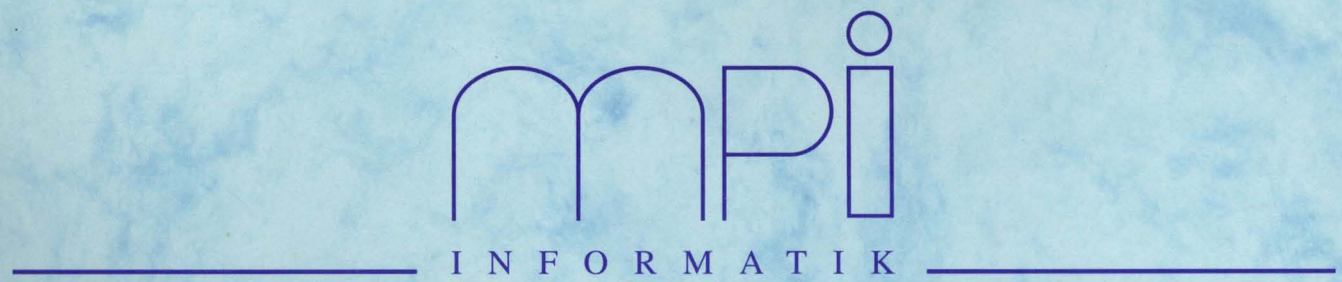

Dynamic Maintenance of 2-d Convex Hulls and Order Decomposable Problems

Sanjiv Kapoor

MPI–I–95–1-015 June 1995

FORSCHUNGSBERICHT ■ RESEARCH REPORT

MAX-PLANCK-INSTITUT
FÜR
INFORMATIK

Im Stadtwald ■ 66123 Saarbrücken ■ Germany

The *Max-Planck-Institut für Informatik* in Saarbrücken is an institute of the *Max-Planck-Gesellschaft*, Germany.

ISSN: 0946 - 011X

Forschungsberichte des

Max-Planck-Instituts für Informatik

Further copies of this report are available from:

Max-Planck-Institut für Informatik
Bibliothek & Dokumentation
Im Stadtwald
66123 Saarbrücken
Germany

Dynamic Maintenance of 2-d Convex Hulls and Order Decomposable Problems

Sanjiv Kapoor

MPI–I–95–1-015 June 1995

Dynamic Maintenance of 2-d Convex Hulls and Order Decomposable Problems

Sanjiv Kapoor
Department of Computer Science
Indian Institute of Technology
Hauz Khas, New Delhi 110016

June 26, 1995

Abstract

In this paper we consider dynamic data structures for Order Decomposable Problems. This class of Problems include the Convex Hull Problem, the Voronoi Diagram Problem, the Maxima Problem and the Intersection of Half Spaces. This paper first describes a scheme for maintaining convex hulls in the plane dynamically in $O(logn)$ amortized time for insertions and $O(log^2 n)$ time for deletions. $O(n)$ space is used. The scheme improves on the time complexity of the original scheme by Overmars and Van Leeuven. We then consider the general class of Order Decomposable Problems. We show improved behavior for insertions in the presence of deletions, under some assumptions. The main assumption we make is that the problems are required to be *change sensitive* , i.e. updates to the solution of the problem at an insertion can be obtained in time proportional to the changes.

1 Introduction

In this paper we consider dynamic data structures for Order-decomposable Problems. This class of problem include the Convex Hull problem, the Maxima problem, the Voronoi Diagram problem, and the Intersection of Half-Spaces etc. [Mb]. Under some assumptions, the main being that of *change sensitivity*, we give improved schemes for insertions in the presence of deletions. We define a problem to be *change sensitive* if updates to the solution due to an insertion can be obtained in time proportional to the changes. We first show improvements in the specific problem of the 2-d Convex Hull Maintenance. No assumptions are required for this problem. We then consider the general class of Order Decomposable Problems.

The convex hull of a set of points is of fundamental importance in Computational Geometry. Moreover, this geometric structure finds applications in the solution of problems in pattern recognition, path planning and graphics etc. [PS]. The algorithms using this structure often maintain the convex hull in a dynamic environment when points are inserted and deleted. Maintenance of the convex hull under insertions and deletions thus becomes of fundamental importance also. The best result for this problem is by Overmars and Van Leeuwen [OL] who gave an $O(log^2 n)$ scheme for updating the convex hull of a set of points in the plane when a point is inserted or deleted.

In this paper we first describe a scheme to improve the performance of the 2-d convex hull maintenance. We show that an amortized behaviour of $O(logn)$ for insertions and $O(log^2 n)$ for deletions can be obtained.

Other related results for specialized cases may be found in [HS1], [HS2]. These results are partial only in that the first result deals only with the case of deletions and the second result deals with an off-line version of the problem. Logarithmic cost is achievable under some restricted cases also, i.e. when updates occur only at the ends of a simple path [FHS].

W.l.o.g. we will consider the maintenance of upper hulls in this paper. These upper hulls will be stored in a balanced binary tree structure. There are two critical components in the maintenance of the convex hull. One is the construction of the edges of the convex hull obtained by constructing a common tangent between the two hulls, of two separable set of points, present at the siblings of a node in the tree. And the other is the data structure for maintenance of hulls at the nodes of the binary tree. We first describe the tangent construction and its analysis and then the details of the data structure used. There are two data structures used here. One uses $O(nlogn)$ space and allows the convex hulls at all nodes of the binary tree to be obtained in constant time. We term maintenance of such a data structure as *strong maintenance*. It is interesting to note that such a data structure can be maintained in $O(1)$ amortized time. The other uses $O(n)$ space and the convex hull at a node of the binary tree can be obtained in $O(logn)$ steps.

We hope that this methodology would be useful to obtain an optimal scheme for insertions

and deletions.

We next consider the general class of Order-Decomposable Problems defined for sets. This class of problems have been studied by Overmars [O]. We show that insertions in this framework can be speeded in the presence of deletions. We make some assumptions which we describe in detail in a later section. The first assumption is that the problems are change sensitive. Secondly, we assume a uniqueness property of elements of the solution sets w.r.t. the underlying domain. The last assumption deals with the behaviour of the data structure under changes. These assumptions are valid for the problems of Convex Hulls, Maxima, Half-space intersections and Voronoi Diagrams.

Our results show that *semi-dynamic* behavior can be achieved for insertions even in the presence of deletions. The basic strategy is to show that, at an insertion, construction of the solution at a node of the tree data structure results in pushing down the tree, elements of the underlying domain from which solution sets are constructed. Since an element can be pushed down the tree only $O(logn)$ times, the insertion time is bounded. When these elements rise in the tree, the changes in the positions of the elements of the solution sets are charged to the deletion operation.

The paper is organized as follows. Section 2 describes an outline of the data Structure for the convex hull problem and details of a critical construct, the *bridge* between two hulls. Section 3 details a $O(nlogn)$ space data structure. And section 4 describes a $O(n)$ space data structure. Section 5 considers Order-Decomposable Problems. The conclusion discusses briefly further results.

2 The Data Structure and Tangent Constructions

Let S be a set of points in the plane. The data structure for maintaining the convex hull is similar to the one used by Overmars and Van Leeuwen [OL]. W.l.o.g we will consider maintaining only the upper hull. The hull is obtained from a tree structure, called the *Convex Hull Tree*, of S. We will let $CHT(S)$ denote the Convex Hull Tree of S. Also at node $u, u \in CHT(S)$, we let $CH(u)$ represent the upper hull at u. $left(u)$ and $right(u)$ represent the left and right sons, respectively, of node u in $CHT(S)$. Finally, we let $ST(u)$ denote the set of points in the subtree rooted at node $u \in CHT(S)$. In the description below we will use $CHT(S)$ and CHT interchangeably. We let $Path(p)$ be the path from the leaf containing p to the root.

The data structure, $CHT(S)$, comprises a primary structure which is a balanced binary tree. The points are stored, sorted by x co-ordinate values, at the leaves of the tree. At each internal node there are auxiliary data structures which store the upper hull of the points in the subtree rooted at that node. At the root of the tree is stored the upper hull of S. The convex hull at each internal node of the tree is constructed from the convex hulls at the two

children of the node by constructing a common tangent between the two convex hulls. This tangent is called a bridge. The convex hulls at internal nodes are stored in auxiliary data structures. Before we describe the data structure in more detail we describe the procedure for constructing bridges assuming that the convex hulls are available at the nodes when required.

2.1 Bridge Construction

The Bridge construction procedure is a modification of the one described by Overmars and Van Leeuwen [OL]. The modification is in the tangent construction part of the insertion sub-procedure.

The following definitions are required: The line segment (p, b) is said to *include* a point a or a line segment (x, y) when a or (x, y) lies *below* the line which contains the line segment (p, b). A point lies *below* a line if the point is vertically below a point on the line. A line segment (x, y) lies below a line, l, if every point on the line segment lies below the line.

The tangents are constructed as follows:

Consider the insertion of a point p. Let $Path(p)$ be the path from the leaf, where p is inserted, to the root. Along this path let $T = (a, b)$ be a tangent at node u in the tree. Assume w.l.o.g. that a is in the same subtree as p. The tangent, T, is to be replaced if (p, b) includes (a, b). Else either (b, p) intersects the current convex hull or p is included within $CH(u)$ and hence does not affect any upper hull at nodes on $Path(p)$ above u.

Suppose a replacement tangent is to be constructed at u. Assume that $Path(p)$ uses $left(u)$ at u. A tangent is constructed from p to $CH(right(u))$. Let $CH_{ru} = (v_1, v_2 \ldots v_k)$ be the list of points on $CH(right(u))$ ordered by increasing x-coordinate value. The following procedure is used:

ALGORITHM $MODIFY - BRIDGE(p, T, u)$
begin
Let $T = (a, b)$; STOP = false;
$j \leftarrow 1$;
Let $v_i = b$;
While $j \leq \log n$ and $i \leq k$ do
begin
if (p, v_i) is a tangent to CH_{ru} then STOP=true
else
begin
$i \leftarrow i + 1$;
$j \leftarrow j + 1$;
end

end;
If STOP=false then
construct tangent from p to CH_{ru} using binary search
end.

In the above process instead of a binary search we perform a limited linear scan of the vertices to find the tangent from point p. The linear scan starts from b and continues to the right until $logn$ points are scanned. At that stage a binary search can be performed.

We keep the deletion procedure the same as in the original scheme in [OL]. Let $Path(p)$ be the path from the leaf containing the deleted point p to the root. At each node $u, u \in Path(p)$, a common tangent between $CH(left(u))$ and $CH(right(u))$ is constructed. The procedure to construct a common tangent requires $O(log^2 n)$ steps. [OL,PS].

We next analyze the time required for tangent constructions. The details of the data structure to be used will be described later.

2.2 Analysis

In this section we analyze the time required to construct the tangents. We ignore, in this section, the time required to manipulate secondary data structures. We assume that the nodes of the tree are labeled. On an insertion a node with a new label is added. On a deletion the node containing the deleted point and its label is removed. And on a single rotation the new node added to merge two subtrees has a new label. A node is also deleted during single rotations and its label is removed. Double rotations are handled as two single rotations.

The analysis uses the following sets:
$Pointset(x, u, t_x)$ is the set of points that are linearly scanned and strictly included within the convex hull, $CH(u)$ when a tangent from x is constructed at node u in the tree at time t_x, the time of insertion of x.
$Pointset(x, u, t)$ is the set of points that are contained in $Pointset(x, u, t'), t > t' \geq t_x$ and are strictly included within convex hull at u when a tangent is drawn from x to $CH(right(u))$ at time t.

$Pointset(x, u, t)$ is said to be *associated* with x and u at time t. These sets will be manipulated by insertions, deletions and rotations only for the purpose of analysis.

It follows from the construction of CHT that a point $p \in S$ is present in convex hulls at consecutive nodes u, $u \in Path(p)$, starting with the leaf node containing p. Let $Last(p)$ be the last node on $Path(p)$ at which p is present. We denote by $height(u)$ on path $Path(p)$, the distance of node u from the leaf node containing p. Furthermore, we let $h(p, t)$ denote the distance of $Last(p)$ from the leaf node containing p at time instant t. This is the *height* at which the point p is last present on a convex hull. We let $NS(p, t)$ be $|Sets(p, t)|$ where

$Sets(p,t) = \{Pointset(q,x,t) | p \in Pointset(q,x,t)\}$. $NS(p,t)$ represents the number of sets from Pointsets which contain point p at time t.

We next describe the manipulation of Pointsets and some of its properties during insertions and deletions. It is important to note that manipulation of Pointsets is for the purpose of analysis only.

In the insertion procedure the following steps are performed when p is inserted at time t_p:

1. Construction of new tangent at nodes on $Path(p)$.
2. Creation of $Pointset(p,u,t_p)$ at nodes $u, u \in Path(p)$.

The steps when a point p is deleted at time t are:

1. Construction of new tangents $T_i = (p_i, q_i)$ at nodes $u_i, u_i \in Path(p)$.
2. Removal of p from *Pointsets* and removal of $Pointsets(p,v,t), v \in CHT$.
3. Removal of *Pointsets* associated with v and containing q_i when q_i becomes part of $CH(v)$ due to deletion of p.

Since rotations will be required for balancing the tree we also consider manipulation of Pointsets at the rotations. At a single rotation at node v at time t, we ensure that points in $\cup_{p \in S} Pointset(p, left(v), t)$ and in $\cup_{p \in S} Pointset(p, right(v), t)$ are removed. Double rotations are treated as two consecutive single rotations.

We now bound the insertion and deletion times. For this purpose we show some properties of Pointsets.

The first lemma uses the fact that there are $O(logn)$ new tangents and Pointsets constructed when p is inserted. Since during deletions and rotations , Pointsets are only deleted, there are $O(logn)$ nodes, along $Path(p)$, where non-empty Pointsets associated with p exist.

Lemma 2.1 *There are $O(logn)$ Pointsets associated with a point p at any time instant.*

The following property is also true for Pointsets containing p.

Lemma 2.2 *At time $t \geq t_p$, p may be contained only in $Pointset(q,u,t)$ where $u \in Path(p)$.*

Proof: The proof is by induction on $t - t_p$. The claim is trivially true when p is inserted since p is not contained in any Pointset. Assume that the claim is true for $t-1$, $t > t_p$. Consider the operations at time t. If a point q is inserted, then p may be contained in $Pointset(q,u,t)$ where u is the least common ancestor of the leaf nodes containing p and q. p is not added to any other Pointset. The claim thus holds after insertions. When a point q is deleted p is not added to any Pointset. Finally, suppose a single rotation takes place at node v. $Path(p)$ may now contain a new node u or a node u may be removed from $Path(p)$. In the first

case there is no Pointset associated with u. In the second case this node is either $left(v)$ or $right(v)$ and after a rotation is no longer on $Path(p)$. However, all Pointsets associated with this node are removed. The induction hypothesis is thus valid for the case of single rotations also. Double rotations are treated as two single rotations.

The lemma follows. ■

We let $T_A(n)$ be the complexity of inserting n elements. The following lemma bounds the insertion complexity. Note that $Pointset(p, v, t_p)$ is not null only along $Path(p)$.

Lemma 2.3 $T_A(n) = O(\sum_{(p,v),p \in S, v \in CHT} |Pointset(p, v, t_p)| + n \log n))$

Proof: Consider insertion of point p. Let $Path(p)$ be the path from the leaf containing p to the root. At a node $v, v \in Path(p)$, the time required to construct the tangent is $T' = O(1 + |Pointset(p, v, t_p)|)$ since all points scanned during linear search, except the last one, are stored in $Pointset(p, v)$. A binary search is performed when $O(logn)$ points have been scanned. Since these points are in $Pointset(p, v, t_p)$ the time for binary search is charged to this set. To obtain the time bound we sum the quantity T' over all inserted points and all nodes on the path from each inserted point, say p, to the root. Since $|Pointset(p, v, t_p)| = 0$ for all nodes not on $Path(p)$, the lemma follows. ■

We next show another property of Pointsets. Let v be a node in the tree and let $ST(v)$ be the subtree rooted at that node. The lemma below shows that a point occurs only in one *Pointset* from amongst all the *Pointsets* constructed at the node v.

Lemma 2.4 *Let* $p, q \in ST(v)$. *If* $q \in Pointset(p, v, t)$ *then* $q \notin Pointset(r, v, t)$, $\forall r \in ST(v)$ *where* $r \neq p$.

Proof: The proof is by contradiction. Suppose $q \in Pointset(p, v, t)$ and $q \in Pointset(p', v, t)$. Let p be inserted before p'. Note that before the insertion of p, q must be on the convex hull, $CH(v)$. This follows from the inclusion of q in $Pointset(p, v, t_p)$. Now, if $q \in Pointset(p', v, t)$ then q must be on $CH(v)$ again when p' is inserted. Since $p \in S$ at this step, $Pointset(p, v, t)$ must be removed as per the manipulation of Pointsets in Step 3 of the deletion procedure. ■

As a corollary to Lemma 2.4 we obtain the following Lemma.

Lemma 2.5 $\cup_p |Pointset(p, v, t)| = O(|S(v)|)$, $\forall p \in S(v)$, *at time* t.

Next, we show a relationship between $h(p, t)$ and $NS(p, t)$.

Lemma 2.6 $NS(p, t) \leq 2logn, t \geq t_p$.

Proof: We prove the result by showing that

$$h(p, t) \leq 2logn - NS(p, t), t \geq t_p.$$

We first show that this property is maintained both under insertions and deletions ignoring rotations. The proof is by induction on $t - t_p$. It is trivially true when $t = t_p$.

On inserting a point q, tangents are constructed along $Path(q)$. The tangent construction at $v \in Path(q)$ requires either a linear scan or a binary search. If the tangent construction scans the point p during the linear scan and the height of p is reduced by at least one then p is added in the corresponding Pointset. Alternatively, the height of p may be reduced by a binary search. Then p is not included within any Pointset associated with v. In both cases the relationship $h(p,t) \leq 2logn - NS(p,t)$ is valid after the insertion.

Next, consider deletions. On deleting a point, say q, the height of points may increase. Let one such point, p, be such that $h(p,t) = h_1$. Consider a subtree rooted at a node u on $Path(p)$. Furthermore, let u be at height $< h_1$ from the leaf containing p. The following property is true:

Claim: There does not exist a point, say r, in the subtree rooted at u, such that $p \in Pointset(r,u,t)$.
The proof of this claim is by contradiction. Suppose there exists such a point r. Let $r \in ST(left(u))$ Then p would be included by a tangent drawn from r to $CH(right(u))$ and thus $h(p,t) < h_1$.

The above claim shows that point p is not contained in any Pointset associated with tree nodes at a height less than h_1 on $Path(p)$. However, the point p maybe an $\in Pointset(q,v,t)$ where v is a node such that $height(v) \geq h_1$ and $q \in ST(v)$. The number of such sets is bounded by the number of nodes with height $\geq h_1$ on the path $Path(p)$. This follows from Lemma 2.4 since p will be contained in at most one Pointset associated with each node $u, u \in Path(p)$ with u being at height $\geq h_1$. Furthermore p is not contained in any Pointset associated with nodes not on $Path(p)$ (Lemma 2.2). This covers all the possible Pointsets that p may be a member of. The induction hypothesis thus holds after deletion also.

Thus the property of the height w.r.t. the Pointsets is maintained under insertions and deletions.

Finally, consider rotations at a node v. We consider only single rotations. The new tangents constructed during rotations do not add points to any Pointset. And points in $\cup_{p \in ST(left(v))} Pointset(p, left(v), t)$ and $\cup_{p \in ST(right(v))} Pointset(p, right(v), t)$ are removed from the corresponding Pointset. Let $p \in S(v)$. By induction, $h(p, t-1) \leq logn - NS(p, t-1)$. First note that all points on $CH(v)$ are unaffected. However, $h(p,t)$ may increase by 1 since it may be present on either $CH(left(v))$ or $CH(right(v))$ when it was not before. But the deletion of points from Pointsets at $left(v)$ and $right(v)$ ensures that p is not contained in any Pointset associated with $left(v)$ or $right(v)$. There are two cases now. Either $h(p,t) = h(p,t-1)$ or $h(p,t) = h(p,t-1) + 1$. In the first case since $NS(p,t)$ may only

have decreased $h(p,t) \leq logn - NS(p,t)$. In the second case $h(p,t)$ has increased by 1. But $NS(p,t)$ has decreased by 1 if p was in a Pointset associated with $left(v)$ or $right(v)$. Thus $h(p,t) \leq logn - NS(p,t)$.

■

We now use the above properties to show the amortized bound. Over a sequence of n insertions and m deletions, points are added and deleted from the pointsets.

First, consider the work done during the deletion of a point p at time t along the path $Path(p)$ leading from the root of the tree to the leaf.

Step 1 of the deletion procedure requires $O(logn)$ steps at each node $u, u \in Path(p)$. Step 2 also requires $O(logn)$ steps at each node since there is only one pointset associated with p at a node u on $Path(p)$ and $|Pointset(p,u,t)| = O(logn)$. To bound the complexity of Step 3 we need the following lemma:

Lemma 2.7 *Let* $NewP(p,u,t) = \{q | q \in CH(u)$ *only after deletion of* p *and* $\exists q'$ *s.t.* $q \in Pointset(q',u,t)\}$. *Then* $|NewP(p,u,t)| = 1$.

Proof: The proof is by contradiction. Suppose there are two points x and y such that both points have become part of $CH(u)$ after the deletion of point p and each point is contained in a pointset associated with u. Let $x \in Pointset(r,u,t)$ and $y \in Pointset(r',u,t)$ where both r and r' are in S at time t. If $r = r'$ then either the line containing (r,x) includes y or the line containing (r',y) includes x. Thus both points cannot be part of the convex hull obtained by constructing the tangent from r to $CH(right(u))$. If not, i.e when $r \neq r'$, then assume w.l.o.g. that r' was inserted after r. If $x \in Pointset(r,u,t)$ then the tangent from r' replaces the current tangent at u at time $t_{r'}$. Since $CH(u)$ includes x at time $t_{r'}$, (r',y) includes x. Thus both points x and y cannot be part of $CH(u)$ when r' is present. ■

By the above lemma, Step 3 requires $O(logn)$ operations at each node on $Path(p)$ since one Pointset is removed. The deletion process thus requires $O(log^2 n)$ steps.

Moreover, at rotations at a node v, points are removed from *Pointsets* of points in the subtree rooted at v. By Lemma 2.5 above, this takes $O(n_1)$ steps where n_1 is the size of the subtree rooted at that node. Each such removal is charged to the rotation at that node. Note that each insertion or deletion operation is charged $O(logn)$ operations with $O(1)$ time allocated to each node on the path from leaf to root [BM]. Since a rotation occurs after cn_1 steps at node v, the removal of nodes from the *Pointsets* during rotations is accounted by the charges allocated to that node during insertions and deletions.

We can now derive the complexity of the tangent finding operations, $T_A(n,m)$ after n insertions and m deletions.

Lemma 2.8 *Bridge Constructions during* n *insertions and* m *deletions require* $O(nlogn + mlog^2 n)$ *operations*

Proof: It is easy to see that
$T_A(n,m) = O(mlog^2n + nlogn + ChngPs)$
where $ChngPs$ is the sum of all changes that have occurred in all the *Pointsets* during insertions and deletions. This includes the creation of the Pointsets. But the changes in the *Pointsets* where a point is added and subsequently removed is charged to the deletion operation. Thus
$T_A(n,m) = O(mlog^2n + nlogn + SizePs)$
where $SizePs$ is the size of all the Pointsets at the end of the insertions and deletions. These pointsets contain points added but not removed. Now each point could be in at most $O(logn)$ Pointsets (By Lemma 2.6 and the fact that $h(p,t) \geq 0$). $SizePs$ is thus bounded by $O(nlogn)$ as $O(n)$ points are present in the tree. ■

3 Data Structure Details

The first data structure that we describe comprises a primary structure which is a balanced tree, CHT. At each node, v, of the balanced tree is stored the sorted list of point , $A(v)$, in the subtree rooted at v and the upper hull, $U(v)$, of the vertices in the subtree, $ST(v)$ rooted at that node. This hull is stored in the form of a linked list. It is stored as a sublist of the list of points, $A(v)$. To obtain $U(v)$ we maintain two edges incident onto p in $A(v)$ called $LE(p,v)$ and $RE(p,v)$. $LE(p,v)$ is an edge with the other endpoint to the *left*, i.e. with x-coordinate less than that of p. And $RE(p,v)$ is an edge with the other endpoint to the *right*, i.e. with x-coordinate greater than that of p. During insertions and deletions these edges change. We let $CLIST(q)$ be the set of points, p, such that insertion of q destroys a bridge incident onto p. It is easy to see that $|CLIST(q)| = O(logn)$. These edges, stored in $A(v)$, satisfy the following property:
Property Uppermost:

1 If point $p \in ST(v)$ is on the upper hull $U(v)$ then the two edges of the convex hull incident onto p, one to the left and the other to the right, are assigned to $LE(p,v)$ and $RE(p,v)$, respectively, in the list $A(v)$.

2 If $p \in ST(v)$ is not on the convex hull and is not in $CLIST(q)$ for a node $q \in ST(v)$ then $LE(p,v)$ and $RE(p,v)$ are $LE(p,w)$ and $RE(p,w)$ where w is the first descendant of v at which p is on the convex hull.

Note that using this property, $U(v)$ can be obtained from $A(v)$ as follows: Start from the leftmost node, say p, in $A(v)$ and proceed to the node given by $RE(p,v)$ which is the next node on $U(v)$. Repeating this scan procedure with the new node gives the upper hull in sorted order.

Also stored at each node, v, of the tree CHT is the bridge, $B(v)$ constructed at that node. Moreover, with each point p is associated a list data structure $L(p)$ which lists, in order, bidirectional pointers. Each pointer is a bidirectional link to a list record which stores point p in the list of points, $A(v)$, where v is a node on $Path(p)$. We denote this pointer by $L(p,v)$. A pointer is stored for each occurrence of p on $Path(p)$. The pointers in $L(p)$ are ordered as follows: $L(p,v)$ occurs before $L(p,w)$ iff v occurs before w on the path $Path(p)$. Note that $|L(p,v)| = O(logn)$

We now describe how to implement the tangent construction procedure described in the previous section. The tangent construction involves two kinds of searches. Firstly a linear search and then a binary search. The linear search at node v is performed on the list of nodes on the upper hull, $A(v)$.

The binary search is performed using the bridges at the nodes. It proceeds as follows: Let l and r be two points such that the point on the hull, $U(v)$, to which the tangent from the inserted point p is to be found, lies in between l and r. Initially these points may be chosen to be the extreme points in the point set. Let $B(v) = (x,y)$ be the bridge at node v. Assume w.l.o.g. that $x(p) < x(y)$. Suppose (p,y) is convex w.r.t the convex hull $U(right(v))$ but intersects $U(v)$. Then the search proceeds in the left subtree at node v with r as x. Alternatively, if (p,y) is concave w.r.t. to $U(right(v))$ then the search proceeds in the right subtree with y as l. The search stops when either a tangent is discovered from p to $U(v)$ at y or a leaf node in CHT is reached. In the second case the tangent is (p,l) where l is the point stored at the leaf node.

The auxiliary data structures are updated after all the tangent constructions during an insertion or deletion of nodes.

Before we give further details of an efficient technique of updating of the auxiliary structures we note that we are storing sorted lists at each of the nodes. These lists can be also be maintained using fractional cascading and can be maintained in a dynamic environment using techniques in [MN].

3.1 Updating the auxiliary data structures

We first consider insertions. On constructing a tangent from a point p at a node v the edges in list $A(v)$ and the upper hull $U(v)$ at the node needs to be modified. These modifications require locating the records containing the endpoint of the tangent in the list $A(v)$. There are two cases.

1 If the tangent (p,q) is found by a linear search then the location of q in the current convex hull, $U(v)$, is obtained during the linear scan.

2 On the other hand if the tangent (p,q) is found by a binary search then the list $L(q)$ is used to locate the record in $U(v)$ representing q.

In both these cases $A(v)$ and hence $U(v)$ is updated as follows: Assume w.l.o.g. that q is to the right of p.

- In the first case, the pointer $RE(p,v)$ is modified to point to the record containing q in $A(v)$. A pointer to this record is available during the linear search.
- In the second case, $RE(p,v)$ is modified in the same fashion. The pointer to q, $L(q,v)$, is obtained from $L(q)$.

Finally, the endpoints of the tangent removed are added to $CLIST(p)$.

The time required for the update is the same as the time required to find the tangent. Moreover, $A(x)$ is required to be changed due to insertion of p and its associated edges for all nodes, x, on $Path(p)$ from v to either the root or the next node at which another tangent from p is constructed. This is to maintain **Property Uppermost**. Since the pointer to $L(q,x)$, x on $Path(p)$, with x initially v, is available from $L(q)$, all the changes can be performed in $O(1)$ steps each. Finally the pointer list for p, $L(p)$ is built in time $O(logn)$ when point p is inserted since the pointers are available in sorted order. Thus the updates to the auxiliary data structures after an insertion requires $O(logn)$ steps.

Consider deletion of a point r next. Edges are removed due to deletion of r. And new edges added due to bridge construction. First consider the case of addition of new edges. Let $e = (p,q)$ be the bridge edge added at a node u. Assume that q is to the right of p. The records containing p and q at u are found using $L(p)$ and $L(q)$ by a linear search requiring $O(logn)$ steps. $A(u)$ is then modified. The bridge edge B is also required at ancestor nodes since **Property Uppermost** needs to be maintained. Thus this edge, (p,q), modifies $RE(p,x)$, ($LE(q,x)$) in the records of p, (q respectively) in $A(x)$, for each node x on the path from u to the root until

(1) Either the root is reached or

(2) A node is reached where a different edge from p, (q respectively), exists on the convex hull and e lies "below" that edge.

This requires $O(logn)$ time for each bridge (p,q), constructed since it requires a scan of $L(p)$ and $L(q)$. As there are $O(logn)$ bridges constructed we get a time bound of $O(log^2 n)$.

Next, consider the case when an edge incident on a point p at tree node v is removed and the bridge constructed at v is not incident onto p. W.l.o.g assume that this edge, e_p has its other endpoint with x-coordinate greater than p i.e. is $RE(p,v)$. The edge incident on p in $A(v)$ is then replaced by the hull edge, $e'_p = RE(p,v_1)$, incident on p at v_1, the son of v which lies on $Path(p)$. The edge e'_p also replaces edge e_p in $A(x)$ at ancestor nodes, x, until

(1) Either the root is reached or

(2) A node is reached where a different edge from p exists on the convex hull and e'_p lies "below" that edge.

$O(logn)$ time is required for this update. There are $O(logn)$ such updates since $O(logn)$ edges of the form (p,r) are removed during bridge constructions and a total of $O(log^2n)$ time is required. The following operations are also performed.

(a) LE and RE edges incident onto points in $CLIST(r)$, which are currently in the point set, are updated by constructing tangents and propagating them up the convex hull tree.

(b) $L(r)$ is removed.

The first step requires $O(log^2n)$ steps since there are $O(logn)$ points in $CLIST(r)$. The second step requires $O(logn)$ steps. We show below that the above steps ensure that for all points affected by the deletion procedure, either due to an addition or deletion of an edge, **Property Uppermost** is satisfied. A total of $O(log^2n)$ time is required for the deletion procedure.

We finally consider the work done during rotations. At every rotation, at a node v, the lists and convex hulls are updated in $O(n_1 + logn)$ time when there are n_1 nodes in the sub-tree rooted at v. The lists at node, v, are rebuilt using the lists at the sons, u and w. Furthermore bridges are constructed using binary search. The changes, due to new bridge constructions, in lists $A(x)$, for nodes x on the path from u to the root, are implemented as in the insertion or deletion case in $O(logn)$ steps. The rebalancing occurs after cn_1 insertions or deletions in the subtree rooted at v. Each such insertion and deletion contributes a unit charge to v and this charge accounts for the work during the rotation at v. Moreover since each insertion or deletion contributes $O(logn)$ charges to nodes along the insertion or deletion path, $O(nlogn)$ total charges are required for n insertions and deletions during rotations.

We prove correctness of the data structure maintenance by the following lemma:

Lemma 3.1 Property Uppermost *is maintained during a sequence of insertions and deletions.*

Proof: The proof is by induction on the number of insertions and deletions. For the base case when no insertions or deletions have been performed, the initial tree satisfies the **Property Uppermost**. For the induction step, assume that the property is satisfied upto the ith step. We now consider the $i+1$st step.

First consider the case of insertions. Suppose the insertion of p to the set of points in $A(u)$ at a node u changes the convex hull and the bridge edge constructed is (p,r). The bridge edge (p,r) is added to the list $A(u)$ and to $A(x)$ for all x on $Path(p)$ until either the root is reached or a new tangent from p to the right, which includes the edge (p,r) , is added to $U(x)$. Thus the property is maintained for p at nodes in CHT.

Next consider another point q in $A(u)$. If q was on the convex hull and is no longer on the convex hull then there are two cases. Either the bridge removed due to insertion of p was from q in which case q is added to $CLIST(p)$. Else q does not have the deleted bridge incident onto it and thus the current edges incident to q are the same as the ones incident to q before the insertion. If q was on the convex hull and still is, then either the edges incident onto it are unaffected or a new bridge is incident onto it. In the first case no change is required to be made. In the second case, the new bridge edge is propagated upwards during the insertion procedure in order to maintain **Property Uppermost**. Finally, if q was not on the convex hull at u then the edges to q remain unaffected. This is true for all nodes in $Path(p)$. Points at nodes not on the path remain unaffected. Thus **Property Uppermost** is satisfied after the insertion operation ignoring rotations.

Now consider deletion of a point q. Let v be a node where $LE(p,v)$ or $RE(p,v)$ is changed. Assume w.l.o.g. that the edge to the right, $RE(p,v)$, is changed. If p was on the convex hull and is still on the convex hull then there are two cases: Either the edge is the new bridge edge in which case the updating of $A(v)$ is correct by construction. Or the edge is obtained from w, the son of v, where p is on the convex hull. This update is correct since the bridge has changed so that it is a tangent to the convex hulls at the left and right sons of v at a point p' such that $x(p') > x(p)$. Thus the edge from p in the convex hull at w is in the convex hull at v.

Next, consider when p was not on the convex hull and now is. This occurs when the bridge at v changes. The point p must be on the convex hull at the son of v and the relevant incident edges already exist correctly at v since by induction, **Property Uppermost** is maintained at the ith step.

Finally, consider the case when p was not in $CH(v)$ before the deletion and is still not on $CH(v)$. This does not affect the convex hull edge at v. However $LE(p,v)$ or $RE(p,v)$ may have changed due to changes at convex hulls at descendant nodes. This change is correctly propagated up along the path $Path(p)$ in the deletion procedure.

Lastly, consider rotations at node v. The lists at nodes involved in the rebalancing are completely rebuilt using lists at the immediate descendant nodes. Thus the points in the lists $A(v)$ have correct edges incident onto them. The lists at nodes above v are modified only due to change in the bridge edges at nodes involved in the rotations. That these modifications are made correctly follows from the correctness of the propagation of the change along the path from v to the root. Also lists at those nodes in the subtree rooted at v not involved in the rotations are also unaffected. Thus the lemma is proved. ■

The space requirement of this data structure is $O(nlogn)$.

We thus have the following result where we show that the convex hulls at each of the nodes of the balanced tree CHT can be maintained efficiently:

Theorem 1 *Strong Maintenance of the Convex Hull Tree requires $O(nlogn + mlog^2n)$ operations and $O(nlogn)$ space when there are $O(n)$ insertions and $O(m)$ deletions.*

4 An $O(n)$ space solution

To get an $O(n)$ space solution for maintenance of the convex hull we use the technique of Overmars and Van Leeuwen and store portions of the convex hull.

Each point will be stored in a record. This record will be part of a list which stores a complete or partial convex hull. Moreover, from the leaf containing point p there is a pointer to the record containing p.

The lists are stored as follows: Suppose in constructing the convex hull at node v, $CH(v)$, in the tree CHT, we eliminate the sequence of points L and R where L is part of the convex hull at the left son, v_l, and R is part of the convex hull at the right son, v_r. L and R are stored at v_l and v_r respectively. The root stores the convex hull of all the points.

Let v be a node in the tree CHT. Suppose we have $CH(v)$ available in the form of an ordered linked list. It is easy to see that using the hull portions stored at v_l and v_r and the position of the bridge in $CH(v)$ the convex hull at the sons of v can be constructed in constant time. Let P be a path from the root to a leaf node in CHT. The convex hull at nodes on the path and their immediate descendants can thus be constructed in $O(logn)$ steps.

Using these hulls it is easy to use linear search to construct tangents. Binary search is performed using the bridges at the nodes of the tree CHT as in the previous section.

To update the convex hull structures after a bridge is constructed at node v we need to determine the position of the points defining the bridge in the linear lists storing the convex hulls. The following data structure is used:

- A pointer from the leaf node containing a point q to the record storing the point q in a list at some tree node v.

The position of bridge points is thus easily determined when the tangent is constructed at a node v. When linear search determines the tangent in the list of convex hull edges at the current node the position of the points is immediately known. When binary search is used to locate a tangent endpoint, say q, on the convex hull at node v, the position is given by a pointer from the leaf node containing q to the record storing the point, q, in the linear list forming the convex hull at node v. The position of the two points of the bridge is used to split the convex hull at v into the required parts in $O(1)$ time.

The changes in the tree T during rotations can be handled similarly. Suppose a rotation performed at a node v. The convex hull at the immediate descendants of v can be obtained in $O(logn)$ steps. The required bridges can be constructed in $O(logn)$ steps using binary

search and the modifications to the convex hulls sublists, stored at the nodes of the tree involved in the rotation, can be done as above. Thus each rotation requires $O(logn)$ steps to perform.

We thus have proved the following result:

Theorem 2 *The convex hull of a set of points can be maintained in $O(nlogn + mlog^2n)$ operations and $O(n)$ space when there are $O(n)$ insertions and $O(m)$ deletions.*

5 Order-Decomposable Problems

In this section we show how to extend the results for the convex hull problem to a larger class of problems termed as *order decomposable problems.*

Definition Let S_1, S_2 and S_3 be sets and let $P : 2^{S_1} \rightarrow S_2, S_2 \subseteq 2^{S_3}$ be a set problem. P is *order decomposable* if there is a linear order $<$ and an operator $\Delta : S_2 \text{ x } S_2 \rightarrow S_2$ such that for every subset $S \subseteq S_1, S = \{a_1 < a_2 < \ldots < a_n\}$ and every i

$$P(\{a_1, \ldots, a_n\}) = \Delta(P(\{a_1, \ldots, a_i\}), P(\{a_{i+1}, \ldots, a_n\})). \quad (1)$$

S_3 is an underlying domain. We will let $|S|, S \in S_2$ represent the size of the set S. S_2 comprises sets of elements from S_3 satisfying some desired property. Δ is termed as a merge operator and the time required to compute it is termed as *merge time.*

Example 1. For the convex hull problem S_1 is a set of points from the plane and S_2 is the set of convex polygons, each convex polygon being defined by a sequence of points. S_3 is thus the set of points in the plane, again.

We make some assumptions. The first assumption we make is about the behaviour of Δ.

Assumption 1 *If $S_a \in S_2$ changes by one element p, then $S_c = \Delta(S_a, S_b)$ can be updated in time proportional to $\min(\delta S_c, C(n))$, where δS_c = Change in the size of the set S_c and $C(n)$ is the merge time, i.e. the time required to merge the solutions of two subproblems of total size n.*

We call such a problem a *change sensitive order decomposable problem.*

In our analysis below we assume the following behaviour of the solutions.

Assumption 2 *$P(S_a) \cap P(S_b) = \Phi$ when S_a and S_b are disjoint.*

This assumption is reasonable for most problems since the solution, $P(S)$, is dependent on the set S.

We first show the following Monotonicity property:

Lemma 5.1 Monotonicity Property: *Let P be an order decomposable problem. Let $p \in Pset, Pset \in S_2$. Suppose $p \in P(\{a_{k'} \ldots a_{l'}\}), l \leq l' \leq k' \leq k$. If $p \notin P(\{a_k, \ldots a_l\})$ then $p \notin P(\{a_i, \ldots a_j\})$, $i \leq k \leq l \leq j$.*

Proof: The proof is by contradiction. Suppose it is, i.e. $p \in P(a_i, \ldots a_j)$. However,

$$P(\{a_i, \ldots a_j\}) = \Delta(P(\{a_1, \ldots, a_{k-1}\}), \Delta(P(\{a_k, \ldots, a_l\}), P(\{a_{l+1}, \ldots, a_j\})))$$

Moreover, by Assumption 2, $p \notin P(\{a_i \ldots a_{k-1}\})$ and $p \notin P(\{a_{l+1} \ldots a_j\})$. Thus $p \in P(\{a_k, \ldots a_l\})$. ■

We now detail the data structure used for dynamic maintenance of order decomposable problems.

We use a weight balanced tree $\mathcal{TP}(S)$, where S is the input set. At the leaves are the elements of the input set S , arranged in order. At each internal node, v, is stored an auxiliary data structure comprising the set $P(ST(v))$, where $ST(v)$ is the subtree rooted at v. We will use $P(ST(v))$ and $P(v)$ interchangeably. $P(ST(r))$, where r is the root of $\mathcal{TP}(S)$, is the solution to the problem.

We make the following additional assumption on the data structure:

Assumption 3 *Let $S_c = \Delta(S_a, S_b)$. Then the auxiliary data structure, storing S_c, can be updated in time proportional to $\min(\delta S_c, C(n))$, where δS_c = Change in the size of the set S_c and $C(n)$ is the time to merge two subproblems of total size n.*

It is easy to see that the assumptions above are valid for the problems of maintenance of 2-d Convex Hulls and for 2-d Voronoi Diagrams given the location of the point in the Voronoi subdivision.

We now describe a scheme for dynamically updating $\mathcal{TP}(S)$ so as to improve the insertion time.

As in the solution for the convex hull problem, we maintain sets called *Pointsets* which help in the analysis.

- *Pointset*(x, u, t_x) is the set of elements that are eliminated from $P(ST(u))$, with size bounded above by $C(n)$, when $x \in S_1$ is added to $ST(u)$ at time t_x, the time of insertion of x.
- *Pointset*(x, u, t) is the set of elements that are contained in *Pointset*$(x, u, t'), t_x \leq t' < t$ and are absent from $P(ST(u))$ at node u at time t.

Pointset(x,u,t) is said to be *associated* with x and u at time t. These sets will be manipulated by insertions, deletions and rotations only for the purpose of analysis.

As in the previous problem we note that an element $p \in S, S \in S_2$ is present in the solution to P only at nodes u s.t. $u \in Path(x)$ for some x where $Path(x)$ is the path from the leaf node to the root node. Let $Last(p)$ be the last node on $Path(x)$ at which p is present. And we denote by $height(u)$, the distance of node u from the leaf node containing x on path $Path(x)$. Furthermore, we let $h(p,t)$ denote the distance of $Last(x)$ from the leaf node containing x at time instant t. This is the *height* at which the element p is last present on a convex hull. We let $NS(p,t)$ be $|Sets(p,t)|$ where $Sets(p,t) = \{Pointset(x,u,t) | p \in Pointset(x,u,t)\}$. $NS(p,t)$ represents the number of sets from Pointsets which contain the element p at time t.

We next describe the manipulation of Pointsets and some of its properties during insertions and deletions. It is important to note that manipulation of Pointsets is for the purpose of analysis only.

Consider insertion of x. The following steps are required.

1. Construction of updated $P(ST(u))$ at nodes $u, u \in Path(x)$.
2. Creation of $Pointset(x,u,t_x)$ at nodes u such that $u \in Path(x)$.

The steps when an element x is deleted at time t are:

1. Construction of updated $P(ST(u))$ at nodes u s.t. $u \in Path(x)$.
2. Removal of p from *Pointsets* and removal of $Pointset(x,u,t), u \in \mathcal{TP}$.
3. Removal of q_i from the *Pointset* containing it when q_i becomes part of $P(ST(v))$ due to deletion of x.

Finally, consider rotations. At a single rotation at node v at time t, we ensure that points in $\cup_{x \in S_1} Pointset(x, left(v), t)$ and in $\cup_{x \in S_1} Pointset(x, right(v), t)$ are removed. Double rotations are treated as two consecutive single rotations.

We next state some properties of Pointsets. The proof of the following lemma is similar to that of Lemma 2.1 and hence omitted.

Lemma 5.2 *There are $O(logn)$ Pointsets associated with a point $x \in S_1$ at any time instant.*

The following property is also true for Pointsets containing p

Lemma 5.3 *At time $t \geq t_p$, p may be contained only in $Pointset(x,u,t)$ where $u \in Path(x)$, for some x.*

Proof: First note that by Assumption 2, p exists only in the solution set at nodes on one path in the tree. Let $Path(x)$ be such a path. Pointsets associated with x and with labeled nodes on $Path(x)$ may cease to exist on this path because of rotations. But the Pointsets associated with nodes involved in the rotation are removed. The lemma follows. ■

We let $T_A(n)$ be the complexity of inserting n elements. The following lemma bounds the insertion complexity. The proof is similar to the one in the previous section and hence omitted.

Lemma 5.4 $T_A(n) = O(\sum_{(x,v),x\in S_1,v\in\mathcal{TP}} |Pointset(x,v,t_x)| + n\log n)$

The lemma below shows that a point occurs only in one *Pointset* from amongst all the *Pointsets* constructed at the node v.

Lemma 5.5 *Let* $p \in P(ST(v))$. *If* $p \in Pointset(x,v,t)$ *then* $p \notin Pointset(y,v,t)$, $\forall y \in ST(v)$ *where* $y \neq x$.

Proof: Suppose $p \in Pointset(x,v,t)$. Then $p \in Pointset(x,v,t_x)$ and thus $p \in P(v)$ at time $t \leq t_x - 1$ but $p \notin P(v)$ at time $t \geq t_x$. If y is inserted before x then obviously $p \notin Pointset(y,v,t_y)$ and hence $p \notin Pointset(y,v,t)$. If y is inserted after x then we come to the same conclusion since $p \notin P(v)$ at time $t, t > t_x$. ■

The following lemma establishes a bound on the number of Pointsets containing p.

Lemma 5.6 $h(p,t) \leq 2logn - NS(p,t), t \geq t_x$ *for some* x.

Proof: We show that this property holds under insertions, deletions and rotations. The proof is by induction on $t - t_x$, where p becomes part of a solution set after inserting x.

On an insertion of some point q, the point p may be eliminated from $P(v)$ for some tree node v. By an argument similar to that in Lemma 2.6 the height relation, $h(p,t) \leq 2logn - NS(p,t)$, is satisfied.

Next, consider deletions. Suppose $h(p,t)$ increases after a deletion. Let u be a node at height $< h(p,t)$. The following claim is true:

Claim: There does not exist an element, say y, in the subtree rooted at u, such that $p \in Pointset(y,u,t)$.

The proof of this claim is by contradiction. Suppose there did exist such a point. Then by the Monotonicity Property, p cannot be in the solution set at the node at height $h(p,t)$. The remainder of the proof is similar to that in Lemma 2.6.

Finally, consider rotations. Since points are removed at the nodes involved in the rotation, if the height of p increases then p must be removed from a Pointset. This is achieved in the rotation procedure by the removal of all Pointsets associated with the nodes $left(v)$ and $right(v)$, when a rotation is performed at node v. The relationship of $h(p,t)$ with $NS(p,t)$, i.e. $h(p,t) \leq 2logn - NS(p,t)$ is thus maintained. The details of the proof are again similar to that of Lemma 2.6. ■

We can now derive the complexity of the tangent finding operations, $T_A(n,m)$ after n insertions and m deletions. We need the following parameters.

- $Size(n)$ is defined to be the number of distinct elements constituting sets in S_2 in the data structure $\mathcal{TP}$, i.e. $Size(n) = |\cup_{1\leq i\leq t}\{p|p \in P(v), v \in \mathcal{TP}(t_i)\}|$,
  where $\mathcal{TP}(t_i)$ is the tree data structure at the ith time instant. $Size(n)$ is the total number of elements of S_3 that are created during the insertions and deletions.

- δR is the maximum number of elements in *Pointsets* added to a solution, $P(v), v \in \mathcal{TP}$ during a deletion. This represents the number of elements in Pointsets that may change height at a node in the tree data structure during a deletion operation. In fact, δR is bounded above by the changes in $P(v), v \in \mathcal{TP}$, during a deletion.

Theorem 3 *$\mathcal{TP}$, the data structure which maintains the solution to the change sensitive order decomposable problem, P, can be updated during n insertions and m deletions in $O(Size(n)logn + m(C(n) + \delta R)logn)$ operations where $C(k)$ is the time required to merge two solutions of total size k.*

Proof: First consider the time complexity of insertions. The time complexity of the insertions is $T_A(n)$. This includes the time for updating the solution to the problem and constructing Pointsets.

We next consider the time complexity of deletions. Let p be the deleted point. The time to update the solution $P(ST(u))$ at a node $u \in \mathcal{TP}$ is $O(C(n))$. This update is required at each node on the deletion path. We next consider the time required to manipulate Pointsets at each node on $Path(p)$. Step 2 of the deletion procedure requires $O(C(n))$ steps for each node giving a total of $O(C(n)logn)$ steps. Step 3 requires $O(\delta R)$ steps at each node. The time complexity of deletions is thus $O(m(C(n) + \delta R)logn)$.

We next show that the amortized time complexity of rotations is $O(Size(n)logn + C(n)logn)$. Every insertion and deletion is charged $O((Size(ST(v)) + C(|ST(v)|))/|ST(v)|)$ for each node v, $v \in Path(p)$ where p is the inserted or deleted node. $Size(ST(v)) = \Sigma_{u\in ST(v)}P(u)$, $C(|T(v)|$ is the time required to merge the solutions at $left(v)$ and $right(v)$and $|ST(v)|$ is the number of nodes in $ST(v)$. We next show that these charges suffice to pay for the recomputation of solutions and removal of Pointsets since rotations at a node v occur after $c|ST(v)|$, c a constant, operations. The time required for Pointset manipulations during rotations at a node v is $O(Size(ST(v))$ since Pointsets associated with the left and right son of v are removed. This is charged to the insertion or deletion of points in $ST(v)$ since each such manipulation occurs after $O(|ST(v)|)$ steps. The total charges for all Pointset manipulation during rotations equals $\Sigma_{v\in\mathcal{TP}}Size(ST(v)) = O(Size(n)logn)$. Next, consider the time required for recomputing the solutions at the left and right son of v and at v itself. This time is $C(|ST(v)|)$ and is charged to the insertion or deletion as described above. Note that $C(|ST(v)|)/|ST(v)|$ is bounded by $C(n)/n$ for linear or superlinear $C(n)$. Thus if every inserted point is charged $(C(n)logn)/n$, reconstruction of the solution at each node during

a rotation can be charged to the inserted point. The total charges are $C(n)logn$. This gives the desired bound.

We finally bound $T_A(n)$. Note that $T_A(n) = SizePs + RemPs$ where $SizePs = O(\sum_{(p,v),p\in S,v\in\mathcal{TP}} |Pointset(p,v,t_f)|)$, and t_f is the time at the end of the insertions and deletions. $RemPs$ is the number of points removed from Pointsets during deletions and rotations. $RemPs$ is bounded by $O(m(C(n)+\delta R)logn)$. Furthermore $SizePs = O(Size(n)logn)$ since each element can occur in $O(logn)$ Pointsets. ■

The above result can be strengthened when $C(n) = O(n)$ resulting in removal of the $logn$ factor in the second term of the time complexity.

We next consider some specific problems:

- For the Convex Hull problem, $Size(n) = O(n)$ and $\delta R = O(logn)$ as proven before.
- For the Voronoi Diagram problem, $\delta R = O(n)$ and $Size(n)$ = No. of distinct Voronoi vertices created, which is $O(n^2)$ in the worst case. It is interesting to note that at any time instant there are only $O(nlogn)$ elements (Voronoi vertices and edges) in $\mathcal{TP}$. The above result gives a scheme for maintenance of Voronoi Diagrams where n insertions require $O((N_V + nlog^2n)logn)$ steps in the presence of deletions. Here N_V is the number of distinct Voronoi vertices created. A data structure for planar point location is required to be maintained, contributing the $nlog^2n$ factor as the merge time [PT]. The deletion time is $mO(nlog^2n)$ for m deletions. The straightforward approach to this problem would require at least $O($ No. of structural changes $)$ steps.

6 Conclusions

In this paper we have shown that semi-dynamic behaviour is possible for insertions in the presence of deletions.

For the convex hull problem, note that in the worst case, insertions require $O(log^2n)$ time and thus the data structure strictly improves on the time complexity of convex hull maintenance.

We further note that the methodology for storing the convex hulls can be improved so as to give a simpler $O(n)$ space data structure than the one described by Overmars and Van Leeuwen [OL]. The scheme is along the same lines as described for the maxima problem [Ka]. A constant amount of space is used per node in the tree. (details omitted)

7 Acknowledgements

I would like to thank Herbert Edelsbrunner, Kurt Mehlhorn, Sandeep Sen and Michiel Smid for helpful comments. I would also like to thank MPI, Germany for generous support during

parts of this work.

References

[1] [BM] N. Blum and K. Mehlhorn, On the Average Number of Rebalancing operations in Weight Balanced Trees, Theoretical Computer Science, 11(1980), 303-320.

[2] [FHS] J. Friedman, J. Hershberger and J. Snoeyink, Compliant motion in a simple polygon, Proc. 5th ACM Symposium on Computational Geometry, 175-186, 1989.

[3] [HS1] J. Hershberger and S. Suri, Application of a semi-dynamic convex hull algorithm, Proc. of the 2nd Scandinavian Workshop on Algorithm Theory, pp. 380-392, Springer-Verlag, 1990.

[4] [HS2] J. Hershberger and S. Suri, Offline Maintainence of Planar Configurations, Proceedings of SODA, 1991, pp. 32-41.

[5] [Ka] Sanjiv Kapoor, Dynamically Maintaining Maxima in 2-dimension, Proceedings of the 10th ACM Conf. on Computational Geometry, 1994.

[6] [Mb] K. Mehlhorn, Data Structures and Algorithms 3, Multi-dimensional Searching and Computational Geometry, Springer Verlag.

[7] [MN] K. Mehlhorn and S. Naher, Dynamic fractional Cascading, Algorithmica 5 (1990), pp.215-241.

[8] [O] Overmars, M.H., Dynamization of order decomposable set problems, J. of Algorithms 2, 245-260.

[9] [OL] M. Overmars and J. Van Leeuwan, Maintainence of configurations in the plane, Journal of Computer and System Sciences, 23, pp.166-204, 1981.

[10] [PS] F.P Preparata and M. I. Shamos, Computational Geometry- An Introduction, Springer Verlag, (1985).

[11] [PT] F.P. Preparata and R. Tamassia, Dynamic Planar Point Location with Optimal Querry Time, TCS, Vol. 74, No. 1, 95-114, 1990.