Computing Convex Layers of a Dynamic Point Set

Sanjib Sadhu and Niraj Kumar

***Abstract*—The convex layers of a given point set can be computed by iterative process of finding convex hull after discarding the points of already computed convex hull. Computation of convex layers has been widely studied in the static environment where the point set are fixed. In this paper, we propose an idea to compute set of convex layers in dynamic context. There exists an optimal time algorithm to solve the static version of the problem in O(n log$n$) time. However, to solve dynamic version of the problem the suggested algorithm requires O(n^2) time for a set of n points.**

***Index Terms*—Computational geometry, convex hull, convex layers, incremental algorithm, tangent.**

I. Introduction

The set of convex layers (or simply convex layers) can be computed by an iterative process of computing convex hull, after discarding points of already computed convex hull. It is one of the widely studied computational geometry problem, because of its wide area of applications. The convex layers can be applied to design a robust estimator [1]. Tukey [2] suggested a procedure called "peeling", for eliminating outliers. Another application of convex layers has been suggested by Lee et al. in the half plane range query problem [3]. Recently, Liew [4] suggested application of convex layers to TSP problem. Much effort has been applied to compute the convex layers of fixed point set. There exist an algorithm [5] to compute the convex layers for fixed point set in optimal time, i.e., in $O(n \log n)$ time. However, there exist instances, where point set is not fixed and points can be inserted or deleted at any time. In this paper, we suggest an algorithm to compute the convex layers in dynamic context. The convex layers can be computed in $O(n^2)$ time by applying the proposed approach.

To compute a convex layers from static point set, a brute force approach can be applied, which involves applying an optimal convex hull algorithm, say [6], repeatedly for each layer, thus it requires $O(n^2 \log n)$ time. Silverman et al. reduced time complexity to $O(n^2)$ [7]. Chazelle [5] proposed algorithm to compute the convex layers in optimal time. In [5] the deletion process batched to give $O(n \log n)$ time algorithm.

This paper is organized as follows. Some preliminaries are presented in Section II. Detailed description of proposed algorithm with its analysis is presented in Section III. Finally, in Section IV, we conclude with possible future works.

Manuscript received July 13, 2014; revised October 10, 2014.

Sanjib Sadhu is with the Department of Computer Science, National Institute of Technology, Durgapur, India 713209 (e-mail: sanjibsadhu411@gmail.com).

Niraj Kumar was with the National Institute of Technology, Durgapur, India 713209. He is now with the Department of Computer Science, Dronacharya College of engineering, Gurgaon, Haryana (e-mail: nirajcse08@gmail.com).

II. Preliminaries

A subset S of the plane is called **convex** [8] if and only if for any pair of points $(p, q) \epsilon S$ the line segment $\ell s(p, q)$ is completely contained in S. The **Convex Hull** $CH(S)$ of a point set S is the smallest convex set that contains S and is represented by set of vertices that defines hull edges.

For a given 2-dimensional planar point set $S = \{s_1, s_2, \ldots, s_n\}$, its C**onvex Layers**, denoted by CL, can be visualized as set of convex hulls computed by iterative procedure of computing convex hull and discarding the points on already computed convex hull (Refer to Fig. 1). In this paper, to represent a convex hull $CL_i \epsilon CL$, we use counterclockwise (CCW) sequence of vertices on convex hull CL_i. For a point $p \epsilon CL_i$ (Refer to Fig. 1), point $p.prev$ and $p.next$ represents point previous to p and the point next from p, respectively (in CCW order).

III. Algorithm

The point set is dynamic, so, at any time some point may be inserted or deleted from point set. Computing convex layers in dynamic context requires insertion as well as deletion of points to CL. In this section we give the detailed description of two procedures viz.: Insert() and Delete() to accomplish insertion and deletion of points, respectively.

Assumption: All the points are in general position, i.e. no three points are collinear.

Assumption: The layers $CL_i \epsilon CL$, are numbered outward to inward. Outermost layer is CL_1 and innermost is CL_k, where k be the number of layers in CL.

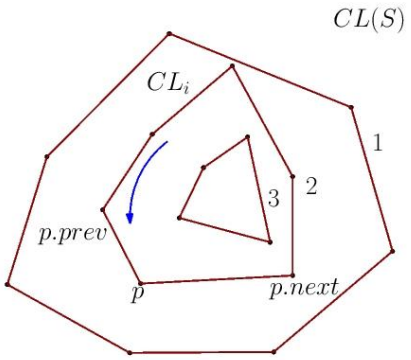

Fig. 1. Convex layer set CL.

Definition: The **region** (here unbounded) is defined as the minor sector formed by two rays. In the Fig. 2, region $R(p_1, p_2, p_3)$ is the minor sector formed by rays r_1 (passing through point p_1) and r_2 (passing through point p_3) such that both rays have common fixed point p_2.

DOI: 10.7763/IJCTE.2015.V7.1008

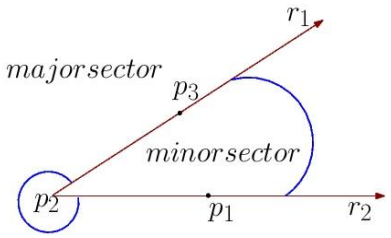

Fig. 2. Region.

The line passing through point p and q is represented by $\ell(p,q)$. However, the line segment with the end points p and q is represented by $\ell s(p,q)$.

Points p and q are said to be visible if $\ell s(p,q)$ does not intersects any other line segment or does not passes through any other point. Hence, in Fig. 3, p_1 and q are visible, whereas p and q are not visible.

We say, a point q is lying outside layer CL_i, if it is lying outside CL_i but not outside CL_{i-1}.

A. Algorithm to Insert a Point q to CL

Algorithm 3 inserts an incoming point to CL. To insert an incoming point q to convex layer set CL, first it is required to determine the layer CL_i to which q will be inserted, the Algorithm 1 serves this purpose. Insertion of a point to CL_i requires removal of chain lying under the wedge formed by two tangents from the point q, i.e., the part of CL_i visible from q. Consider Fig. 3, insertion of point q to CL_i requires removal of the visible part of CL_i , i.e., $m.prev$ to p_1. A point is said to be lower (resp. upper) tangent point, denoted by v_l (resp v_u), if it is first (resp. last) point of the chain in CCW order. Algorithm 2 computes the two tangent points.

1) Algorithm 1: Find layer

To determine the layer $CL_i \in CL$, to which point q is to be inserted, a search is performed outward (from CL_1) to inward (to CL_k). While inspecting a layer CL_i, a point $p \in CL_i$ is chosen randomly. If point q is not lying in region, $R(p.prev, p, p.next)$, then the point is certainly outside the layer CL_i. For instance, in Fig. 3 for point p_1 the point q lies outside the region $R(p.prev, p, p.next)$. Hence, point q lies outside layer CL_i (It is important to note that layer CL_1 to CL_{i-1} has already been examined and search has failed.) Otherwise, point may be lying inside or outside the layer. A binary search is performed to contract the region and every time either $p.prev$ or $p.next$ is updated until both become adjacent depending on whether q is on the left or right of $\ell(p, p_m)$, where p_m is midpoint of chain $p.next$ to $p.prev$. If the line segment $\ell s(q,p)$ intersects $\ell s(p.next, p.prev)$ then point q is outside the layer CL_i (layer being examined), otherwise next layer checked to find out if q is lying outside.

Algorithm 1: Find Layer(q, CL)

Input: Point q and convex layers CL, say k is the number of layers in CL.

Output: The convex layer CL_i such that q lies outside CL_i.

1) $i \leftarrow 1$
2) while($i \neq k+1$) do
3) if ($i = k$ and CL_k contains less than three points)
4) Return k
5) else
6) select a point $p \in CL_i$
7) if (q is not in region $R(p.prev, p, p.next)$)
8) Return i
9) else
10) repeat
11) $m := (p.next + p.prev)/2$
12) if (q is to left of $\ell(p, p_m)$)
13) $p.next := m$
14) else
15) $p.prev := m$
16) until ($p.next$ & $p.prev$ are not adjacent)
17) if ($\ell s(q,p)$ intersects $\ell s(p.next, p.prev)$)
18) return i
19) else
20) $i++$
21) Return $k+1$ //q is inside CL_k

Let n be the total number of points in the point set S and n_i be the number of points in the layer CL_i. Let k be the number of layers in convex layer set CL. Algorithm 1 determines the layer to which incoming point should be inserted. The layers are considered in outward to inward fashion. Inspection of each layer, CL_i, takes $log(n_i)$ time. So, in the worst case all layers inspected, hence, time to compute layer is

$$log(n_1) + log(n_2) + \ldots + log(n_k) \leq n_1 + n_2 + \ldots + n_k = n$$

Hence, it is clear that correct layer to which incoming point should be inserted can be found in $O(n)$ time.

2) Algorithm 2: Compute tangent

A line segment from point q to a point $v \in CL_i$ is said to be a tangent from q to CL_i, if both the points $v.prev$ and $v.next$ lie on same side of $\ell(q,v)$. If $v.prev$ and $v.next$ lies on the right (resp. left) of $\ell(q,v)$ then v is said to be lower (resp. upper) tangent point, denoted by v_l (resp. v_u).

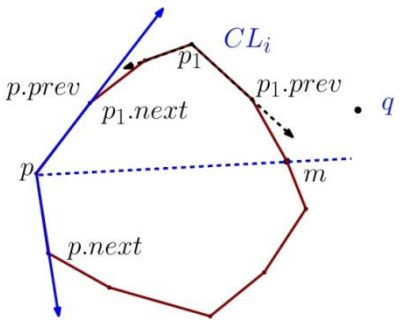

Fig. 3. Find layer.

Let q be a point outside layer CL_i. Then, two points v and $v_1 (\in CL_i)$ are said to be of similar nature if i). $\ell s(q,v)$ intersects $\ell s(v.prev, v.next)$ and $\ell s(q, v_1)$ intersects $\ell s(v_1.prev, v_1.next)$, or ii). Neither $\ell s(q,v)$ intersects $\ell s(v.prev, v.next)$ nor $\ell s(q, v_1)$ intersects $\ell s(v_1.prev, v_1.next)$.

A point, say v_2 ($\in CL_i$), is chosen randomly and if it is not a tangent point then select another point, say v_3, at $n/2$ distance from v_2. If both the points are of same nature, then chain v_2 to v_3 does not contain any tangent point, whereas chain v_3 to v_2 contains both the tangent points. Otherwise, both the chains v_2 to v_3 and v_3 to v_2 contains one tangent point (Refer Fig. 4).

In Algorithm 2, whether the randomly selected point is

tangent or not can be checked in constant time. In the contraction step 9-13, binary search like procedure followed, every time length of chain reduced to $n/2$. As the number of points (n_i) in layer CL_i is $O(n)$. Thus, a tangent point can be computed in $O(logn)$ time. Hence, it is clear that two tangent points can be computed in $O(logn)$ time.

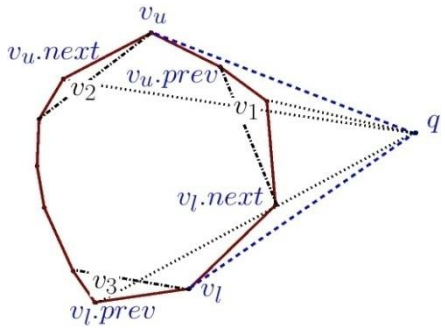

Fig. 4. Computing tangent.

3) Algorithm 3: Insert point

To insert a point, say q, to convex layer CL_i, the chain v_l to v_u must be added to next inner layer, which in turn may require inner layer(s) to be reconfigured (Refer to Fig. 5).

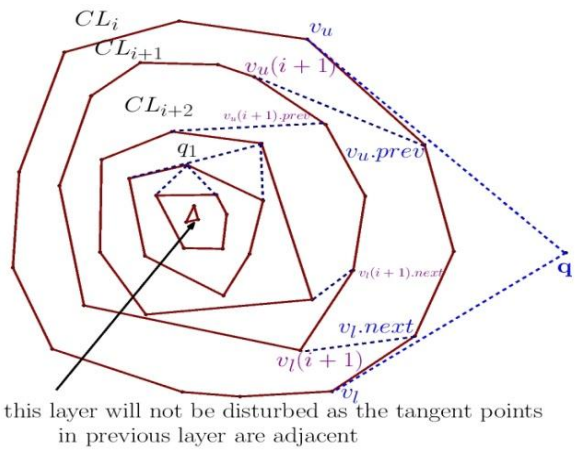

Fig. 5. Inserting a point to CL.

The Algorithm 2 can be easily modified to compute either upper tangent only (i.e. procedure Compute Upper Tangent() in Algortihm 3) or to compute lower tangent only (i.e. procedure Compute Lower Tangent() in Algortihm 3). A convex hull edge is represented by $e(u, v)$, where u and v are two consecutive vertices on the convex hull.

Algorithm 2: Compute Tangent(q, CL_i)

Input: point q and convex layer CL_i

Output: upper (v_u) and lower tangent point (v_l) from q to CL_i

1) n:= no. of points in CL_i, S_i:= points of CL_i
2) while (two tangent points not computed) do
3) select a point $v \in S_i$
4) if($v.prev$ and $v.next$ lies on the right of $\ell(q, v)$)
5) return v as lower tangent point (v_l)
6) else if($v.prev$ and $v.$next lies on the left of $\ell(q, v)$)
7) return v as upper tangent point (v_u)
8) else
9) select a point v_1 at $n/2$ distance from v
10) if points v and v_1 are of similar nature
11) discard chain v to v_1
12) S_i:= points of chain v_1 to v
13) go to step 3
14) else
15) S_1:= points of chain v to v_1
16) S_2:= points of chain v_1 to v
17) go to step 3 with S_1
18) go to step 3 with S_2

Algorithm 3: Insert Point(q, CL)

Insert a point in the convex layer set while maintaining the convexity property. It assumes that at least three points are already arrived.

Input: q is the point to be inserted to the convex layers CL with k layers.

Ouptut: Convex layers set CL after insertion of point q.

1) $i = FindLayer(q, CL)$
2) if $i = k + 1$ then do
3) $CL_{k+1} = q$ //new layer added
4) return CL with $k + 1$ layers
5) if $i = k$ and CL_k contains less than three points
6) $CL_k = CL_k \cup q$
7) return CL
8) $ComputeTangent(q, CL_i)$
9) if v_l & v_u are adjacent
10) connect q with v_u and v_l
11) remove edge $e(v_l, v_u)$
12) return //CL_{i+1} not disturbed
13) remove chain v_l . next to v_u. prev from CL_i
14) remove edges e(v_l, v_l. next) and e(v_u. prev, v_u)
15) connect q with v_u and v_l to complete CL_i
16) $i = i + 1$
17) while ($i \neq k$) do
18) $v_u(i + 1) =$ Compute Upper Tangent(CL_{i+1}, v_u. prev)
19) $v_l(i + 1) =$ Compute Lower Tangent (CL_{i+1}, v_l. next)
20) if $v_l(i + 1)$ & $v_u(i + 1)$ are adjacent
21) connect v_u. prev with $v_u(i + 1)$ and v_l. next
22) with $v_l(i + 1)$
23) remove edge $e(v_l(i + 1), v_u(i + 1))$
24) return //next inner layer not disturbed
25) remove chain $v_l(i + 1)$. next to $v_u(i + 1)$. prev from CL_{i+1}
26) remove edges $e(v_l(i + 1), v_l(i + 1).$ next)
27) & $e(v_u(i + 1).$ prev , $v_u(i + 1))$
28) connect v_u. prev with $v_u(i + 1)$ & v_l. next with
29) $v_l(i + 1)$
30) v_u. prev := $v_u(i + 1)$. prev
31) v_l. next := $v_l(i + 1)$. next
32) $i := i + 1$
33) CL_i is chain v_u. prev to v_l. next

To insert an incoming point q the layer, say CL_i, to which it should be inserted can be computed in $O(n)$ time and two tangent points to the layer CL_i can be found in $O(logn_i)$ time. With this information, point q can be inserted to layer CL_i in constant time. Consequently, inner layer(s) may require to be modified. In the worst case, the point inserted to outer layer and all inner layers required to be modified, in that case running time is:

$$\left.\begin{matrix} O(n) + O(log\ n_1) \\ + O(log\ n_2) + \cdots + O(log\ n_k) \end{matrix}\right\} \begin{matrix} \text{updating outermost layer} \\ \text{updating all inner layers} \end{matrix}$$
$$\leq O(n) + O(log\ n_1) + (n_2 + n_3 + \cdots + n_k)$$
$$\leq O(n)$$

Hence, a single point can be inserted in $O(n)$ time.

B. Algortithm to Delete a Point from CL

Deletion of a point q from convex layer CL_i may result into reconfiguration of inner layer(s), whereas outer layers remain unaltered. Algorithm 4 deletes a point q from CL and reconfigures CL, so that convexity is maintained. Deletion involves finding tangents and updating inner layer(s) if required (Refer Fig. 6).

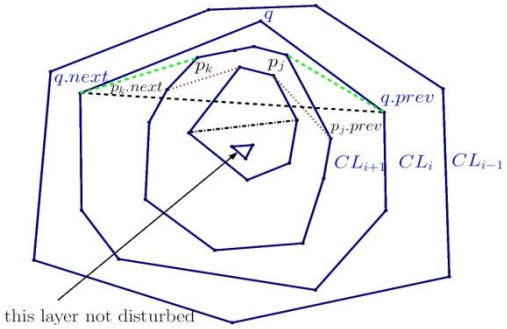

Fig. 6. Deletion.

Deletion of a point (or chain) from layer CL_i, may require inclusion of a part of layer CL_{i+1} to layer CL_i (Refer Fig. 6).

Candidate list $CList$ stores points of CL_{i+1} that are to be added to CL_i.

In step 7 of Algorithm 4, to maintain CCW order, points added to $CList$ after $q.prev$ and before $q.next$.

A point (or chain) can be added to or removed from layer CL_i in constant time. The expensive step is computation of tangents. By applying Algorithm 2, a tangent can be computed in $O(\log n)$ time. In the worst case, after deletion of a point from outermost layer, all the inner layers are required to be modified. Thus, tangent computed for all inner layers,

$$log(n_2) + \ldots + log(n_k)$$
$$\leq n_1 + n_2 + \ldots + n_k = n$$

Thus, the time required to delete a point is $O(n)$.

Algorithm 4: Delete(q, CL_i)

Input: q is the point to be deleted from CL_i and k is number of layers in CL.

Output: Convex layers set CL after deletion of point q.

1) Candidate list $CList$ initialized as an empty list.
2) $i := 1$
3) repeat
4) Compute tangents from $q.prev$ and $q.next$ to CL_{i+1} lying to the right of the line $l(q.prev, q.next)$, say p_a and p_b be tangent points, respectively. If no such points found, return.
5) The points lying on the chain p_a to p_b are added to $CList$ in order.
6) if $CList$ is empty //CL_{i+1} not disturbed
   Remove q from CL_{i+1}
   Return
7) else

$$CL_{i+1} := CL_{i+1} - CList$$
$$CL_i := CL_i - \{q\}$$
$$CL_i := CL_i \cup CList$$
$$q.next := CList[last].next$$
$$q.prev := CList[first].prev$$
$$empty\ CList$$
$$i = i + 1$$

8) while ($i \neq k$)

Hence, it is clear that a single point can be inserted or deleted in $O(n)$ time. Algorithm 3 and Algorithm 4 can be used to compute the convex layers in the dynamic context in $O(n^2)$. However, the amortized time complexity will be much lower.

IV. Conclusion

In this paper we have presented an idea to compute convex layers in dynamic environment. The primary requirement is to allow insertion and deletion at any instant. Time complexity of proposed algorithm is $O(n^2)$. To reduce the running time for insertion and deletion will be our future work.

References

[1] M. I. Shamos, "Computational geometry," Ph.D. thesis. Dept. of Computer Science, Yale University, 1978.
[2] P. J. Huber., "Robust statistics: a review," *Annals of Mathematical Statistics*, vol. 43, no. 3, pp. 1041-1067, 1972.
[3] B. Chazelle, L. J. Guibas, and D. T. Lee, "The power of geometric duality," in *Proc. 24th IEEE Annual Symposium Foundations of Computer Science*, 1983, pp. 217-225.
[4] S. Liew, "Introducing convex layers to the Traveling Salesman Problem," *The Computing Research Repository*, 2012.
[5] B. Chazelle, "On the convex layers of a planar set," *IEEE Transactions on Information Theory*, vol. IT-31, pp. 509-517, 1985.
[6] R. L. Graham, "An efficient algorithm for determining the convex hull of a finite planar set," *Information Processing Letter*, vol. 1, pp. 132-133, 1972.
[7] P. J. Green and B. W. Silverman, "Constructing the convex hull of a set of points in the plane," *Computer Journal*, vol. 22, pp. 262-266, 1979.
[8] M. D. Berg, O. Cheong, M. V. Kreveld, and M. Overmars, *Computational Geometry: Algorithms and Applications*, 3rd ed., Springer-Verlag TELOS Santa Clara, CA, USA, 2008.

Sanjib Sadhu has done his B.E. degree in computer science and engineering from NIT Durgapur and M.Tech. degree in computer science and engineering from N.I.T. Durgapur. Currently he is working with the Department of Computer Science and Engineering at NIT Durgapur as an assistant professor. His research areas include algorithm design and computational geometry.

Niraj Kumar got the B.E. degree in computer science and engineering from MDU Rohtak and M.Tech. degree in computer science and engineering from N.I.T. Durgapur. Currently he is working with the Department of Computer Science and Engineering at Dronacharya College of Engineering Gurgaon as an assistant professor. His research areas include algorithm design and computational geometry.