Convex Hulls and Triangulations of Planar Point Sets on the Congested Clique

Jesper Jansson
Kyoto University

Christos Levcopoulos
Lund University

Andrzej Lingas
Lund University

CCCG 2023
Montreal, Quebec, Canada

Congested Clique Model

The model of congested clique focuses on the communication cost and ignores that of local computation (Lotker *et al.*, 2003)

Originally, it has been applied to dense graph problems. The n nodes of a clique network one-to-one correspond to vertices of the input graph and have information about the neighborhood of the corresponding vertex initially. In each round, each node can:

1. send an $O(\log n)$ bit message to each other node, the messages to different nodes can be different;
2. receive an $O(\log n)$ bit message from each other node;
3. perform unlimited local computations on own data.

The objective is to minimize the number of rounds.

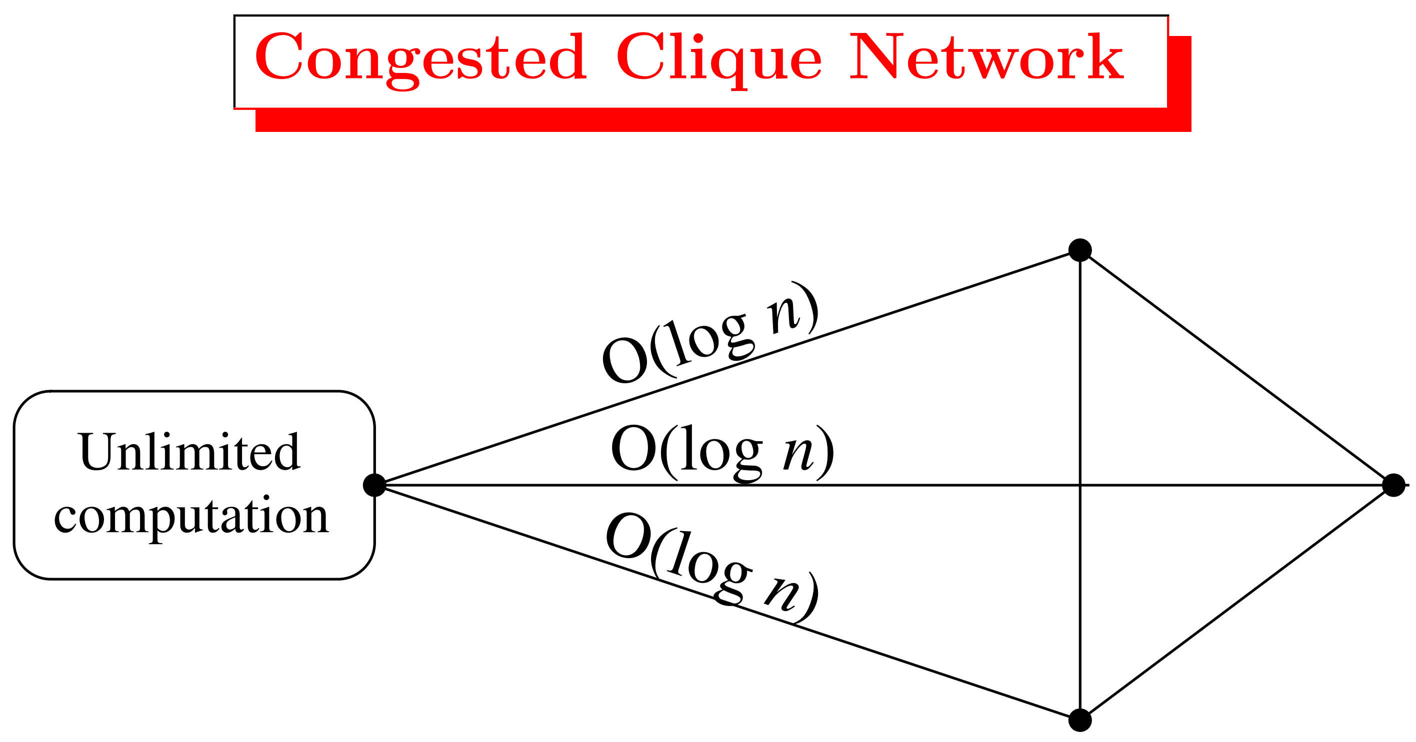

Figure 1: An example of congested clique network, each node is supposed to hold a distinct piece of the input initially.

Congested Clique Model 2

For several dense graph problems, e.g., minimum spanning tree, round efficient, often even $O(1)$-round algorithms have been designed in this model (Robinson, 2022).

One has also designed round efficient algorithms for matrix multiplication (Censor-Hillel *et al.*, 2015), sorting and routing (Lenzen, 2013) in this model. E.g., in case of sorting, one assumes that each of the n nodes initially stores a distinct batch of n $O(\log n)$ bit keys. The target is to sort the n^2 keys.

We extend this approach to include geometric problems on sets of n^2 points with $O(\log n)$ bit coordinates in the Euclidean plane. Thus, each node holds initially a batch of n points with $O(\log n)$ bit coordinates in the plane. The target is to compute the convex hull or a triangulation, or the Voronoi diagram of the set S of n^2 points.

Our Contributions

Input: A set S of n^2 points with $O(\log n)$ bit coordinates, each node holds a batch of n input points.

- An implementation of Quick Convex Hull for S on congested clique in $O(h)$ rounds, where h is the size of the convex hull of S.
- A refined algorithm for the convex hull of S on congested clique running in $O(\log n)$ rounds.
- An algorithm for a triangulation of S running in $O(\log^2 n)$ rounds.

Quick Convex Hull on Congested Clique

1. Sort the n^2 points in S by their x-coordinates so each node receives a subsequence consisting of n consecutive points in S.
2. Each node sends the first point and the last point in its subsequence to the other nodes.
3. Each node computes the same points p_{max} of the maximum x-coordinate and p_{min} of the minimum x-coordinate in S. Next, it decomposes its sorted subsequence into the upper subsequence over (p_{max}, p_{min}) and the lower one below (p_{max}, p_{min}).
4. $QuickUpperHull(p_{min}, p_{max})$
5. $QuickLowerHull(p_{min}, p_{max})$
6. Rearrange the output by using round efficient routing.

procedure $QuickUpperHull(p, r)$

- Each node u determines the set S_u of points in its upper subsequence that have x-coordinates between those of p and r and lie above or on (p, r). If $S_u \neq \emptyset$ the node sends a point in S_u with the largest y-coordinate to the node holding p (master).
- If the master hasn't received any point sent in Step 1 then it proclaims p, r to be vertices of the upper hull. Next, it pops a call of $QuickUpperHull$ from the top of a stack of recursive calls. If the stack is empty it terminates $QuickUpperHull(p_{min}, p_{max})$.
- If the master has received some points sent in Step 1 than it picks a point q of maximum y-coordinate among them. Next, it activates $QuickUpperHull(p, q)$ and puts $QuickUpperHull(q, r)$ on the top of the stack.

The idea of $QuickUpperHull(p, r)$

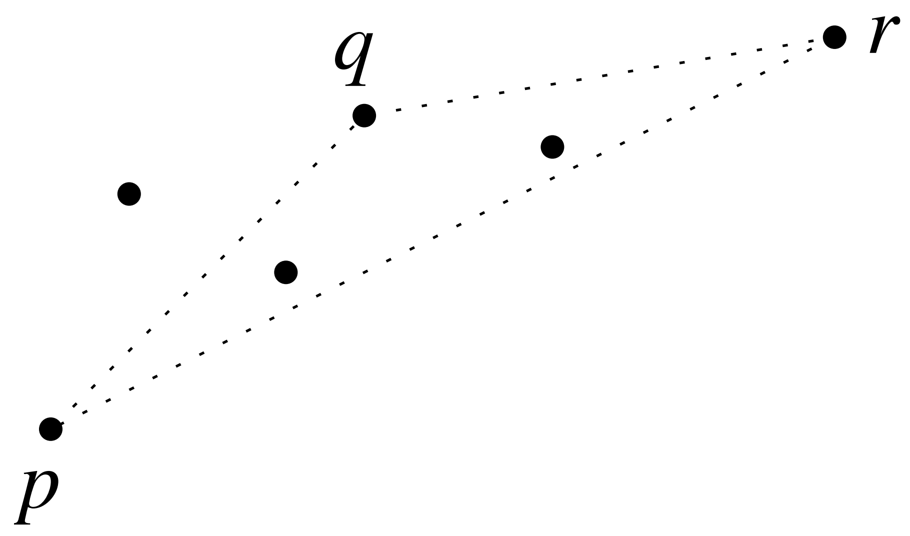

Figure 2: The point q of largest y coordinate between the points p and r is selected in order to call $QuickUpperHull(p, q)$ and $QuickUpperHull(q, r)$.

The procedure $QuickLowerHull(p, r)$ is defined analogously.

Each step of Quick Convex Hull but for $QuickUpperHull(p_{min}, p_{max})$ and $QuickLowerHull(p_{min}, p_{max})$ can be done in $O(1)$ rounds on the congested n-clique. In particular, the sorting and routing can be done in $O(1)$ rounds by the results of Lenzen. Similarly, each step of $QuickUpperHull(p, r)$ and $QuickLowerHull(p, r)$, but for recursive calls, can be done in $O(1)$ rounds. Since each non-leaf call of $QuickUpperHull(p, r)$ and $QuickLowerHull(p, r)$ results in a new vertex of the convex hull, their total number does not exceed the number h of vertices on the convex hull of S.

Theorem 1 *The convex hull of the set S of n^2 points can be computed in $O(h)$ rounds on the congested n-clique.*

An $O(\log n)$-round Algorithm for Convex Hull

The algorithm uses refined procedures for Upper Hull and Lower Hull, $NewUpperHull(S)$, $NewLowerHull(S)$, respectively.

The procedure $NewUpperHull(S)$ lets each node ℓ construct the upper hull H_ℓ of its batch of at most n points in the upper-hull subsequence locally.

The crucial step of $NewUpperHull(S)$ is a parallel computation of bridges between all pairs H_ℓ, H_m, $\ell \neq m$, of the constructed upper hulls by parallel calls to the procedure $Bridge(H_\ell, H_m)$.

Based on the bridges between H_ℓ and the other upper hulls H_m, each node ℓ can determine which of the vertices of H_ℓ belong to the upper hull of S (see Lemma 1).

Bridges

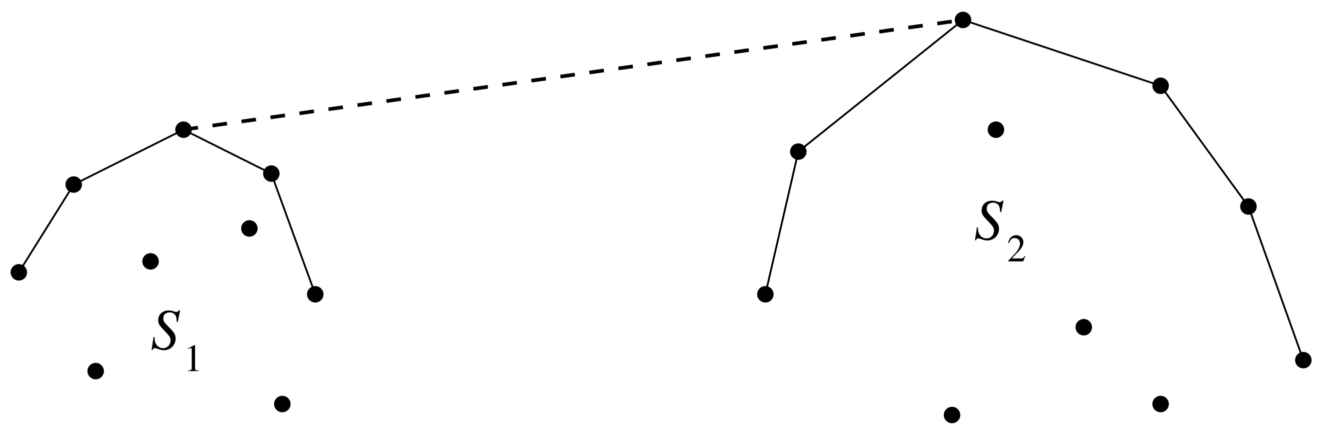

Figure 3: An example of the bridge between the upper hulls of S_1 and S_2.

Lemma 1

Lemma 1 *For $\ell \in [n]$, let H_ℓ be the upper hull of the upper-hull subsequence of S assigned to the node ℓ. A vertex v of H_ℓ is not a vertex of the upper hull of S if and only if it lies below a bridge between H_ℓ and H_m, where $\ell \neq m$, or there are two bridges between H_ℓ and H_s, H_t, respectively, where $s < \ell < t$, such that they touch v and form an angle of less than 180 degrees at v.*

Illustration to Lemma 1

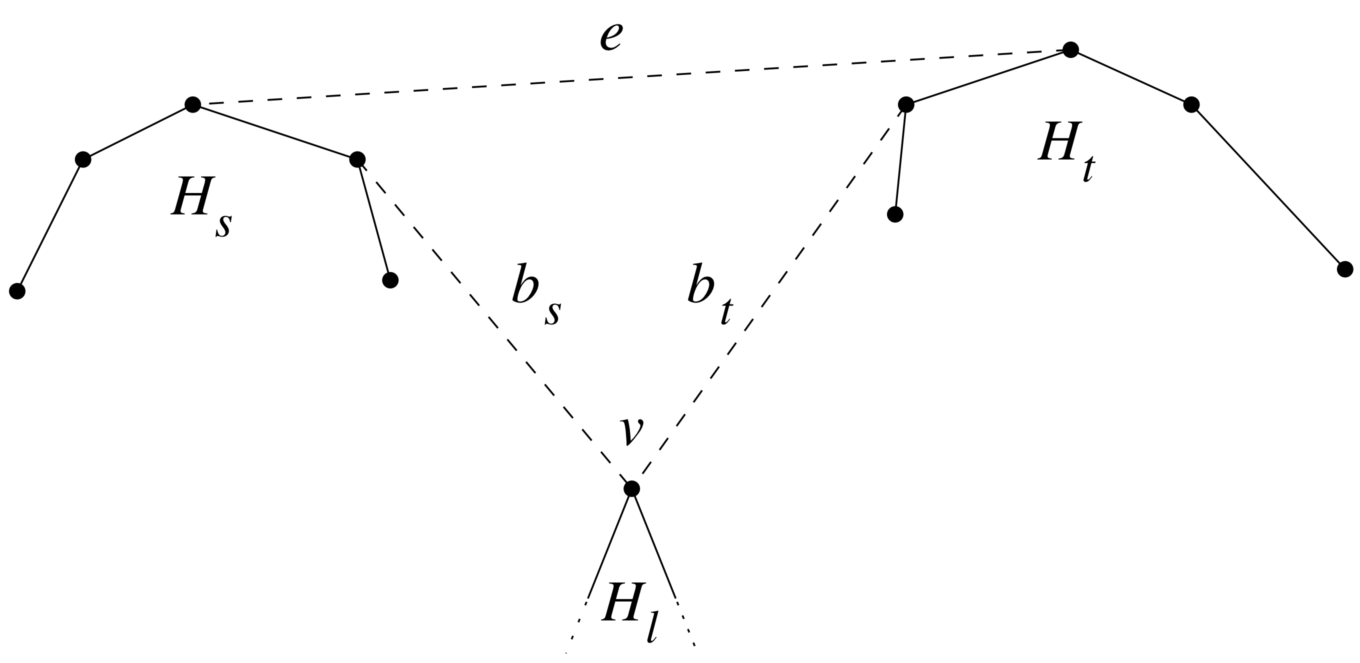

Figure 4: The final case in Lemma 1.

Lemma 2

The recursive procedure *Bridge* is based on the following folklore lemma.

Lemma 2 *Let $S_1,\ S_2$ be two n-point sets in the Euclidean plane separated by a vertical line. Let $H_1,\ H_2$ be the upper hulls of $S_1,\ S_2$, respectively. Suppose that each of H_1 and H_2 has at least three vertices. Next, let $m_1,\ m_2$ be the median vertices of $H_1,\ H_2$, respectively. Suppose that the segment connecting m_1 with m_2 is not the bridge between H_1 and H_2. Then, nome of the vertices on H_1 either to the left or to the right of m_1, or none of the vertices on H_2 either to the left or to the right of m_2 can be an endpoint of the bridge between H_1 and H_2.*

Illustration of Lemma 2

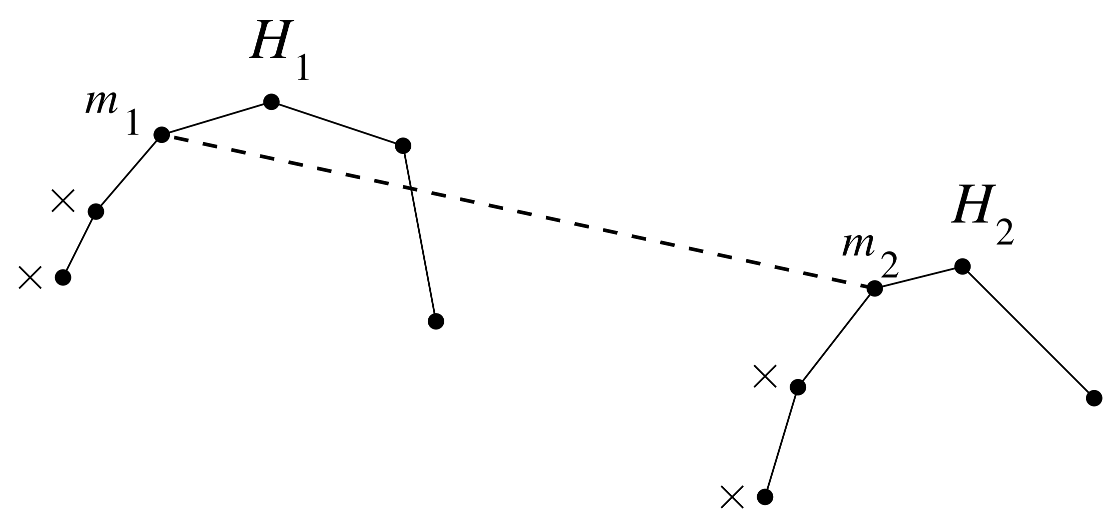

Figure 5: An illustration to Lemma 2 on which the procedure Bridge is based.

procedure *Bridge*(H'_ℓ, H'_m)

Input: Two continuous pieces H'_ℓ, H'_m of the upper hull of the points assigned to ℓ and m, respectively.
Output: The bridge between H'_ℓ and H'_m.

1. If H'_ℓ or H'_m has at most two vertices then compute the bridge between H'_ℓ and H'_m by binary search and mark all the vertices below it as not qualifying for the upper hull. .
2. Find a median m_1 of H'_ℓ and a median m_2 of H'_m.
3. If the straight line passing through m_1 and m_2 is a supporting line for both H'_ℓ and H'_m then mark all the vertices below between ℓ and m as not qualifying for the upper hull.
4. Else $Bridge(H''_\ell, H''_m)$, where $H'_\ell = H''_\ell$ and H''_m is obtained from H'_m by removing vertices on a side of m_2 or *vice versa* by Lem. 2.

Time Analysis of New Convex Hull

The procedure $NewLowerHull(H'_\ell, H'_m)$ is defined analogously. Roughly, all steps but for the n^2 parallel calls of the *Bridge* procedure can be done in $O(1)$ rounds.

By Lemma 2,the recursion depth of the *Bridge* procedure is logarithmic. The nodes ℓ and m need to exchange $O(\log n)$ $O(\log n)$-bit messages in order to implement $Bridge(H_\ell, H_m)$. It follows that all the n^2 calls of $Bridge(H_\ell, H_m)$ can be implemented in parallel in $O(\log n)$ rounds.

Theorem 2 *The convex hull of the set S of the n^2 input points with $O(\log n)$-bit coordinates in the Euclidean plane can be computed in $O(\log n)$ rounds on the congested clique.*

procedure $Triangulation(S)$

1. Sort the points in S by their x-coordinates so each node receives a subsequence consisting of n consecutive points in S, in the sorted order.
2. Each node q constructs a triangulation $T_{q,q}$ of the points in its sorted subsequence locally.
3. For $1 \leq p < q \leq n$, $T_{p,q}$ will denote the already computed triangulation of the points in the sorted subsequence held in the nodes p through q. For $i = 0, \log n - 1$, in parallel, for $j = 1, 1 + 2^{i+1}, 1 + 2 \cdot 2^{i+1}, 1 + 3 \cdot 2^{i+1}, ...$ the union of the triangulations $T_{j,j+2^i-1}$ and $T_{j+2^i,j+2^{i+1}-1}$ is transformed to a triangulation $T_{j,j+2^{i+1}-1}$ by $Merge(i, j)$.

procedure $Merge(i, j)$

Input: Triangulations $T_{j,j+2^i-1}$ and $T_{j+2^i,j+2^{i+1}-1}$.

Output: A triangulation $T_{j,j+2j+1-1}$.

1. Compute the bridges between the convex hulls of $T_{j,j+2^i-1}$ and $T_{j+2^i,j+2^{i+1}-1}$. Determine the polygon P formed by the bridges between the convex hulls of $T_{j,j+2^i-1}$ and $T_{j+2^i,j+2^{i+1}-1}$, the right side of the convex hull of $T_{j,j+2^i-1}$, and the left side of the convex hull of $T_{j+2^i,j+2^{i+1}-1}$ between the bridges.
2. $Triangulate(P, j, j + 2^{i+1} - 1)$

procedure $Triangulate(P, p, q)$

1. The nodes $p, ..., q$ determine the node holding the median vertex v of the longest convex chain on the perimeter of P and v sends the coordinates of v and the adjacent vertices to $p, ..., q$.
2. The nodes holding vertices of the convex chain opposite to that with v determine if they hold vertices u that could be connected by a segment with v within P. If so, they send such a vertex u to the node holding v.
3. he node holding v selects one of the received vertices u as the mate and sends its coordinates to the other nodes p through q.
4. The nodes $p, ..., q$ split P into subpolygons $P_1,\ P_2$ by (v, u) and move their edges to consecutive destinations $p, ..., r_1 \leq r_2, ..., q$.
5. In parallel, $Triangulate(P_1, p, r_1)$ and $Triangulate(P_2, r_2, q)$.

The idea of Triangulate

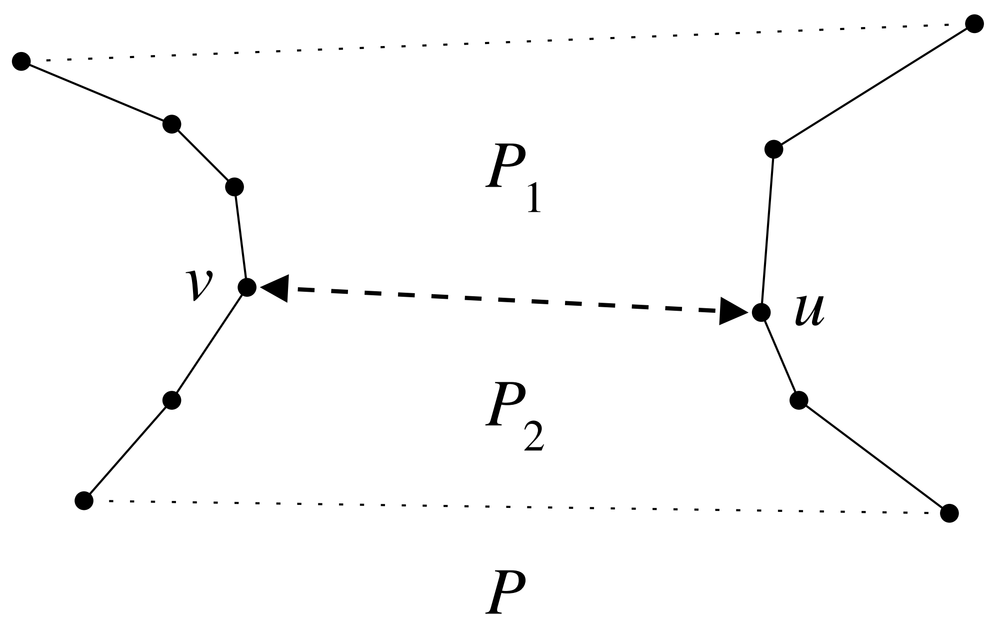

Figure 6: The recursive procedure Triangulate finds a diagonal between the median v of the longer convex chain and a vertex u on the other convex chain in order to split the polygon P into subpolygons P_1 and $P - 2$.

Time Analysis

To verify if (v, u) is within P the relevant node checks if this segment is within the intersection of the union of the half-planes induced by the edges adjacent to v with the union of the half-planes induced by the edges adjacent to u.

All steps, but for those involving calls $Merge$, $Triangulate$ and computing the bridges, require $O(1)$ rounds. The bridges can be computed in $O(\log n)$ rounds by our prior algorithm. The recursive depth of $Triangulate$ is $O(\log n)$. Hence, $Triangulate$ and $Merge$ can be implemented in $O(\log n)$ rounds. The parallel calls $Merge(i, j)$ for fixed i can be done in $O(\log n)$ rounds by global routing.

Theorem 3 *A triangulation of the set S of the n^2 input points can be computed in $O(\log^2 n)$ rounds on the congested clique.*

Voronoi Diagram on Congested Clique

The primary difficulty in the design of efficient parallel algorithms for the Voronoi diagram of a planar point set using a divide-and-conquer approach is the efficient parallel merging of Voronoi diagrams.

In the full version of this paper, we show:

Theorem 4 *The Voronoi diagram of* n^2 *points with* $O(\log n)$*-bit coordinates drawn uniformly at random from a unit square in the Euclidean plane can be computed within the square with high probability in* $O(1)$ *rounds on the congested clique.*

This research was partially supported by Swedish Research Council grants 621-2017-03750 and 2018-04001, and JSPS KAKENHI JP20H05964.

Thank you for your attention