O(s \log s)$ time, and $O(n \log n)$ work. We leave the removal of these restrictions from our contour tree construction algorithm as an interesting open problem. For an arbitrary triangulation $\mathbb{M}$, a possible approach is to build a small, well-balanced planar separator with small boundary, build the contour tree for each piece in parallel, and sew them along the boundary. However, computing a separator in the MPC model appears difficult, and will require new ideas.

Acknowledgements. Work on this paper is supported by DARPA grant FA8750-16-2-0173, the NSF under grants CCF-1161359, IIS-1408846, CCF-1408784, IIS-1447554, CCF-1546392, and CCF-1513816, by an ARO grant W911NF-15-1-0408, and by Grant 2012/229 from the U.S.-Israel Binational Science Foundation

6. REFERENCES

[1] A. Acharya and V. Natarajan. A parallel and memory efficient algorithm for constructing the contour tree. In *2015 IEEE Pacific Vis. Symp.*, pages 271–278, 2015.

[2] P. K. Agarwal, L. Arge, and K. Yi. I/O-efficient construction of constrained Delaunay triangulations. In *Proc. 13th Euro. Symp. Algo.*, pages 355–366, 2005.

[3] P. K. Agarwal, L. Arge, and K. Yi. I/O-efficient batched union-find and its applications to terrain analysis. *ACM Trans. Alg.*, 7(1):11:1–11:21, 2010.

[4] P. K. Agarwal, K. Fox, K. Munagala, and A. Nath. Parallel algorithms for constructing range and nearest-neighbor searching data structures. In *Proc. 35th Symp. Princ. Database Syst.*, pages 429–440, 2016.

[5] P. K. Agarwal, T. Mølhave, M. Revsbæk, I. Safa, Y. Wang, and J. Yang. Maintaining contour trees of dynamic terrains. In *31st Int. Symp. Comp. Geom.*, pages 796–811, 2015.

[6] T. Akidau, R. Bradshaw, C. Chambers, S. Chernyak, R. J. Fernandez-Moctezuma, R. Lax, S. McVeety, D. Mills, F. Perry, E. Schmidt, and S. Whittle. The Dataflow model: A practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data processing. *Proc. VLDB Endow.*, 8(12):1792–1803, 2015.

[7] A. Andoni, A. Nikolov, K. Onak, and G. Yaroslavtsev. Parallel algorithms for geometric graph problems. In *Proc. 46th ACM Annu. Symp. Theory Comput.*, pages 574–583, 2014.

[8] L. Arge, M. Revsbæk, and N. Zeh. I/O-efficient computation of water flow across a terrain. In *Proc. 26th ACM Symp. Comp. Geom.*, pages 403–412, 2010.

[9] F. Aurenhammer, R. Klein, and D.-T. Lee. *Voronoi Diagrams and Delaunay Triangulations.* World Scientific, 2013.

[10] P. Beame, P. Koutris, and D. Suciu. Communication steps for parallel query processing. In *Proc. 32nd Symp. Princ. Database Syst.*, pages 273–284, 2013.

[11] K. G. Bemis, D. Silver, P. A. Rona, and C. Feng. Case study: a methodology for plume visualization with application to real-time acquisition and navigation. In *IEEE Vis.*, pages 481–484, 2000.

[12] G. E. Blelloch, J. C. Hardwick, G. L. Miller, and D. Talmor. Design and implementation of a practical parallel Delaunay algorithm. *Algorithmica*, 24(3-4):243–269, 1999.

[13] P. Bose and L. Devroye. On the stabbing number of a random Delaunay triangulation. *Comput. Geom.*, 36(2):89–105, 2007.

[14] H. Carr, J. Snoeyink, and U. Axen. Computing contour trees in all dimensions. *Comput. Geom.*, 24(2):75–94, 2003.

[15] B. Chazelle and J. Friedman. A deterministic view of random sampling and its use in geometry. *Combinatorica*, 10(3):229–249, 1990.

[16] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms (3. ed.).* MIT Press, 2009.

[17] A. Crauser, P. Ferragina, K. Mehlhorn, U. Meyer, and E. A. Ramos. Randomized external-memory algorithms for line segment intersection and other geometric problems. *Int. J. Comput. Geometry Appl.*, 11(3):305–337, 2001.

[18] A. Danner, T. Mølhave, K. Yi, P. K. Agarwal, L. Arge, and H. Mitásová. Terrastream: from elevation data to watershed hierarchies. In *15th ACM Inter. Symp. Geo. Inf. Sys.*, page 28, 2007.

[19] M. de Berg, M. van Kreveld, M. Overmars, and O. Cheong. *Computational geometry: Algorithms and Applications.* Springer, 3rd edition, 2000.

[20] J. Dean and S. Ghemawat. MapReduce: simplified data processing on large clusters. *Comm. ACM*, 51(1):107–113, 2008.

[21] A. Eldawy, L. Alarabi, and M. F. Mokbel. Spatial partitioning techniques in SpatialHadoop. *Proc. VLDB Endow.*, 8(12):1602–1613, 2015.

[22] A. Eldawy and M. F. Mokbel. SpatialHadoop: A MapReduce framework for spatial data. In *Proc. 31st IEEE Int. Conf. Data Eng.*, pages 1352–1363, 2015.

[23] M. T. Goodrich. Randomized fully-scalable BSP techniques for multi-searching and convex hull construction (preliminary version). In *Proc. 8th ACM-SIAM Annu. Symp. Disc. Alg.*, pages 767–776, 1997.

[24] M. T. Goodrich. Communication-efficient parallel sorting. *SIAM J. Comput.*, 29(2):416–432, 1999.

[25] M. T. Goodrich, N. Sitchinava, and Q. Zhang. Sorting, searching, and simulation in the MapReduce framework. In *Proc. 22nd Int. Symp. Alg. Comput.*, pages 374–383, 2011.

[26] D. Haussler and E. Welzl. ε-nets and simplex range queries. *Disc. Comp. Geom.*, 2:127–151, 1987.

[27] H. Karloff, S. Suri, and S. Vassilvitskii. A model of computation for MapReduce. In *Proc. 21st ACM-SIAM Annu. Symp. Disc. Alg.*, pages 938–948, 2010.

[28] G. Malewicz, M. H. Austern, A. J. C. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski. Pregel: a system for large-scale graph processing. In *Proc. 2010 ACM SIGMOD Int. Conf. Manage. Data*, pages 135–146, 2010.

[29] J. Matoušek. Reporting points in halfspaces. *Comput. Geom. Theory Appl.*, 2(3):169–186, 1992.

[30] N. Max, R. Crawfis, and D. Williams. Visualization for climate modeling. *IEEE Comp. Graph. Appl.*, 13(4):34–40, 1993.

[31] D. Morozov and G. H. Weber. Distributed merge trees. In *ACM SIGPLAN Symp. on Princ. and Practice of Parallel Prog.*, pages 93–102, 2013.

[32] D. Morozov and G. H. Weber. Distributed contour trees. In *Topological Methods in Data Analysis and Vis. III*, pages 89–102. 2014.

[33] S. P. Tarasov and M. N. Vyalyi. Construction of contour trees in 3D in $O(n \log n)$ steps. In *Proc. 14th Annu. Symp. Comput. Geom.*, pages 68–75, 1998.

[34] L. G. Valiant. A bridging model for parallel computation. *Comm. ACM*, 33(8):103–111, 1990.

[35] M. van Kreveld, R. van Oostrum, C. Bajaj, V. Pascucci, and D. Schikore. Contour trees and small seed sets for isosurface traversal. In *Proc. 13th Annu. Symp. Comput. Geom.*, pages 212–220, 1997.

[36] T. White. *Hadoop: The Definitive Guide.* O'Reilly Media, Inc., 2012.

[37] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. Spark: Cluster computing with working sets. In *Proc. 2nd USENIX Conf. Hot Topics Cloud Comput.*, pages 10–10, 2010.