# A Streaming Algorithm for the Convex Hull

Raimi Rufai* Dana Richards†


## Abstract

Given a continous stream of data points $S$ and a memory budget for $O(k)$ points. We present an algorithm that maintains an approximate convex hull at a cost of $O(\log k)$ time per input point. We discuss changes to the algorithm to support the sliding window model.


## 1 Introduction

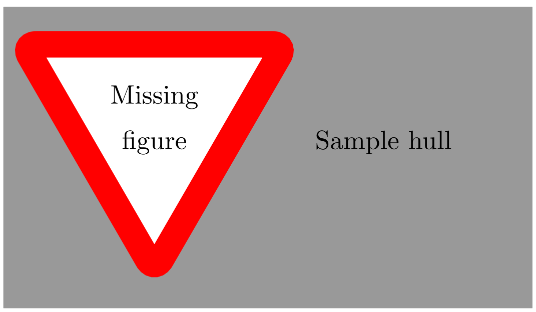


Figure 1: Convex Hull

A streaming algorithm essentially has three parts: an initialization part, a processing part and a query answering part.

**Initilization** In this part, counters and data structures are initialized.

**Process** This part computes an intermediate structure that can be easily updated with a new input as well as easily queried to obtain an answer based on the inputs seen so far.

**Query** This part responds to queries using the latest state of the intermediate structure built in the process step above.

A streaming algorithm is typically limited in the amount of resources it is allowed.

In this paper, we present a simple streaming algorithm for the convex hull problem. Section 2 relates relevant literature. Section 3.1.1 presents our proposed algorithm. Section 4 concludes the paper and speculates on future work.


*Department of Computer Science, George Mason University, `rrufai@gmu.edu`

†Department of Computer Science, George Mason University, `richards@gmu.edu`


## 2 Related Work

We review the the *priority search trees* of McCreight [1].

describe priority search trees – may not be needed anymore with the idea of using parallel structures with pointers back and forth

## 3 Contributions

### 3.1 Streaming Algorithm

#### 3.1.1 Description of Algorithm

L = {heap, rbt, centroid}. heap uses the dogears as its key. rbt uses the polar angle as its key.

Given an input stream of points and a memory limit of $O(k)$

**Input:** The first $k$ input points $S_k$ and parameter $k$
**Output:** The sequence of points $L$ forming the convex hull of $S_k$ and their centroid $c$
INITIALIZE$(S_k, k)$
1 $L \leftarrow \text{conv}(S_k)$
2 $L.c \leftarrow \text{CENTROID}(L)$
3 **for each** $p$ **in** $L$
4 **do**
5 $\quad p.\Theta \leftarrow \text{POLAR}(p, L.c)$
6 $\quad p.dogear \leftarrow \text{AREA}(L.pred(p), p, L.succ(p))$
7 $\quad L.rbt.insert(p)$
8 $\quad L.heap.insert(p)$
9 **return** $L$

Figure 2: Algorithm INITIALIZE

The call UPDATEHULL$(L, p)$ in line 3.1.1 takes an existing approximate hull, with no more than $k$ vertices and updates it with a new point $p$. If $p$ falls within the interior of conv$(L)$ or on its boundary, it is discarded.

**Input:** Current set of hull vertices $L$, the next streaming input point $p$ and a parameter $k$
**Output:** A subset of $S$
PROCESS$(L, p, k)$
$p.\Theta \leftarrow$ POLAR$(L.c, p)$
$L =$ UPDATEHULL$(L, p)$
**if** $|L| > k$
  **then** $L \leftarrow$ SHRINKHULL$(L)$
**return** $L$

Figure 3: Algorithm PROCESS

**Input:** The hull structure $L$, the new point $q$.
**Output:** The hull structure $L$ updated with point q.
UPDATEHULL$(L, q)$
$(p, r) \leftarrow L.rbt.FIND(q)$
**if** $q \notin \Delta(p, q, r)$
  **then** $(p', r') \leftarrow$ FINDTANGENTIALNEIGHBORS$(L, q)$
    $L.rbt.delete(q)$
    $L.heap.delete(q)$
    $L.heap.refresh(p')$
    $L.heap.refresh(r')$
**return** $L$

Figure 4: Algorithm UPDATEHULL

**Input:** The hull structure $L$, the hull size limit $k$.
**Output:** The hull structure $L$ that respects the $k$ limit.
SHRINKHULL$(L, k)$
$q \leftarrow L.heap.findMax()$
$(p, r) \leftarrow (L.rbt.pred(q), L.rbt.succ(q))$
$L.rbt.discardChain(p', r')$
$L.heap.discarChain(p', r')$
$L.heap.refresh(p)$
$L.heap.refresh(r)$
**return** $L$

Figure 5: Algorithm UPDATEHULL

This test can be done in $O(k)$ time using one of these simple algorthms:

The procedure REFRESH does three things: recomputes the dogear of its argument, removes it from and inserts it back into the heap.

the procedure DISCARDCHAIN removes a whole chain of contiguous points between (but not including) its arguments from the RBT or the Heap.

**Edge Intersection Tests** Let $c$ be an interior point of conv$(L)$. Let $e$ be the segment connecting the points $c$ and $p$. Test for intersection of $e$ with each each edge of conv$(L)$. If it intersects none of them, then it is an interior point. If the intersection point is the point $p$ itself, then $p$ lies on the boundary of conv$(L)$. Otherwise, $p$ is an exterior point.

**Triangle Inclusion Tests** Let $c$ be defined as above. Test for $p$'s inclusion in each triangle formed by $c$ and an edge of conv$(L)$. If $p$ lies within one of them then it is an interior point. Otherwise, it is an exterior point.

**Binary search**

incorporate ideas from [?]

**Input:**
**Output:** A subset of $S$
QUERY()
**return** $L.rbt$

Figure 6: Algorithm QUERY

#### 3.1.2 Complexity Analysis

#### 3.1.3 Error Analysis

#### 3.1.4 Emplirical Results

#### 3.1.5 Open Issues

### 3.2 Sliding Window Algorithm

#### 3.2.1 Description of Algorithm

#### 3.2.2 Complexity Analysis

#### 3.2.3 Error Analysis

#### 3.2.4 Emplirical Results

#### 3.2.5 Open Issues

**Underlying Convex Hull Definition.**

**Hull Approximation Type.**

**Analogous Sorting Algorithm.**

**Generalization to $d$-space.**

**Complexity.**

**Accuracy Measures.**

**Input Space.**

**Parallelizability.**

**Streaming model.**

**Miscellaneous Issues.**

## 4 Conclusion

to summarize results and discuss open problems

## References

[1] E. McCreight. Priority search trees. *SIAM Journal on Computing*, 14(2):257–276, 1985.