A Fully Dynamic Algorithm for Planar Width[*]

Timothy M. Chan[†]

November 27, 2002

Abstract

We show how to maintain the width of a set of n planar points subject to insertions and deletions of points in $O(\sqrt{n}\log^3 n)$ amortized time per update. Previously, no fully dynamic algorithm with a guaranteed sublinear time bound was known.

1 Introduction

The *width* of a set in the two-dimensional plane is the smallest distance between a pair of parallel lines that enclose the set. Clearly, the width of the set coincides with the width of its convex hull, and since the width of a convex polygon can be determined by a simple linear scan, the width of n planar points can be computed in $O(n \log n)$ time [14, 19].

Since the early 1990s, a number of researchers raised the question of how to compute this basic geometric quantity *dynamically* as points are inserted and deleted from the set. Agarwal and Sharir [2] appeared to be the first to study the dynamic width problem (motivated by consideration of a static problem known as "two-line center"). They were able to obtain an efficient data structure for the *decision* problem (deciding whether the width is less than a fixed value) in the *off-line* case only, where the complete sequence of insertions and deletions is given in advance. Each update costs $O(\log^3 n)$ time. Janardan, Rote, Schwarz, and Snoeyink [15, 20] solved the weaker *approximate* problem (finding a value within a factor $1+\delta$ of the width for a fixed $\delta > 0$) in $O(\log^2 n)$ time per update, using known data structures for dynamic convex hulls [18] (see also [5, 6]). Eppstein [11] studied the exact problem in the case of a *random* sequence of insertions and deletions and obtained an $O(\log n)$ expected update time bound.

Recently, Eppstein [12] gave another solution to the exact problem that given any fixed $\varepsilon > 0$, requires $O(n^\varepsilon)$ amortized time per update for the *incremental* case, where only insertions are allowed, as well as the *decremental* case, where only deletions are allowed after preprocessing. In fact, in the general case, this algorithm takes $O(kn^\varepsilon)$ amortized time to process an update, where k is the amount of change to the convex hull. In some situations, k is small on average, but for an arbitrary sequence of insertions and deletions, unfortunately k may be as large as $\Theta(n)$ for every update.

In the general case, a nontrivial worst-case result for the original dynamic width problem has remained open: does there exist a fully dynamic method that is guaranteed to beat (asymptotically)

[*]A preliminary version of this work appeared in *Proc. 16th ACM Sympos. Comput. Geom.*, 2001.

[†]School of Computer Science, University of Waterloo, Waterloo, Ontario N2L 3G1, Canada, `tmchan@uwaterloo.ca`. Work supported in part by an NSERC Research Grant.

the naive linear-time method that simply recomputes the width of the convex hull from scratch? We answer the question in the affirmative in this paper. More precisely, we present an algorithm that maintains the exact width over an arbitrary (on-line) sequence of insertions and deletions, with an amortized update time of $O(\sqrt{n}\log^3 n)$.

Admittedly, the new algorithm is not as fast as Eppstein's algorithms in the various special cases. However, unlike Eppstein's $O(n^\varepsilon)$-time algorithm, which uses Agarwal and Matoušek's advanced dynamic data structure for 3-d polytopes [1], our algorithm is elementary and requires only static data structures for 3-d polytopes (specifically, point location in the medial axis of a convex polygon). Implementation of our algorithm is feasible.

We first attempt to explain why so far there has been no successful attack on the general dynamic width problem. We then proceed to describe our new plan of action.

1.1 Why a sublinear bound was thought to be hard...

For some dynamic problems in geometric optimization, a sublinear algorithm is easily derivable from standard tricks. Take for example the problem of answering *farthest neighbor queries* on a planar point set [19]. We can simply divide the points into $\sqrt{n}$ groups each with $O(\sqrt{n})$ points and construct the farthest-point Voronoi diagram of each group. An update can be handled by rebuilding the Voronoi diagram of the affected group in $O(\sqrt{n}\log n)$ time, and a query can be performed in $O(\sqrt{n}\log n)$ time by querying each group separately (by point location) and combining the answers [4].

By a similar approach, near-$O(\sqrt{n})$-time dynamic algorithms can also be obtained for the *diameter* and the *smallest enclosing circle* of a planar point set [19]: although the task of "combining the answers" is not as self-evident (the technical term for the required property is *decomposability*), both problems actually reduce to the farthest neighbor problem (which is decomposable)—the former by a technique of Eppstein [10], and the latter by parametric or randomized search [9, 16]. (An $O(n^\varepsilon)$ bound is now known for these problems through Agarwal and Matoušek's data structures [1], but historically, this $\sqrt{n}$ approach had indeed been suggested for both diameter [21] and smallest enclosing circle [9].)

Why can't the same approach be applied to maintain the width? The smallest enclosing circle enjoys properties associated with convex programming, but the width is essentially a nonconvex optimization problem. It is not clear how one can divide the points into groups and build a data structure for each group in such a way that the width can be determined by querying these separate structures. It seems that one has to maintain features of the convex hull of the entire point set globally, but then, one faces the problem encountered by Eppstein's recent algorithm [12]: an update may cause many changes to the hull in the worst case.

1.2 Why a sublinear bound isn't that hard after all...

The way we bypass the above-mentioned problem is simple. We know how to deal effectively with the decremental case by Eppstein's result (mainly because an entire deletion sequence causes only $O(n)$ changes to the convex hull, since a vertex can be created and destroyed only once). The idea is to forbid insertions to change the data structure (and in particular, the hull), but simulate the effect of the insertions when we answer a query; when many insertions have accumulated, we rebuild the structure.

Specifically, we will establish the following theorem—which, incidentally, also improves Eppstein's result for the decremental case from $O(n^\varepsilon)$ to $O(\log^3 n)$.

Theorem 1.1 *We can design a data structure for a planar n-point set P that supports deletions of points from P and queries of the following kind: given a point set Q, determine the width of $P \cup Q$. The total preprocessing and deletion time is $O(n \log^3 n)$, and the query time is $O(|Q| \log^3(n + |Q|))$.*

To maintain the width of a point set in the general fully dynamic case, we divide our point set into two subsets P and Q. Deletions of points from P are handled by the above theorem. Deletions in Q are done directly. We insert only to Q. After $\sqrt{n}$ insertions, we reset P to the entire point set and Q to $\emptyset$; the cost of this reconstruction is $O(n \log^3 n)$ time. At any time, we can determine the width of the entire point set in $O(|Q| \log^3 n) = O(\sqrt{n} \log^3 n)$ time. So, the amortized cost for each update is $O(\sqrt{n} \log^3 n)$, as claimed.

2 The Solution

All that remains is for us to prove Theorem 1.1. In other words, we will modify Eppstein's decremental data structure [12] to support our strengthened form of queries. First, we recall some known data structuring tools.

The problem in the following lemma reduces to point location [19] in the *medial axis* [3] (i.e., Voronoi diagram of the edges) of the lower envelope of n lines (a convex polygon): the medial axis of a convex polygon lifts to a three-dimensional halfspace intersection by a standard transformation, and the result follows from known algorithms on 3-d polytope construction and planar point location.

Lemma 2.1 *We can design a data structure for a set of n lines in the plane that supports queries of the following kind: given a point q below all the lines, find the closest line to q. The preprocessing time is $O(n \log n)$, and the query time is $O(\log n)$.* $\square$

Formally, a query problem is *decomposable* if the answer on a set $S_1 \cup S_2$ can be computed from the answer on S_1 and the answer on S_2 in constant time. Bentley and Saxe [4] observed a simple "binary counting" method that in general turns any static data structure for a decomposable problem into an incremental data structure that supports insertions with the cost of an extra logarithmic factor.

Lemma 2.2 *Consider a decomposable query problem. Suppose we can preprocess any set of size n in $nT(n)$ time such that queries can be answered in $Q(n)$ time, where $T(n)$ and $Q(n)$ are monotone increasing functions. Then we can design a data structure for a set of size n that supports insertions in $O(T(n) \log n)$ amortized time and queries in $O(Q(n) \log n)$ time.* $\square$

Another general transformation can turn a static data structure for a decomposable problem into a structure that supports queries on any contiguous subsequence, also at the expense of a logarithmic factor: this well-known method simply builds a complete binary tree and stores a data structure at each node for the leaves underneath.

Lemma 2.3 *Consider a decomposable query problem. Suppose we can preprocess any set of size n in $nT(n)$ time such that queries can be answered in $Q(n)$ time, where $T(n)$ and $Q(n)$ are monotone increasing functions. Then we can preprocess a sequence S of size n in $O(nT(n) \log n)$ time such that a query on any contiguous subsequence of S can be answered in $O(Q(n) \log n)$ time.* $\square$

Finally, the following specific data structure for a list of numbers can be designed by augmenting a standard balanced search tree (technically called a "priority search tree"):

Lemma 2.4 *We can design a data structure for a sequence S of n numbers that supports insertions, deletions, and queries of the following kind: find the minimum of a contiguous subsequence of S. All operations take $O(\log n)$ time.* $\square$

2.1 Notation

We find it more convenient to study the width problem in the *dual* setting [19]. Map a line ℓ with equation $y = mx + b$ to the dual point $\ell^* = (m, b)$. Given n-point set P, let set A consist of the dual of all lines above the (upper) convex hull of P, and let set B consist of the dual of all lines below the (lower) convex hull of P. Here, A and B are disjoint convex polygons (one unbounded from above, the other unbounded from below), as shown in Figure 1. Define the following "dualized distance function" $d : \mathbb{R}^2 \times \mathbb{R}^2 \to \mathbb{R}$:

$$d(\ell_1^*, \ell_2^*) \;=\; \begin{cases} \text{the distance between } \ell_1 \text{ and } \ell_2 & \text{if } \ell_1 \text{ and } \ell_2 \text{ have the same slope} \\ \infty & \text{otherwise.} \end{cases}$$

Note that $d(s,t)$ is finite only if the two points s and t lie on the same vertical line. Given two sets S and T, let $d(S,T) = \min\{d(s,t) \;:\; s \in S,\; t \in T\}$.

The width of P is just $d(A,B)$. It can be verified [14] that the value is attained by $d(s,t)$ either for some vertex s of A and a point t on some edge of B, or for some vertex t of B and a point s on some edge of A.

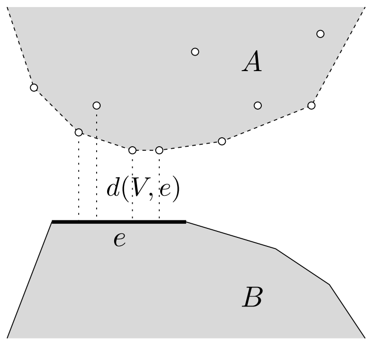


Figure 1: The dual polygons A and B.

2.2 The data structure

Since the vertices/edges of the dual polygons A and B correspond to the edges/vertices of the convex hull of P, we can maintain A and B by a known dynamic convex hull structure; in the decremental case, the required time bound is $O(n \log n)$ [8, 13].

In addition, we need structures for the vertices of A and for the edges of B:

1. We maintain a set $V \subset A$ of $O(n)$ points that contains all vertices of polygon A. (We permit extra non-vertex points in the set, as long as they are inside A.)

Lemma 2.1, when translated to the dual, allows us to preprocess the set V in $O(n \log n)$ time so that given a query line ℓ below A, we can compute $d(V, \ell)$ in $O(\log n)$ time. By ordering points from left to right and applying Lemma 2.3, we can extend the data structure, with an $O(\log n)$-factor increase in the time bounds, so that given any query line segment e below A, we can compute $d(V, e)$ (by querying on those points restricted to the vertical slab spanned by e). Finally, by applying Lemma 2.2 with another logarithmic-factor increase, we have a data structure $\mathcal{S}_A$ that supports insertions to V in $O(\log^3 n)$ amortized time and can find $d(V, e)$ for any segment e below A in $O(\log^3 n)$ time.

2. We maintain the value $d(V, e)$ for each edge e of polygon B.

   By ordering edges from left to right and applying Lemma 2.4, we can extend this list of numbers into a data structure $\mathcal{S}_B$ that supports insertions and deletions to the list in $O(\log n)$ time, and finds $d(V, \gamma)$ for any *chain* γ (union of consecutive edges) of the polygon B in $O(\log n)$ time.[1]

We similarly keep a "reverse" data structure, with the roles of the polygons A and B interchanged.

Preprocessing takes $O(n \log^3 n)$ time: initially, we determine the vertices V of A, and for each edge e of B, compute $d(V, e)$ by querying structure $\mathcal{S}_A$ (or directly by performing a linear scan).

Whenever a vertex v is created in polygon A, we insert v to V and update structure $\mathcal{S}_A$. We locate the edge e of polygon B that intersects the vertical line at v by binary search, and update the value $d(V, e)$ (if $d(v, e)$ is smaller) in structure $\mathcal{S}_B$.

Whenever a vertex v is destroyed in A, we have to retain v in V, as $\mathcal{S}_A$ does not support deletions (otherwise, we would need Agarwal and Matoušek's complicated dynamic data structures [1]). Fortunately, v always stays inside A, since A can only expand during a deletion sequence.

Whenever an edge e is created in polygon B, we query structure $\mathcal{S}_A$ to determine $d(V, e)$ and insert this value to structure $\mathcal{S}_B$.

Whenever an edge e is destroyed in B, we simply delete the value $d(V, e)$ from structure $\mathcal{S}_B$.

The reverse data structure can similarly be maintained.

Thus, our data structure can be updated in $O(\log^3 n)$ time per change in the polygons A and B. As the total change over a deletion sequence is $O(n)$, the total update time is $O(n \log^3 n)$.

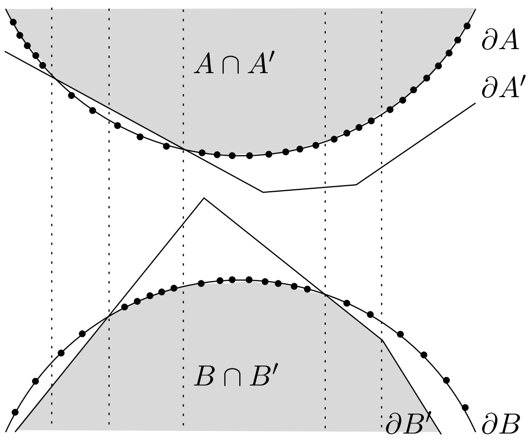
2.3 The query algorithm

Given point set Q, let polygon A' consist of the dual of all lines above the convex hull of Q, and polygon B' consist of the dual of all lines below the convex hull of Q. We can compute these dual polygons by a convex hull algorithm in $O(|Q| \log |Q|)$ time.

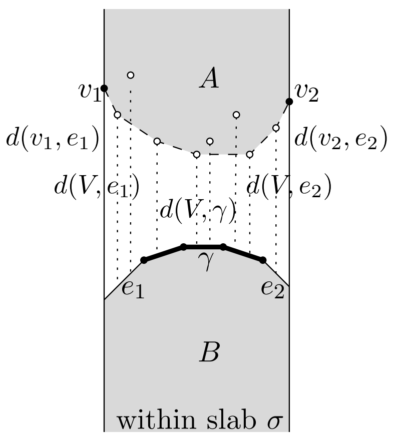
The width of $P \cup Q$ is just $d(A \cap A', B \cap B')$. Since we can intersect a convex polygon with a line in logarithmic time by binary search [19], we can intersect ∂A with $\partial A'$ and ∂B with $\partial B'$ in $O(|Q| \log n)$ time. Draw vertical lines at these intersection points. In addition, draw vertical lines at those vertices of A' and B' that are inside A and B, respectively.

As a result, these $O(|Q|)$ vertical lines divide the plane into $O(|Q|)$ slabs (computable by a linear scan), and it suffices to find $d(A \cap A', B \cap B' \cap \sigma)$ for each of these slabs σ and take the minimum. Within a slab σ, $\partial(A \cap A')$ either coincides with ∂A or is a single line segment f from $\partial A'$, as illustrated by Figure 2(a). Similarly, $\partial(B \cap B')$ either coincides with ∂B or is a single segment e

[1]For those familiar with Eppstein's paper [12], this extension is essentially the only addition to his data structure (with the exception that his data structure was described in the primal setting).



(a)



(b)

Figure 2: (a) The polygons $A \cap A'$ and $B \cap B'$. (b) Computing $d(A, B \cap \sigma)$.

from $\partial B'$. (We can tell which case by a local test.) Therefore, $d(A \cap A', B \cap B' \cap \sigma)$ can be one of four possible quantities:

1. $d(f, e)$.

2. $d(A, e)$: This value is attained by a vertex of $A \cap \sigma$. Let v_1 and v_2 be the intersection points of ∂A with $\partial \sigma$ (computable by binary search). Then $d(A, e) = \min\{d(V, e), d(v_1, e), d(v_2, e)\}$ can be computed by one query to structure $\mathcal{S}_A$.

3. $d(f, B)$: We can similarly compute this value by querying the reverse structure.

4. $d(A, B \cap \sigma)$: This value is attained by a vertex of $A \cap \sigma$ or by a vertex of $B \cap \sigma$. Say the former case is true. Let v_1 and v_2 be as before. Write $\partial B \cap \sigma$ as a union of a chain γ of polygon B with two segments e_1 and e_2 (computable by binary search), as shown in Figure 2(b). Then $d(A, B \cap \sigma) = \min\{d(V, \gamma), d(V, e_1), d(V, e_2), d(v_1, e_1), d(v_2, e_2)\}$ can be computed by one query to structure $\mathcal{S}_B$ and two queries to structure $\mathcal{S}_A$. We can similarly handle the latter case by querying the reverse structures and take the smaller of the two candidate values.

Thus, $d(A \cap A', B \cap B' \cap \sigma)$ can be found in $O(\log^3 n)$ time, so $d(A \cap A', B \cap B')$ can be found in $O(|Q| \log^3 n)$ time. Theorem 1.1 is proved. $\square$

3 Conclusion

We have extended Eppstein's decremental algorithm [12] to obtain the first sublinear time bound for the general dynamic width problem in the plane. Only relatively simple techniques are required.

A slight improvement on the time bound is possible by experimenting with non-binary variants of Lemmas 2.2 and 2.3, but the savings appear to be insignificant (a $\log^2 \log n$ factor only). The main open problem is whether the width can be maintained in $o(\sqrt{n})$ amortized time.

In the off-line case where we know when each point is to be deleted from the set, a static version of Theorem 1.1 is sufficient and our update time bound reduces by at least a logarithmic factor.

However, we believe that there is a more efficient solution in this case (as demonstrated by Agarwal and Sharir [2] for the decision problem).

In the incremental case, a static version of Theorem 1.1 is again sufficient, and although Eppstein's bound is better, our near-$O(\sqrt{n})$ update time bound can be made worst-case instead of amortized, by a standard trick of spreading the rebuilding work over time [17]. It is interesting to see whether a faster "real-time" algorithm for width is possible.

In the decremental case, can our $O(\log^3 n)$ amortized time bound be reduced to near-logarithmic?

The planar width problem lifts to a nonconvex optimization problem over a 3-d halfspace intersection. Both of Agarwal and Matoušek's 3-d data structures [1] did not actually maintain the polytope explicitly but exploited decomposability. Is there a sublinear dynamic algorithm for 3-d halfspace intersection that can maintain the minimum of an arbitrary objective function, or other global parameters like volume or the number of vertices? (Overmars and van Leeuwen [18] answered the question in the affirmative for the planar case, but we suspect that the answer could be negative in the 3-d fully dynamic setting.)

Other challenging open problems include the maintenance of the smallest area/perimeter rectangle (or the smallest square) enclosing a planar point set, and the width of a point set in 3-d. Some progress concerning the smallest enclosing rectangle and other geometric optimization problems in the incremental and off-line cases has been reported in a follow-up paper [7].

References

[1] P. K. Agarwal and J. Matoušek. Dynamic half-space range reporting and its applications. *Algorithmica*, 13:325–345, 1995.

[2] P. K. Agarwal and M. Sharir. Off-line dynamic maintenance of the width of a planar point set. *Comput. Geom. Theory Appl.*, 1:65–78, 1991.

[3] A. Aggarwal, L. J. Guibas, J. Saxe, and P. W. Shor. A linear-time algorithm for computing the Voronoi diagram of a convex polygon. *Discrete Comput. Geom.*, 4:591–604, 1989.

[4] J. Bentley and J. Saxe. Decomposable searching problems I: static-to-dynamic transformation. *J. Algorithms*, 1:301–358, 1980.

[5] G. S. Brodal and R. Jacob. Dynamic planar convex hull. In *Proc. 43rd IEEE Sympos. Found. Comput. Sci.*, pages 617–626, 2002.

[6] T. M. Chan. Dynamic planar convex hull operations in near-logarithmic amortized time. *J. ACM*, 48:1–12, 2001.

[7] T. M. Chan. Semi-online maintenance of geometric optima and measures. In *Proc. 13th ACM-SIAM Sympos. Discrete Algorithms*, pages 474–483, 2002.

[8] B. Chazelle. On the convex layers of a planar set. *IEEE Trans. Inform. Theory*, IT-31:509–517, 1985.

[9] D. Eppstein. Dynamic three-dimensional linear programming. *ORSA J. Comput.*, 4:360–368, 1992.

[10] D. Eppstein. Dynamic Euclidean minimum spanning trees and extrema of binary functions. *Discrete Comput. Geom.*, 13:111–122, 1995.

[11] D. Eppstein. Average case analysis of dynamic geometric optimization. *Comput. Geom. Theory Appl.*, 6:45–68, 1996.

[12] D. Eppstein. Incremental and decremental maintenance of planar width. *J. Algorithms*, 37:570–577, 2000.

[13] J. Hershberger and S. Suri. Applications of a semi-dynamic convex hull algorithm. *BIT*, 32:249–267, 1992.

[14] M. E. Houle and G. T. Toussaint. Computing the width of a set. *IEEE Trans. Pattern Anal. Mach. Intell.*, PAMI-10:761–765, 1988.

[15] R. Janardan. On maintaining the width and diameter of a planar point-set online. *Int. J. Comput. Geom. Appl.*, 3:331–344, 1993.

[16] J. Matoušek. Linear optimization queries. *J. Algorithms*, 14:432–448, 1993.

[17] K. Mehlhorn. *Data Structures and Algorithms 3: Multi-Dimensional Searching and Computational Geometry.* Springer-Verlag, Heidelberg, 1984.

[18] M. H. Overmars and J. van Leeuwen. Maintenance of configurations in the plane. *J. Comput. Sys. Sci.*, 23:166–204, 1981.

[19] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction.* Springer-Verlag, New York, 1985.

[20] G. Rote, C. Schwarz, and J. Snoeyink. Maintaining the approximate width of a set of points in the plane. In *Proc. 5th Canad. Conf. Comput. Geom.*, pages 258–263, 1993.

[21] K. J. Supowit. New techniques for some dynamic closest-point and farthest-point problems. In *Proc. 1st ACM-SIAM Sympos. Discrete Algorithms*, pages 84–90, 1990.