Semi-dynamic Maintenance of the Width of a Planar Point Set*

— Abstract —

Christian Schwarz[§]

1 Introduction

Let S be a set of n points in the plane. The width of S, denoted by $W(S)$, is defined as the smallest distance of two parallel lines enclosing S. Another characterization of $W(S)$ is as follows. Let $CH(S)$ denote the convex hull of S, and let e and v be an edge and a vertex of $CH(S)$, respectively. Furthermore, let ℓ_e be the supporting line of e and ℓ_v be the line through v that is parallel to ℓ_e. The pair (e, v) is called an *antipodal pair* of $CH(S)$ if ℓ_e and ℓ_v support $CH(S)$. Using this terminology, $W(S)$ is the minimum distance between lines ℓ_e and ℓ_v, taken over all antipodal pairs (e, v) of $CH(S)$, see [3]. In [3], it is shown how to compute $W(S)$ for a given set S in $O(n \log n)$ time and $O(n)$ space: first, compute the convex hull $CH(S)$, and then generate the — only $O(n)$ — antipodal (edge-vertex) pairs in linear time. Thus, if the convex hull is at hand, $W(S)$ can be computed in $O(n)$ time.

The dynamic width problem is to maintain $W(S)$ while the set S is modified by insertions and deletions of points. In this paper, we consider the *semi-dynamic* width problem. We want to maintain $W(S)$ when the update sequence consists either of insertions only or of deletions only. We require the updates to be performed *on-line*, i.e. each update operation must be completed before the next update request is specified. The goal is to find algorithms that beat the trivial $O(n)$ bound obtained by updating the convex hull using any efficient dynamic or semi-dynamic convex hull algorithm and then recomputing the width in linear time by generating and checking all antipodal pairs.

One thing that makes dynamic maintenance of the width of a point set difficult is that the width problem is not decomposable. Actually there is no on-line algorithm known that maintains the width in less than linear time under insertions or deletions. There is, however, an algorithm by Janardan [4] that maintains an *approximation* of $W(S)$ under insertions and deletions.

Theorem 1 (Janardan [4]) *Let $\alpha \geq 1$ be an integer-valued parameter. Then there is a data structure that uses $O(\alpha(n))$ space and supports the following operations in $O(\alpha \log^2 n)$ time:* insert a point into S, delete a point from S *and* report $\widetilde{W}(S)$, *where* $\widetilde{W}(S)/W(S) \leq \sqrt{1 + \tan^2 \frac{\pi}{4\alpha}}$. *All bounds are worst case.*

Note that good approximations are already available for constant values of α. For example, $\widetilde{W}(S)/W(S) \leq 1.01$ for $\alpha = 5$, and $\widetilde{W}(S)/W(S) \leq 1.0001$ for $\alpha = 16$.

Based on the method of Theorem 1, we show

Theorem 2 *Let α and $\widetilde{W}(S)$ be as described in Theorem 1. There is a data structure that reports $\widetilde{W}(S)$ in $O(\alpha \log n \log\log n)$ worst case time and inserts a point into S in $O(\alpha \log n)$ amortized time. A related structure supports the deletion of a point from S in $O(\alpha \log n)$ amortized time and reports $\widetilde{W}(S)$ in $O(\alpha \log n \log\log n)$ worst case time. Both data structures use $O(\alpha n)$ space.*

We start by reviewing the solution that leads to Theorem 1.

2 Maintaining an approximation of the width

The definition of width using antipodal pairs given above motivates the following strategy for the dynamic width problem: Maintain the convex hull $CH(S)$ by using the algorithm of [6], and then search for the antipodal pair that yields the minimal distance between two parallel lines enclosing S. Unfortunately, it is not known how to carry out such a search efficiently.

*This work was supported by the ESPRIT Basic Research Actions Program, under contract No. 7141 (project ALCOM II).

[§]Max-Planck-Institut für Informatik, W-6600 Saarbrücken, Germany. email: schwarz@mpi-sb.mpg.de.

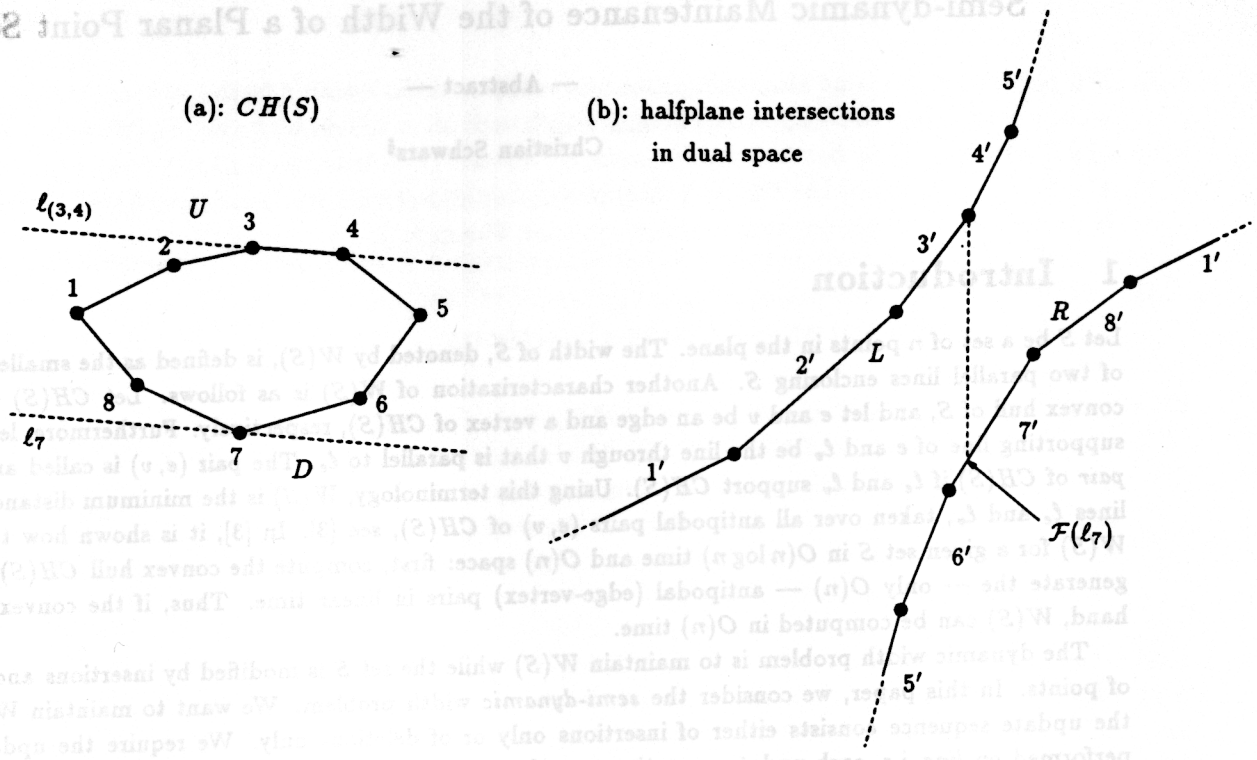

Figure 1: The duality transform $\mathcal{F}$. (a): vertices 1-5 of $CH(S)$ belong to U, and vertices 5-8 belong to D. (b): The dual chains L and R of U resp. D. For each vertex i of $CH(S)$, i' denotes the edge(s) supported by line $\mathcal{F}(i)$. $(7,(3,4))$ is an antipodal pair of $CH(S)$. The distance between the parallel lines ℓ_7 and $\ell_{(3,4)}$ corresponds to the vertical distance of the vertex $3' \cap 4'$ on L to the segment $7'$ on R.

Janardan [4] observed that transforming the problem using *duality* helps: in contrast to the distance of parallel lines of support in primal space, the interpretation of this distance in dual space has a nice monotonicity property, and this makes efficient searching for the minimum possible.

Using the standard duality transform that converts a point $p = (a, b)$ into a line $\mathcal{F}(p) : y = ax + b$ and a non-vertical line $\ell : y = mx + c$ into a point $\mathcal{F}(\ell) = (-m, c)$ to map the points of S, we obtain a set of lines $\mathcal{L}$.

If we orient each non-horizontal line of $\mathcal{L}$ upwards and each horizontal line rightwards, the sets $\mathcal{H}_{left}(\mathcal{L})$ and $\mathcal{H}_{right}(\mathcal{L})$ of left resp. right halfplanes of the lines in $\mathcal{L}$ are well-defined. Let $L := \partial \bigcap_{\ell \in \mathcal{L}} \mathcal{H}_{left}(\mathcal{L})$ be the boundary of the intersection of the halfplanes in $\mathcal{H}_{left}$. Similarly, let $R := \partial \bigcap_{\ell \in \mathcal{L}} \mathcal{H}_{right}(\mathcal{L})$. See Figure 1. The convex chain L corresponds to the upper part U of $CH(S)$ as follows: Every vertex of L is obtained by mapping the supporting line of an edge of U, and every edge (line segment or ray) of L is obtained by mapping the set of supporting lines of a vertex of U.

It follows that for an edge $e \in U$ and a vertex $v \in D$, (e, v) is an antipodal pair of $CH(S)$ if and only if $\mathcal{F}(\ell_v)$ is the point where the vertical line through vertex $\mathcal{F}(\ell_e)$ in L intersects the chain R, where ℓ_e and ℓ_v are the parallel lines through e and v, respectively. See Figure 1 for an example. An analogous statement holds for $e \in D$ and $v \in U$.

With this correspondence, we transform the problem to compute the width into dual space as follows: In dual space, the distance between the vertex $\mathcal{F}(\ell_e)$ and the point $\mathcal{F}(\ell_v)$ will play the role of the distance of the parallel lines of support belonging to (e, v). Instead of finding the antipodal pair (e, v) with minimal distance between ℓ_e and ℓ_v, we want to minimize, over the vertices of both chains L and R, the vertical distance of a vertex to the other chain. For a vertex p on L, let $\rho(p, R)$ denote the vertical distance of p to the chain R. The vertical distance of a point $p \in R$ to L is defined analogously. The distance function ρ has the following monotonicity property which effects an efficient searching algorithm.

Lemma 1 (Janardan [4]) *Let $p_0 \in L$ such that $\rho(p_0, R) = \min_{p \in L} \rho(p, R)$. Index the vertices on L with subsequent integers such that the vertices to the right of p_0 get positive indices and the vertices to the left of*

p_0 get negative indices, i.e. $L = \ldots, p_{-2}, p_{-1}, p_0, p_1, p_2, \ldots$. Then there are indices $k^- \leq 0$ and $k^+ \geq 0$ such that $\rho(p_i, R) = \rho(p_0, R)$ for $k^- \leq i \leq k^+$, $\rho(p_i, R) > \rho(p_{i-1}, R)$ for $i > k^+$ and $\rho(p_i, R) > \rho(p_{i+1}, R)$ for $i < k^-$. An anologous statement holds for the points on R.

Using Lemma 1, p_0 can be found by binary search, as follows. Pick the middle vertex of chain L, compute its distance to chain R and compare it to the distances of its neighbor vertices on L to the chain R. If the distances are equal, we have found p_0. Otherwise half of the chain L can be discarded. The distance of a vertex on L to R is computed by a binary search on R. Applying this algorithm to both L and R, the minimal distance of a vertex on $L \cup R$ to the other chain can be found in time $O(\log^2 n)$.

Maintaining the chains L and R under insertions and deletions of points in S is equivalent to maintaining the convex hull. Using the data structure of Overmars and van Leeuwen [6], this takes $O(\log^2 n)$ time per update.

This method *does not maintain the width of S*, however, because by transforming the problem into dual space, the distance between parallel lines can be distorted considerably, and so the result computed in dual space may be meaningless. The amount of distortion depends on the slope of the lines. By maintaining a family of coordinate systems instead of only one, there will always be a system where the error is kept small. This establishes Theorem 1.

3 Semi-dynamic maintenance of the width approximation

We look for improvements upon the solution of Section 2 in the case where only insertion or only deletions are performed. From now on, we will concentrate on the data structure for one coordinate system. The approximation result then follows from the method of maintaining a family of coordinate systems mentioned above.

Let us examine the running times in Section 2. The update time was $O(\log^2 n)$, determined by the time to maintain the structure of [6] for the convex chains L and R, which are dual to the convex hull of S. In the semi-dynamic case, convex hull algorithms with $O(\log n)$ amortized update time are available: see [2] for deletions and [7] for insertions. The time bound for insertions in [7] is even worst case.

The running time for a query in the structure of Section 2 is $O(\log^2 n)$, because it is a twofold binary search: a vertex p_0, say on L, is searched such that the vertical distance of p_0 to the other chain is minimal. For each vertex p on L that is tested, a binary search on R is performed to find the segment s_p that intersects the vertical line through p. We want to speed-up the query by replacing this inner binary search by a faster method.

To compute the segment s_p, it suffices to compute its incident vertex to the right, denoted *the right neighbor of p in R*. If the right neighbor does not exist, then the segment s_p is the ray at the right end of the chain; we can easily handle this special case.

Note that the vertices are ordered by x-coordinate on both chains L and R. So, in an abstract sense, our problem is as follows: We have two ordered lists of real numbers, and we are given a number in one list. Our goal is to *localize* this number in the other list, where localizing means to find the position (i.e. right neighbor) that the number would have in the other list.

The data structure in Section 2 is organized such that the two chains L and R are kept separately. To perform the above described search for a neighbor of a vertex in the other list, we maintain an additional structure, the *mixed L, R-structure*, where the vertices of both chains are stored together, ordered by x-coordinate.

The problem that we want to solve turns out to be an instance of a well-known data-structuring problem called *union-split-find problem*, see e.g. [5, 1], which is defined as follows.

Let B be a linear list of elements some of which are marked. The marked elements partition the list into intervals. We want to carry out the following operations:

Find(x : element): return the next marked element y to the right of x in B (including x);

Split(x : unmarked element): turn x into a marked element;

Union(x : marked element): turn x into a unmarked element;

Add(y, x : element): insert a new unmarked element y before x into B;

Erase(x : unmarked element): remove x from B;

In [5], a data structure is given that uses space $O(|B|)$ and supports each of the above five operations in time $O(\log\log|B|)$, where the time bound is worst case for Union, Split and Find and amortized for Add and Erase.

Let B be the list containing the vertices of $L \cup R$, ordered by x-coordinate. We keep two instances of B. In each instance, the vertices of one chain (L or R) are *marked*, while the vertices of the other chain are *unmarked*.

Now, finding the right neighbor in R of a vertex v in L can then be done by a query in the structure where the elements of R are marked: find the next marked vertex to the right of v in the list. The running time of this operation is $O(\log\log n)$. It follows that the time to find the vertex $p_0 \in L$ with minimal distance to R is reduced to $O(\log n \log\log n)$.

If a point is inserted into a chain, say L, then a marked element has to be inserted into one of the mixed chains and an unmarked element into the other one. This can be implemented using the operations Add and Split. Similarly, deletions of vertices can be implemented using Union and Erase. Since, additional to the union-split-find structures representing the two instances of the list B, we have to store a dictionary for $L \cup R$, supporting insertions, deletions and lookups of vertices in $O(\log n)$ time each, the time to update the mixed L, R-structure is $O(\log n)$ amortized.

Note that in the mixed L, R-structure, each vertex that, for example, vanishes from one of the chains L and R due to an update that changes the convex hull, has to be deleted explicitly. The number of vertices changing on L and R in dual space is proportional to the number of points that change on the convex hull. That is, if k is the number of points changing on the hull due to an update, the running time for the whole update operation is the time to update the the convex hull plus the time for k updates on the mixed L, R-structure, summing to $O(k \log n)$ amortized.

In the semi-dynamic case we know that, summed over all operations, the number of points deleted from or inserted into the hull is $O(n)$. In the deletions only case, n is the size of the initial set, and in the insertions only case, n is the size of the final set.

Therefore, the total time for all updates is $O(n \log n)$. Combining this with the method of maintaining the data structure for a family of coordinate systems mentioned before, we obtain Theorem 2.

References

[1] K. Mehlhorn, St. Näher. *Dynamic fractional cascading.* Algorithmica **5**, Nr. 2, 1990, pp. 215-241.

[2] J. Hershberger and S. Suri. *Applications of a semi-dynamic convex hull algorithm.* Proc. Second Scand. Workshop on Alg. Theory (SWAT), 1990, pp. 380-392.

[3] M. Houle and G. Toussaint. *Computing the width of a set.* Proc. First Annual ACM Symp. on Computational Geometry, 1985, pp. 1-7.

[4] R. Janardan. *On maintaining the width and diameter of a planar point set online.* International Symp. on Algorithms, 1991, LNCS Vol. 557, pp. 137-149.

[5] K. Mehlhorn. *Datenstrukturen und effiziente Algorithmen 1.* B.G. Teubner, Stuttgart, Germany, 1986

[6] M. Overmars and J. van Leeuwen. *Maintenance of configurations in the plane.* Journal of Computer and System Sciences **23**, 1981, pp. 166-204.

[7] F.P. Preparata. *An optimal real time algorithm for planar convex hulls.* Comm. of the ACM **22**, 1979, pp. 402-405.