Maintaining the Approximate Width of a Set of Points in the Plane*

(extended abstract)

Günter Rote† Christian Schwarz‡ Jack Snoeyink§

Abstract

The width of a set of n points in the plane is the smallest distance between two parallel lines that enclose the set. We maintain the set of points under insertions and deletions of points and we are able to report an approximation of the width of this dynamic point set. Our data structure takes linear space and allows for reporting the approximation with relative accuracy ϵ in $O(\sqrt{1/\epsilon}\log n)$ time; and the update time is $O(\log^2 n)$. The method uses the tentative prune-and-search strategy of Kirkpatrick and Snoeyink.

1 Introduction

Let P be a set of n points in the plane. The *width function* $w(\theta)$ is the smallest distance between two parallel lines of direction θ which enclose P, where the angle θ is measured counterclockwise from the x-axis, see figure 1. The minimum w^* of the width function is *the width* of the point set.

If the convex hull of P is available, one can construct in linear time the width function $h(\theta)$ for the whole range of θ and thus the width w^* [2, 3]. The *dynamic* width problem is to maintain w^* while the point set P is modified by insertions and deletions of points. The goal is to find algorithms that beat the trivial $O(n)$ bound obtained by updating the convex hull with any efficient dynamic convex hull algorithm and then recomputing the width. We do not know of any algorithm achieving this goal, but Janardan [4] gave an algorithm which maintains an *approximation* of the width. The following table summarizes his results and our improvements for maintaining an approximation with relative error ϵ.

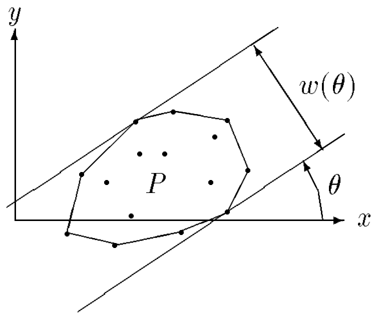

Figure 1: The function $w(\theta)$

	update time	query time	space
Janardan [4]	$\sqrt{1/\epsilon}\cdot\log^2 n$	$\sqrt{1/\epsilon}\cdot\log^2 n$	$\sqrt{1/\epsilon}\cdot n$
our results:			
fully dynamic	$\log^2 n$	$\sqrt{1/\epsilon}\cdot\log n$	n
insertions only	$\log n$	$\sqrt{1/\epsilon}\cdot\log n$	n
deletions only	$\log n$ (amortized)	$\sqrt{1/\epsilon}\cdot\log n$	n

*The work of Christian Schwarz was supported by the ESPRIT Basic Research Actions Program, under contract No. 7141 (project ALCOM II). The work of Jack Snoeyink was supported in part by an NSERC Research Grant.

†Institut für Mathematik, Technische Universität Graz, Kopernikusgasse 24, A-8010 Graz, Austria; electronic mail: rote@ftug.dnet.tu-graz.ac.at

‡Max-Planck-Institut für Informatik, W-6600 Saarbrücken, Germany; electronic mail: schwarz@mpi-sb.mpg.de; from September 1993: schwarz@icsi.berkeley.edu

§Department of Computer Science, University of British Columbia, 6356 Agricultural Rd., Vancouver, BC, V6T 1Z2, Canada; electronic mail: snoeyink@cs.ubc.ca

2 The overall strategy

The first approach is to approximate the width by sampling the width function $w(\theta)$ in sufficiently many equally spaced directions $\theta_i \in [-\pi/2, \pi/2]$, and use the minimum of those values. This, however, does not work for very long and narrow point sets like in figure 2: We may miss the correct direction and thus get an arbitrarily large relative error. (This is in contrast to the maximum of the width function, the diameter, see [4].) Therefore we improve the estimate by searching for a *local minimum* of $w(\theta)$ in the vicinity of each direction θ_i. It turns out that this makes the error much smaller.

However, in the dynamic setting, we do not know how to find a true local minimum in $O(\log n)$ time. (Schwarz [11] proposed an algorithm using a union-split-find data structure, see e.g. [7, 8]; it gives a time bound of $O(\log n \log\log n)$ for finding a local minimum and for updates.) Nevertheless, we can apply the *tentative prune-and-search* strategy which was recently introduced by Kirkpatrick and Snoeyink [5]. This either leads in $O(\log n)$ steps to a local minimum or gets stuck in a situation where it cannot proceed. Fortunately, this situation is just as good: We can still estimate the width with sufficient precision.

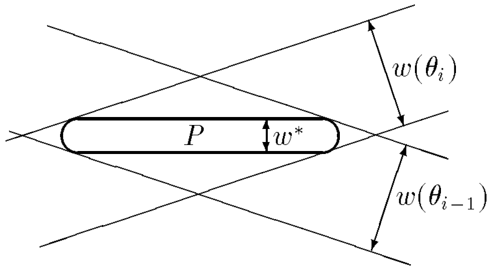

Figure 2: A bad case

3 The algorithm

As the underlying data structure, we dynamically maintain the convex hull of the point set P. Using the data structure of Overmars and van Leeuwen [9], this takes $O(\log^2 n)$ time per update, and it provides the convex hull in the form of a balanced binary tree of the edges in sorted order: Each internal node corresponds to a subchain of the convex hull. It provides constant-time access to one point on that chain which splits the chain into the two chains corresponding to the child nodes. Given this data structure, we can compute the width $h(\theta)$ in any particular direction θ in $O(\log n)$ time.

We want to find a number w whose relative difference from the width w^* is at most a given bound ϵ. We set

$$k := \lfloor\sqrt{1/\epsilon}\rfloor, \quad \alpha := \pi/k,$$

and

$$\theta_i := -\pi/2 + i\alpha, \text{ for } i = 0, \ldots, k.$$

We look at each interval $[\theta_{i-1}, \theta_i]$ in succession and try to find a local minimum in that interval.

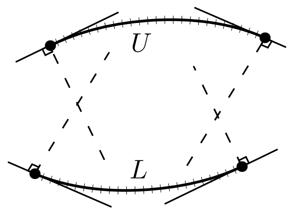

Figure 3: Extreme tangents

Let us consider a fixed such interval. The part of the convex hull that has tangents in this range of directions consists of two convex polygonal chains, an *upward chain* U and a *downward chain* L, see figure 3. If the width w^* occurs as the minimum distance between parallel tangents to U and L, the number w which we will compute w will satisfy

$$\cos\alpha \le w/w^* \le 1/\cos\alpha.$$

We say that a segment $\overline{ab}$ is a *left fence* (*right fence*) if all tangents to U at a and to L at b form non-obtuse angles to the left (right) of $\overline{ab}$, see the right part of figure 6. Segment $\overline{ab}$ is *mixed* if obtuse angles can be formed on opposite sides of $\overline{ab}$ by tangents at a and b, see the left part of figure 6. A triangle $\triangle bac$ with $a \in U$ and $b, c \in L$ is a *fence triangle* if $\overline{ab}$ is a left fence and $\overline{ac}$ is a right fence. Figure 4 shows a fence triangle for chains U and L. It also shows our schematic representation of fences—we mark the obtuse angles and use arrows to indicate where tangents intersect.

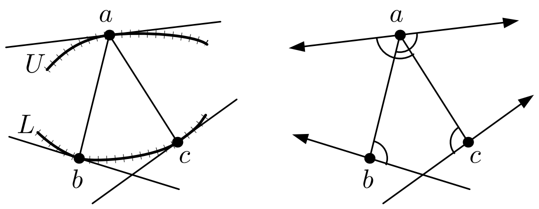

Figure 4: A fence triangle

A segment $\overline{ab}$ that attains a local or global minimum of $w(\theta)$ is normal to tangents at a and b; it can be seen as a degenerate fence triangle $\triangle bab$. We will see that any fence triangle (degenerate or not) gives a good approximation w for width. We begin with a simple lemma about fences.

Lemma 1 *No segment that is a local minimum for width has both endpoints left of a left fence or right of a right fence.*

Proof: The tangents for a left fence $\overline{ab}$ intersect and form a triangle to the left of $\overline{ab}$. Tangents to a' left of a and b' left of b intersect inside this triangle, therefore $\overline{a'b'}$ cannot be normal to both of these tangents. ∎

We can return the length of the longer segment of a fence triangle as the approximate width w.

Theorem 2 *Suppose that $a \in U$ and $b, c \in L$ form a fence triangle $\triangle bac$. Let $w = \max(ab, ac)$. Then $w \geq w^* \cos\alpha$. If, in addition, the minimum width w^* is attained at some chord $\overline{a^*b^*}$ between chain U and chain L, we have $w \leq w^*/\cos\alpha$.*

Proof: Let θ be the direction of the tangent t_a at a, and let d denote the intersection of the tangents at b and c, see figure 5a.

Since the three tangents at a, b, and c delimit P, we may bound $w(\theta)$ by the distance of d to t_a. Suppose w. l. o. g. that a lies to the right of the orthogonal projection d' of d onto t_a, and let $\alpha' \leq \alpha$ denote the angle between t_a and the tangent t_b at b. Now ab is at least the distance from d' to t_b, which equals $dd' \cdot \cos\alpha'$. Thus we have

$$\begin{aligned} w \geq ab \quad &\geq \quad dd' \cdot \cos\alpha' \geq dd' \cdot \cos\alpha \\ &\geq \quad w(\theta) \cdot \cos\alpha \geq w^* \cdot \cos\alpha. \end{aligned}$$

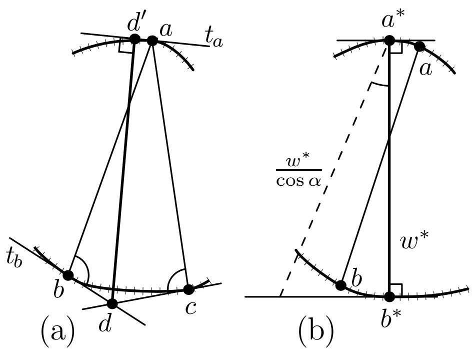

Figure 5: Under- and overestimates

Now assume that $a^* \in U$ and $b^* \in L$ and a^*b^* attains the minimum width w^*. Figure 5b shows by how much w may overestimate w^*. The normals to $\overline{ab}$ and $\overline{a^*b^*}$ both fall in the angle range $[\theta_{i-1}, \theta_i]$, so their angles differ by at most α. Because U and L lie between the tangents at a^* and b^*, we get $w^* \geq w\cos\alpha$. ∎

We use the tentative prune-and-search technique of Kirkpatrick and Snoeyink [5, 6] to find a fence triangle in logarithmic time.

Theorem 3 *Let U be an upper convex polygonal chain with tangent directions in an interval $[\theta_{i-1}, \theta_i]$ and let L be a lower convex polygonal chain with tangent directions in the same interval, with a total of n vertices. If U and L are given in the form of balanced binary search trees we can compute a fence triangle in $O(\log n)$ time, or we can conclude that no local minimum of the width function exists.*

Proof: Tentative prune and search has two modes of operation. In *normal* mode, we look at the vertices $a \in U$ and $b \in L$ with median indices. We either discard half of one chain or else we tentatively discard half of each chain, making at most one mistake, and go to *tentative* mode. In tentative mode we refine the chains U and L alternately and either discard (tentatively or permanently) half of the chain being refined or certify all tentative discards on one chain, revoke those on the other, and return to normal mode. This mode of operation ensures termination after at most $O(\log n)$ steps [6].

The above description is somewhat idealized because it assumes that we may always split the current subchains of U and L evenly. In our case, however, the splitpoint is provided by the dynamic convex hull data structure. It does not necessarily give an even split, but it is ensured that the depth of the underlying tree is $O(\log n)$, which suffices for the $O(\log n)$ time bound.

In normal mode, segment $\overline{ab}$ is mixed or is a fence. These cases are shown schematically in figure 6. In the former case, we discard the half of the chain that is incident to the obtuse angle nearest the intersection of the tangents at a and b—the left portion of L in the figure. The justification is that any point $a' \in U$ with tangent parallel to a tangent at b must be right of a. Thus, there is a local minimum or maximum that is right of b and left of a'.

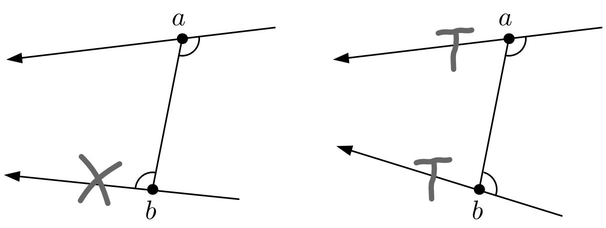

Figure 6: Cases in normal mode

In the case that $\overline{ab}$ is a fence, we tentatively discard the portions outside the fence $\overline{ab}$ according to lemma 1 and enter tentative mode.

In tentative mode, let us assume that $\overline{ab}$ is a left fence and everything to the left of a and b has been tentatively discarded; the subchains that are currently under consideration are to the right of a and b. Let us further assume that we are going to refine the lower subchain, looking at some chord $\overline{ab'}$. If $\overline{ab'}$ is another left fence, we extend the tentative discard to b', see figure 7. If $\overline{ab'}$ is a right fence, we have found a fence triangle: We stop and report the candidate approximation of the width according to theorem 2. Otherwise we have one of four mixed cases, depending on where the obtuse angles and intersections formed by tangents to a and b' lie. In one case we discard the portion right of b' and remain in tentative mode. In all other cases we can certify that some tentative discard is correct and return to normal mode. (The most complicated justification is for the discard right of a in the last case: we discover that the tentative discard left of a was a mistake, so lemma 1 certifies the tentative discard left of b.)

The algorithm stops when it has found a fence triangle or when the two chains are exhausted. ∎

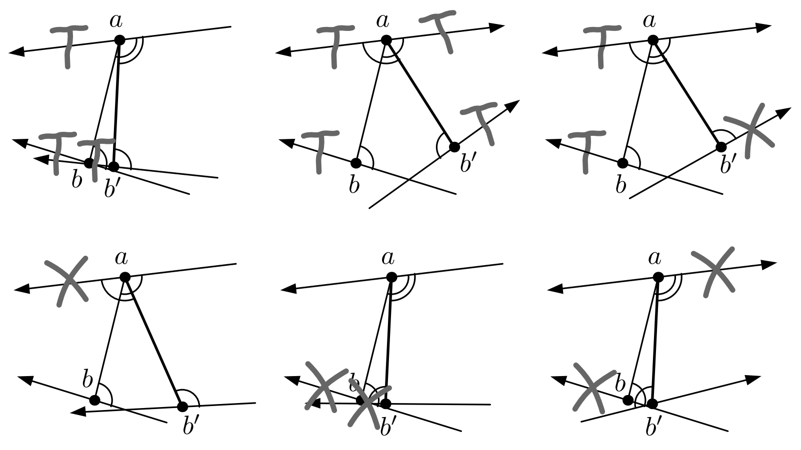

Figure 7: Cases in tentative mode

It is now clear how to proceed: For every interval $[\theta_{i-1}, \theta_i]$, $i = 1, \ldots, k$, we start the tentative prune-and-search for the corresponding two chains U and L. Each time we terminate with a fence triangle we get an estimate w_i of the width. We take w as the minimum of those numbers w_i, and by theorem 2 we get

$$\frac{w}{w^*} \in [\cos\alpha, 1/\cos\alpha] \subseteq 1 \pm O(\alpha^2) \subseteq 1 \pm O(1/k^2) \subseteq 1 \pm O(\epsilon),$$

by our choices of k and α. Thus we can conclude:

Theorem 4 *Given a representation of a convex polygon as a balanced binary tree, we can compute an approximation w of its width w^* with a relative accuracy of ϵ in $O(\sqrt{1/\epsilon}\log n)$ time:*

$$\left|\frac{w^* - w}{w^*}\right| \leq \epsilon.$$

Proof: The proof follows from the above discussion. ∎

Theorem 5 *We can maintain an approximation of the width of a dynamic set of points with linear space and update time of $O(\log^2 n)$ per insertion or deletion of a point and $O(\sqrt{1/\epsilon}\log n)$ query time. In the semi-dynamic case, where only insertions or only deletions occur, the update time is reduced to $O(\log n)$. In the case of deletions, this bound is only amortized over the sequence of operations.*

Proof: The time bound for updates follows from the corresponding bounds for maintaining the convex hull, see [9] for the fully dynamic case, [10] for insertions, and [1] for deletions. The bound for the query time follows from the previous theorem. ∎

Note that the only thing that we actually maintain is the convex hull, and this is independent of ϵ. We may thus specify ϵ only at the time of the query for the width.

4 Conclusion

The major open question is maintaining the true width in sub-linear time, even under insertions only. Another problem would be to determine more precisely those angular ranges where the minimum width can occur, and perform a local search more adaptively, only in the vicinity of the promising directions. We have not been able to restrict the local searches to a subset of the directions without a degradation in the quality of the approximation.

We remark that our time bounds can also be achieved by the following improvements of Janardan's method [4]: Janardan looks at $O(\sqrt{1/\epsilon})$ equally spaced directions; for each direction he constructs and maintains the dual arrangement of lines where that given direction is considered vertical. We can reduce the space and update requirements by working with a single copy of the convex hull instead of a separate copy for each of direction. For each direction, Janardan finds the pair of enclosing parallel lines whose distance *in that direction* is minimal. Here we can cancel a factor of $\log n$ by using the prune-and-search technique instead of nested binary search.

References

[1] J. Hershberger and S. Suri. Applications of a semi-dynamic convex hull algorithm. In: *Proc. 2nd Scandinavian Workshop on Algorithm Theory* (SWAT 90), 1990, Lecture Notes in Computer Science **447**, Springer-Verlag, pp. 380–392.

[2] M. E. Houle and G. T. Toussaint. Computing the width of a set. In: *Proc. First Annual Symposium on Computational Geometry*, 1985, pp. 1–7.

[3] M. E. Houle and G. T. Toussaint. Computing the width of a set. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **PAMI-10** (1988), 761–765.

[4] R. Janardan. On maintaining the width and diameter of a planar point set online. In: *Proc. 2nd International Symposium on Algorithms* (ISA'91), 1991, Lecture Notes in Computer Science **557**, Springer-Verlag, pp. 137–149. To appear in *International Journal of Computational Geometry & Applications*.

[5] D. Kirkpatrick and J. Snoeyink. Tentative prune-and-search for computing Voronoi vertices. In: *Proc. 9th Annual Symposium on Computational Geometry*, 1993, pp. 133–142.

[6] D. Kirkpatrick and J. Snoeyink. Computing constrained shortest segments: Butterfly wingspans in logarithmic time. In: *Proc. 5th Canadian Conference on Computational Geometry*, 1993, these proceedings.

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

[8] K. Mehlhorn, S. Näher. Dynamic fractional cascading. *Algorithmica* **5** (1990), 215–241.

[9] M. Overmars and J. van Leeuwen. Maintenance of configurations in the plane. *Journal of Computer and System Sciences* **23** (1981), 166–204.

[10] F. P. Preparata. An optimal real time algorithm for planar convex hulls. *Communications of the ACM* **22** (1979), 402–405.

[11] C. Schwarz. *Semi-dynamic maintenance of the width of a planar point set.* Technical Report MPI-I-92-153, Max-Planck-Institut für Informatik, Saarbrücken, Germany, December 1992.