A Time-Space Trade-off for Triangulations of Points in the Plane*

Hee-Kap Ahn[1], Nicola Baraldo[2], Eunjin Oh[1], and Francesco Silvestri[2]

[1] Pohang University of Science and Technology, Korea.
{heekap,jin9082}@postech.ac.kr
[2] University of Padova, Italy.
nicola.baraldo@gmail.com, silvestri@dei.unipd.it

Abstract. In this paper, we consider time-space trade-offs for reporting a triangulation of points in the plane. The goal is to minimize the amount of working space while keeping the total running time small. We present the first multi-pass algorithm on the problem that returns the edges of a triangulation with their adjacency information. This even improves the previously best known random-access algorithm.

1 Introduction

There are two optimization goals in the design of algorithms: the time complexity and the space complexity. However, one cannot achieve both goals at the same time in general. This can be seen in a time-space tradeoff of algorithmic efficiency that an algorithm has to use more space to improve its running time and it has to spend more time with less amount of space. With this reason, time-space trade-offs for a number of problems were considered even as early as in 1980s. For example, Frederickson presented optimal time-space trade-offs for sorting and selection problems in 1987 [11]. After this work, a significant amount of research has been done for time-space trade-offs in the design of algorithms.

In this paper, we consider time-space trade-offs for one of fundamental geometric problems, reporting a triangulation of points in the plane. We assume that the points are given in a read-only memory. This assumption has been considered in applications where the input is required to be retained in its original state. Many time-space tradeoffs for fundamental problems have been studied under this read-only assumption. For instance, a few read-only sorting algorithms have been presented under the assumption [6, 14].

There are two typical access models to the read-only input, a *random-access model* and a *multi-pass model*. In the *multi-pass model*, the only way to access elements in the input array is to scan the array from the beginning, and algorithms are allowed to make multiple passes over the input. A single pass is a special case of the multi-pass model. Multi-pass algorithms are more restrictive than algorithms under the *random-access* for any element in the input array.

* This work was supported by the NRF grant 2011-0030044 (SRC-GAIA) funded by the government of Korea.

The multi-pass model has applications where large data sets are stored somewhere such as an external memory and it is more efficient in I/O to read them sequentially in a few passes. Multi-pass algorithms have been studied recently in areas including geometry [1, 15] and graphs [9, 10].

The goal of our problem is to minimize the amount of working space while keeping the total running time small. More precisely, we are allowed to use $O(s)$ words as working space in addition to the memory for input and output for a positive integer parameter s which is determined by users. We assume that a word is large enough to store a number and a pointer. While processing input, we send the answer to a write-only output stream without repetition. An algorithm designed in this setting is called an *s-workspace algorithm*.

1.1 Related Works

A triangulation of a set S of n points in the plane is defined to be a maximal subdivision of the plane whose vertices are in S and faces are triangles, except for the unbounded face. The unbounded face of a triangulation of S is the region outside of the convex hull of S. Thus the sorting problem which asks for sorting n numbers reduces to this problem. Similarly, the problem of computing the convex hull of n points in the plane reduces to this problem. In the following, we simply call these problems the *sorting problem* and the *convex hull problem*, respectively.

The optimal trade-offs for the sorting problem and the convex hull problem are known for both models (the random-access model and the multi-pass streaming model.) Under the random-access model, both problems can be solved in $O(n^2/(s \log n) + n \log(s \log n))$ time[3] using $O(s)$ words of workspace [8, 14]. Under the multi-pass streaming model, both problems can be solved in $O(n^2/\log n + n \log s)$ time [7, 13] using $O(s)$ words of workspace and $O(n/s)$ passes of the input array.

With linear-size working space, a triangulation of S can be computed in $O(n \log n)$ time. For the case that a space is given as a positive integer parameter s at most n, several results are known for the random-access model while no result is known for the multi-pass streaming model. Korman et al. presented an s-workspace algorithm for computing a triangulation of S in $O(n^2/s + n \log n \log s)$ time [12]. In the same paper, they presented an s-workspace algorithm for computing the Delaunay triangulation of S in $O((n^2/s) \log s + n \log s \log^* s)$ expected time. Recently, it is improved to $O(n^2 \log n/s)$ deterministic time [4]. Combining [4] and [12], a triangulation of S can be computed in $O(\min\{n^2/s + n \log n \log s, n^2 \log n/s\})$ time.

The problem of computing a triangulation of a simple polygon has also been studied under the random-access model. Aronov et al. [2] presented an s-workspace algorithm for computing a triangulation of a simple n-gon. Their algorithm returns the edges of a triangulation without repetition in $O(n^2/s +$

[3] They state that their running time is $O(n^2/s + n \log s)$ for s bits of workspace, but we measure workspace in words.

$n \log s \log^5 n/s)$ expected time. Moreover, their algorithm can be modified to report the resulting triangles of a triangulation together with their adjacency information. For a monotone n-gon, Barba et al. [5] presented an $(s \log_s n)$-workspace algorithm for triangulating the polygon in $O(n \log_s n)$ time for a parameter $s \in \{1, \ldots, n\}$. Later, Asano and Kirkpatrick [3] showed how to reduce the working space to $O(s)$ words without increasing the running time.

1.2 Our Results.

We present an s-workspace $O(n^2/s + n \log s)$-time algorithm for computing a triangulation of a set of n points in the plane. Our algorithm uses $O(n/s)$ passes over the input array. To our best knowledge, this is the first result on the problem under the multi-pass model. These bounds are asymptotically optimal, which can be shown by a reduction from the sorting problem [13].

Our multi-pass algorithm also improves the previously best known algorithm under the random-access model by Korman et al. which takes $O(\min\{n^2/s + n \log n \log s, n^2 \log n/s\})$ time [4, 12] although the multi-pass model is more restrictive than the random-access model. It seems unclear whether the algorithm by Korman et al. [12] can be extended to a multi-pass streaming algorithm.

Our algorithm has an additional advantage compared to the previously best one. Our algorithm can be extended to report the triangles together with adjacency information as well as the edges of a triangulation without increasing the running time and space. The edge adjacency is essential information in representing and reconstructing the triangulation. In contrast, the algorithms in [4, 12] report the edges of a triangulation in an arbitrary order with no adjacency information of them. Furthermore, the algorithm by Korman et al. [12] uses the algorithm by Asano and Kirkpatrick [3] as a subprocedure, but it is unclear how to modify the subprocedure to report a triangulation together with edge adjacency information [2].

2 Reporting the Edges of a Triangulation

In this section, we present an s-workspace $O(n^2/s + n \log s)$-time algorithm to compute a triangulation of a set S of n points in the plane using $O(n/s)$ passes. Our algorithm is based on the multi-pass streaming algorithm by Chan and Chen [7] for computing the convex hull of a set of points in the plane. For a subset S' of S, we use $\textsc{ch}(S')$ to denote the convex hull of S'.

Chan and Chen presented an algorithm to compute the convex hull of a set S of points in the plane by scanning the points $O(n/s)$ times. They consider $\lceil n/s \rceil$ disjoint vertical slabs each of which contains exactly s points of S, except for the last vertical slab. They use two passes to compute the boundary of $\textsc{ch}(S)$ contained in each vertical slab. For one pass, they find the points of S contained in the vertical slab using Lemma 1. Then they compute the convex hull of them in $O(s \log s)$ time. For the other pass, they find the part of the boundary of the convex hull which appears on the boundary of $\textsc{ch}(S)$. In total, their algorithm takes $O(n^2/s + n \log s)$ time and uses $O(s)$ words of working space.

Lemma 1 ([7]). *Given a point $p \in S$, we can compute the leftmost s points lying to the right of p in $O(n)$ time using $O(s)$ words of working space in a single pass.*

2.1 Our Algorithm

Imagine $\lceil n/s \rceil$ disjoint vertical slabs each of which contains exactly s points of S, except for the last vertical slabs. Let S_i be the set of points of S contained in the ith slab for $i \in \{1, 2, \ldots, \lceil n/s \rceil\}$. By Lemma 1, we can compute all points in S_i by scanning the points in S once using $O(s)$ words of working space if we have the leftmost point of S_i. The pseudocode of the overall algorithm can be found in Algorithm 1.

Algorithm 1 Computing a triangulation of S

```
1: procedure Triangulation(S)
2:     s ← the leftmost point of S
3:     for i ← 1 to ⌈n/s⌉ do
4:         Compute S_i by scanning all points in S once.
5:         Report all edges of a triangulation of S_i.
6:         Let T_i be the set of points lying to the left of any point in S_i.
7:         a ← the rightmost point of T_i and b ← the leftmost point of S_i
8:         LowerTriangulation(a, b, S_i)
9:         UpperTriangulation(a, b, S_i)
```

The overall algorithm (Algorithm 1). We consider all vertical slabs from left to right one by one. After we process a vertical slab, we guarantee that we report all edges of a triangulation of the points of S contained in the union of all previous vertical slabs. First, we compute S_1 explicitly in $O(n)$ time by applying Lemma 1, and compute a triangulation of S_1 in $O(s \log s)$ time. Assume that we just considered S_{i-1} for some $i \in \{2, \ldots, \lceil n/s \rceil\}$ and we computed a triangulation of $S_1 \cup \ldots \cup S_{i-1}$. Let T_i be the set of points lying to the left of any point in S_i, that is, $T_i = S_1 \cup \ldots \cup S_{i-1}$.

Now we handle S_i. We compute S_i explicitly in $O(n)$ time by applying Lemma 1, and compute a triangulation of S_i. Let a be the rightmost point of T_i and b be the leftmost point of S_i. (See Fig. 1(a).)

For two convex polygons C_1 and C_2, we say a line segment c_1c_2 for $c_1 \in C_1$ and $c_2 \in C_2$ a *bridge* of C_1 and C_2 if it appears on the boundary of the convex hull of C_1 and C_2. If a bridge connects the lower chains of C_1 and C_2, we call the bridge the *lower bridge*. Otherwise, we call it the *upper bridge*.

We traverse the boundary of $\text{CH}(T_i)$ from a in clockwise order and traverse the boundary of $\text{CH}(S_i)$ from b in counterclockwise order until we find the lower bridge of $\text{CH}(T_i)$ and $\text{CH}(S_i)$. Let L_i be the polygon whose boundary consists of the chains we visited, ab and the lower bridge. During the traversal, we compute

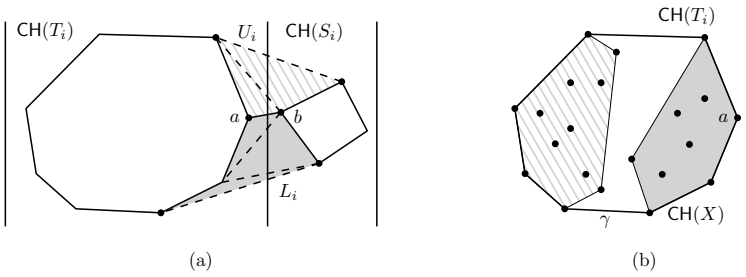

Fig. 1. (a) Starting from ab, we report the edges of a triangulation of L_i (the gray region) and the edges of a triangulation of U_i (the dashed region). (b) The edge γ is the lower bridge of $\text{CH}(X)$ and the convex hull of the points lying to the left of any point of X.

a triangulation of L_i. See Fig. 1(a). The pseudocode of this procedure can be found in Algorithm 2. We call this procedure LowerTriangulation.

Similarly, we find the upper bridge of $\text{CH}(T_i)$ and $\text{CH}(S_i)$ by traversing the boundaries of $\text{CH}(T_i)$ and $\text{CH}(S_i)$. Let U_i be the polygon whose boundary consists of the chains we visited, ab and the upper bridge. We call this procedure Upper-Triangulation. This can be done a way similar to LowerTriangulation. Note that $L_i \cup U_i = \text{CH}(T_{i+1}) \setminus (\text{CH}(T_i) \cup \text{CH}(S_i))$. We show how to compute a triangulation of L_i only because a triangulation of U_i can be computed analogously.

Computing a triangulation of L_i (Algorithm 2). We can construct and traverse the boundary of $\text{CH}(S_i)$ in $O(s \log s)$ time since we can store $\text{CH}(S_i)$ explicitly. However, this does not hold for $\text{CH}(T_i)$ since the size of $\text{CH}(T_i)$ might exceed $O(s)$. To traverse the boundary of $\text{CH}(T_i)$ using $O(s)$ words of working space, we first find the rightmost s points of T_i using one pass by applying Lemma 1. Let X denote the set of such s points. We store X explicitly.

Then we compute $\text{CH}(X)$ and compute the lower bridge of $\text{CH}(X)$ and the convex hull of points of S lying to the left of any point of X. See Fig. 1(b). We can compute the lower bridge in $O(n)$ time by considering all points of S lying to the left of any point of X one by one. Due to this bridge, we can decide which part of the boundary of $\text{CH}(X)$ appears on the boundary of $\text{CH}(T_i)$. We traverse the part of $\text{CH}(X)$ appearing on the boundary of $\text{CH}(T_i)$ until we find the lower bridge of $\text{CH}(T_i)$ and $\text{CH}(S_i)$.

Once we reach the most clockwise vertex of $\text{CH}(X)$ appearing on the boundary of $\text{CH}(T_i)$, we again find the rightmost s points lying to the left of the endpoint of γ not in X, where γ is the lower bridge of $\text{CH}(X)$ and the convex hull of the points lying to the left of any point of X. Then we update X to the set of these points. Note that X may not be S_j for any $1 \leq j \leq i$ in this case. By

construction, the rightmost point of X appears on the lower chain of $\textsc{ch}(T_i)$. We do this until we find the lower bridge of $\textsc{ch}(T_i)$ and $\textsc{ch}(S_i)$.

Algorithm 2 Computing a triangulation of L_i

```
 1: procedure LowerTriangulation(a,b,S_i)
 2:     ⟨x_1, ..., x_t⟩ ← be the lower hull of S_i (from right to left).
 3:     repeat
 4:         X ← the rightmost s points lying to the left of a including a
 5:         ⟨y_1, ..., y_{t'}⟩ ← the part of the lower hull of X appearing on CH(T_i)
            (from left to right)
 6:         repeat
 7:             Report the edge x_t y_{t'}.
 8:             if y_{t'-s} lies to the left of the line containing x_t y_{t'} in direction x_t y_{t'}→ then
 9:                 t' ← t' − 1
10:             else
11:                 t ← t − 1
12:         until all points in X ∪ S_i lie above the line containing x_t y_{t'}
13:         γ ← the bridge of CH(X) and the convex hull of points lying to the left of
            any point of X
14:         Report γ.
15:         a ← the endpoint of γ not in X
16:     until all points in T_i lie above the line containing γ
```

2.2 Analyses

In Algorithm 1, Line 2 can be done in $O(n)$ time using a single pass. Line 4 and 6 can be done in $O(n)$ time using $O(1)$ passes due to Lemma 1. Line 5 can be done in $O(s \log s)$ time since we compute S_i explicitly. Thus, the total running time is $O(n^2/s + n \log s + \sum_i \tau_i)$, where τ_i is the running time of $\textsc{LowerTriangulation}(\cdot, \cdot, S_i)$.

Now consider Algorithm 2. Let t_i be the number of updates of X for S_i. Line 4 takes $O(n)$ time using a single pass due to Lemma 1. Line 5 takes $O(s \log s + n)$ time using a single pass. For Lines 6–12, we compute an edge of a triangulation in each iteration. And each iteration takes $O(1)$ time. Note that Line 12 can be also done in $O(1)$ time since it suffices to consider the boundaries of $\textsc{ch}(S_i)$ and $\textsc{ch}(X)$ locally. Thus Lines 6–12 can be done in $O(n)$ time in total for all S_i's. Lines 13–15 take $O(n)$ time using $O(1)$ passes. Therefore, for a fixed i, the running time of Algorithm 2, except Lines 6–12, is $O(s \log s + t_i n)$. Since Lines 6–12 can be done in $O(n)$ time for all indices i, the total running time of Algorithm 1 is $O(n^2/s + n \log s + \sum_i \tau_i) = O(n^2/s + n \log s + n \sum_i t_i)$.

We claim that the sum of t_i over all i's is $O(n/s)$, which implies that Algorithm 1 takes $O(n^2/s + n \log s)$ time using $O(n/s)$ passes. Assume that X is set to $A_1, A_2, \ldots, A_k$ in order when we handle S_i. No point in A_ℓ appears on the lower chain of $\textsc{ch}(T_i)$ for $\ell = 1, 2, \ldots, k-1$. Thus, no point in A_ℓ appears on the

lower chain of $\mathrm{CH}(T_j)$ for any $j \geq i$. Recall that X is set to a point set whose the rightmost point appears on the lower chain of $\mathrm{CH}(T_j)$ when we handle S_j. Therefore, no point in A_t is contained in X at any time after we handle S_i for $t = 2, 3, \ldots, k-1$. Therefore, the sum of t_i is $O(n/s)$, and the total running time is $O(n^2/s + n \log s)$.

Theorem 1. *Given a set S of n points in the plane, we can report the edges of a triangulation of S in $O(n^2/s + n \log s)$ time using $O(s)$ words of working space and $O(n/s)$ passes.*

3 Reporting the Triangles with Adjacency Information

Let $\mathcal{T}$ be the triangulation of S computed by the algorithm in Section 2.1. In this section, we show how to modify the algorithm to report the triangles of $\mathcal{T}$ together with their adjacency information in addition to the edges of $\mathcal{T}$. That is, we report every pair (τ, τ') of the triangles of $\mathcal{T}$ such that τ and τ' are adjacent to each other in $\mathcal{T}$.

We say a triangle τ of $\mathcal{T}$ is an *inner-slab triangle* if all three corners of τ are in the same vertical slab S_i for some $i = 1, \ldots, \lceil n/s \rceil$. Otherwise, we say τ is a *cross-slab triangle*. Note that if two inner-slab triangle are adjacent to each other in $\mathcal{T}$, they are contained in the same vertical slab. Moreover, for an inner-slab triangle τ and a cross-slab triangle τ', we compute τ' after computing τ. In this case, we report the adjacency between τ and τ' when we compute and report τ'.

Reporting an inner-slab triangle. Consider an inner-slab triangle τ with corners in S_i for some i. Recall that we compute all points in S_i explicitly, and compute a triangulation of them. When we compute a triangulation of them, we also report the triangles of it with their adjacency information. For a cross-slab triangle τ' of $\mathcal{T}$ adjacent to τ, we will report their adjacency information when we compute and report τ'.

Reporting a cross-slab triangle. Consider a cross-slab triangle τ' whose rightmost corner lies on S_i for some i. This triangle comes from a triangulation of $L_i \cup U_i$. We compute τ' while we traverse the boundaries of $\mathrm{CH}(T_i)$ and $\mathrm{CH}(S_i)$. A cross-slab triangle adjacent to τ' is also computed during this traversal, thus the adjacency information between them can be computed during the traversal.

Each corner of τ' lies on the boundary of $\mathrm{CH}(S_i)$ or the boundary of $\mathrm{CH}(T_i)$. If two corners lie on the boundary of $\mathrm{CH}(S_i)$, there is an inner-slab triangle adjacent to τ' contained in S_i. The adjacency information between them can be computed without increasing the running time because we compute the convex hull of S_i explicitly.

Now assume that two corners a and b lie on the boundary of $\mathrm{CH}(T_i)$. Let τ'' be the triangle of $\mathcal{T}$ incident to ab other than τ'. To report the adjacency information between τ' and τ'', we compute τ'' together with ab when we traverse

the boundary of $\mathrm{CH}(T_i)$. To do this, we specify a way to triangulate $L_i \cup U_i$ as described in Algorithm 2.

For L_i, we initially set a' to the rightmost point of X and b' to the leftmost point of S_i for each set X. Then we move a' along the part of the boundary of $\mathrm{CH}(X)$ appearing on $\mathrm{CH}(T_i)$ in clockwise direction as much as possible until $a'b'$ intersects the boundaries of $\mathrm{CH}(X)$ and $\mathrm{CH}(S_i)$. Then we move b' one step along the boundary of $\mathrm{CH}(S_i)$ in counterclockwise direction, and move a' again. We do this until we all points in $X \cup S_i$ lie above the line containing $a'b'$. For U_i, we can compute a triangulation similarly.

Then we have the following lemma.

Lemma 2. *Given the convex hull of the set of points in S_j for some $j = 1, \ldots, \lceil n/s \rceil$, we can find the lowest triangle of $\mathcal{T}$ contained in L_i in $O(n)$ time using $O(s)$ words of workspace and $O(1)$ passes.*

Proof. The lowest triangle of $\mathcal{T}$ contained in L_i is incident to the lower bridge of $\mathrm{CH}(T_j)$ and $\mathrm{CH}(S_j)$. By scanning the points of S once, we compute the lower bridge of $\mathrm{CH}(T_j)$ and $\mathrm{CH}(S_j)$. Let a and b be the endpoints of the lower bridge such that $a \in \mathrm{CH}(T_j)$ and $b \in \mathrm{CH}(S_j)$. Then, by scanning the points in S once again, we compute the counterclockwise neighbor a' of a along the boundary of $\mathrm{CH}(T_i)$. Since we maintain the set S_i explicitly, we can compute the clockwise neighbor b' of b along the boundary of $\mathrm{CH}(S_i)$ in constant time without scanning the points of S.

By construction, the triangle with corners a, b, b' is the lowest triangle of $\mathcal{T}$ contained in L_i if b and b' lie below the line passing through a and a'. Otherwise, the triangle with corners a, a', b is the lowest triangle of $\mathcal{T}$ contained in L_i. In any case, we can report the lowest triangle of $\mathcal{T}$ in L_i using $O(s)$ words of workspace and $O(1)$ passes. ◻

We modify Algorithm 2 as follows. For Line 4, we set X to the set of points in S_j lying to the left of a for $a \in S_j$ using Lemma 3. Then we can obtain the triangles of $\mathcal{T}$ incident to the part of the lower hull of X appearing on $\mathrm{CH}(T_i)$ by applying the algorithm for triangulating S_j we use in Line 5 of Algorithm 1. To compute the triangle of $\mathcal{T}$ incident to γ and contained in $\mathrm{CH}(T_i)$, we apply Lemma 2.

Lemma 3. *Given any point $p \in S_j$ for some $j = 1, \ldots, \lceil n/s \rceil$, we can compute S_i in $O(n)$ time using $O(s)$ words of workspace and $O(1)$ passes.*

Proof. For the first pass, we compute the number of points in S lying to the left of p. This determines the value of j with $p \in S_j$. Then we find the rightmost s points lying to the left of p in $O(n)$ time using a single pass by applying Lemma 1. One of them is the leftmost point of S_j, and we can find it in $O(s)$ time. We can compute S_i using a single pass by applying Lemma 1 again. ◻

The following theorem summarizes this section.

Theorem 2. *Given a set S of n points in the plane, we can report the triangles of a triangulation of S with their adjacency information in $O(n^2/s + n \log s)$ time using $O(s)$ words of working space and $O(n/s)$ passes.*

4 Conclusion

In this paper, we present an s-workspace $O(n^2/s + s \log n)$-time algorithm for computing a triangulation of a set of n points in the plane under the multi-pass model. Our algorithm uses $O(n/s)$ passes over the input array. It is not only the first algorithm for this problem under the multi-pass model, but it also improves the previously best known random-access algorithm [12]. Moreover, its running time is optimal under the multi-pass model.

One interesting open problem remaining from our work is whether our algorithm can be improved under the random-access model. Under this model, the best known lower bound is $\Omega(n^2/(s \log n) + n \log(s \log n))$ for $s \leq n/\log n$.

References

1. P. K. Agarwal, S. Krishnan, N. H. Mustafa, and S. Venkatasubramanian. Streaming geometric optimization using graphics hardware. In *11th Annual European Symposium on Algorithms (ESA 2003)*, pages 544–555, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.
2. B. Aronov, M. Korman, S. Pratt, A. van Ressen, and M. Roeloffzen. Time-space trade-offs for triangulating a simple polygon. In *15th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2016)*, volume 53, pages 30:1–30:12, 2016.
3. T. Asano and D. Kirkpatrick. Time-space tradeoffs for all-nearest-larger-neighbors problems. In *13th Algorithms and Data Strucutres Symposium (WADS 2013)*, pages 61–72, 2013.
4. B. Banyassady, M. Korman, W. Mulzer, A. van Renssen, M. Roeloffzen, P. Seiferth, and Y. Stein. Improved Time-Space Trade-Offs for Computing Voronoi Diagrams. In *34th Symposium on Theoretical Aspects of Computer Science (STACS 2017)*, volume 66, pages 9:1–9:14, 2017.
5. L. Barba, M. Korman, S. Langerman, K. Sadakane, and R. I. Silveira. Space-time trade-offs for stack-based algorithms. *Algorithmica*, 72(4):1097–1129, 2015.
6. A. Borodin and S. Cook. A time-space tradeoff for sorting on a general sequential model of computation. *SIAM Journal on Computing*, 11(2):287–297, 1982.
7. T. M. Chan and E. Y. Chen. Multi-pass geometric algorithms. *Discrete & Computational Geometry*, 37(1):79–102, 2007.
8. O. Darwish and A. Elmasry. Optimal time-space tradeoff for the 2D convex-hull problem. In *22nd Annual European Symposium on Algorithms (ESA 2014)*, pages 284–295, 2014.
9. J. Feigenbaum, S. Kannan, A. McGregor, S. Suri, and J. Zhang. On graph problems in a semi-streaming model. In *31st International Colloquium on Automata, Languages and Programming (ICALP 2004)*, pages 531–543, 2004.
10. J. Feigenbaum, S. Kannan, A. McGregor, S. Suri, and J. Zhang. Graph distances in the data-stream model. *SIAM Journal on Computing*, 38(5):1709–1727, 2009.

11. G. N. Frederickson. Upper bounds for time-space trade-offs in sorting and selection. *Journal of Computer and System*, 34(1):19–26, 1987.
12. M. Korman, W. Mulzer, A. van Renssen, M. Roeloffzen, P. Seiferth, and Y. Stein. Time-space trade-offs for triangulations and Voronoi diagrams. In *Algorithms and Data Structures (WADS 2015)*, pages 482–494, 2015.
13. J. Munro and M. Paterson. Selection and sorting with limited storage. *Theoretical Computer Science*, 12(3):315 – 323, 1980.
14. J. Pagter and T. Rauhe. Optimal time-space trade-offs for sorting. In *39th Annual Symposium on Foundations of Computer Science (FOCS 1998)*, pages 264–268, 1998.
15. S. Suri, C. D. Toth, and Y. Zhou. Range counting over multidimensional data streams. *Discrete & Computational Geometry*, 36(4):633–655, 2006.