Adaptive Planar Convex Hull Algorithm for a Set of Convex Hulls

Jérémy Barbay and Eric Y. Chen

Cheriton School of Computer Science
University of Waterloo, Canada.
{jbarbay,y28chen}@uwaterloo.ca

Technical Report CS-2007-20

Abstract. An adaptive algorithm is one which performance can be expressed more precisely than as a mere function of the size of the input: output-sensitive algorithms are a special case of adaptive algorithms. We consider the computation of the convex hull of a set of convex hulls, for instance in the case where the set of points has been composed from simpler objects from a library, for each of which the convex hull has been precomputed. We show that in this context an adaptive algorithm performs better if it takes advantage of other features than the size of the output.

Keywords: Computational geometry; Convex hulls; Adaptive Algorithm

1 Introduction

Adaptive algorithms are algorithms that take advantage of "easy" instances of the problem at hand, i.e. their complexity depends on some measure of difficulty, for example a function of the size of the instance and of other parameters.

For example, *sorting* an array A of numbers is a basic problem, where some instances are easier than others to sort (e.g. a sorted array, which can be checked/sorted in linear time). Considering the disorder in an array as a measure of the difficulty of sorting this array [3, 11], one can yield a finer analysis of the complexity of the problem. There are many ways to measure this disorder: one can consider the number of exchanges required to sort an array; the number of adjacent exchanges required; the number of pairs (i, j) in the wrong order, but there are many others [14]. For each disorder measure, the logarithm of the number of instances with a fixed size and a fixed number of disorder pairs forms a natural lower bound to the worst case complexity of any sorting algorithm in the comparison model, as a correct algorithm must at least be able to distinguish all instances. As a consequence, there could be as many optimal algorithms as the difficulty measures. Indeed, one can reduce difficulty measures between themselves, which yields a hierarchy of disorder measures [9].

A particular case of this approach has been applied to some fundamental problems in computational geometry, such as convex hull problems, intersection-reporting of line segments [7]. where algorithms said to be "output-sensitive" are analyzed under the assumption that the size of the *output* measures the difficulty of each instance. For example, Kirkpatrick and Seidel [10] proposed an algorithm for computing the convex hull that has running time $O(n \log h)$, where n is the number of input vertices, and h is the number of output vertices in the resulting convex hull. It was later simplified by Chan [4]. As previously known algorithms guarantee only a running time of $O(n \log n)$ in the worst case, clearly, the adaptive algorithm performs better when the size of the convex hull size is small (e.g. a triangle).

Can we find an even finer measurement and outperform output-sensitive algorithms? In this paper, we focus on improving the computation of the convex hull in the case where the set of points is composed of a few simpler objects (i.e. $k < n$) for which the convex hull has been precomputed (e.g. an application combining pieces from a library of mechanical pieces to obtain more complex objects). Let $I = \{H_1, \ldots, H_k\}$ be a set of k convex hulls in the Euclidian plane, respectively composed of $n_1, \ldots, n_k$ points each, all distinct and non-collinear. We compute the minimal convex hull containing every point of I.

We describe adaptive algorithms which take optimally advantage of the relative positions of the objects composing I, output the shortest possible description of the convex hull from the input, and write the result in a write-only stream, using working storage of $\mathcal{O}(k)$ words, independent of the input size n.

The asymptotic performance of our adaptive algorithms is never worse than the performance of traditional or output-sensitive algorithms: their performance is $\mathcal{O}(n \log h)$ in the worst case over instances of input size n and output size h, and hence $\mathcal{O}(n \log n)$ in the worst case over instances of input size n. In particular, our adaptive algorithms perform better than any output-sensitive algorithm on instances of large output size which solution can be shortly described and certified.

In this paper, we describe an algorithm, the adaptive analysis of its complexity and the matching adaptive computational lower bound, all in the planar comparison model where only point-to-point and point-to-line comparisons are allowed. Our algorithm (Section 2) computes the convex hull of an instance I formed of k convex upper hulls. We here only describe the algorithm to compute the upper hull of a set of upper hulls. The lower hull can be computed symmetrically. To compute the convex hull, we can compute the upper hull and lower hull separately, and then merge them together. The analysis of this algorithm introduces the notion of the *certificate* of an instance, which is essential in both the proof of correctness and the complexity analysis of our algorithms: this notion is similar in essence to the one used of the combination of sorted sets in one dimension [1, 8]. The adaptive computational lower bound (Section 3) shows that the worst case complexity of our algorithms over instances of fixed input size n, k and difficulty δ is optimal.

2 Convex Hull Problem

2.1 Notion of Certificate

Before we describe our algorithms, we introduce the concept of the *certificate* for an instance.

Given two upper hulls, in some circumstance, the merged hull is easier to compute. For example, two horizontally disjoint upper hulls can be merged in $\mathcal{O}(\log(n_1) + \log(n_2))$ time [10], where n_1 and n_2 are the size of the two upper hulls respectively. There are even easier cases: if all the points of the first upper hull are contained in the truncated cone formed by the first and last edges of the second upper hull, then no point from the first upper hull will contribute to the convex union. In such a case, the upper hull can be computed in constant time.

By one *assertion*, we verify whether a point is above a line. Given two upper hulls, the merged hull can be certified by a set of such assertions. We call one set of such assertions a *certificate*.

Definition 1. *Given k sorted upper hulls $H_1, \ldots, H_k$ represented by arrays of points $A_1, \ldots, A_k$ of respective sizes $n_1, \ldots, n_k$ and their convex hull H, expressed as several intervals on the arrays $A_1, \ldots, A_k$. A* certificate *of H is a set of assertions of the type "$A_i[p]$ is above[1] the line $(A_j[q], A_j[k])$",*

[1] In the clockwise orientation.

such that the convex hull of any instance satisfying those assertions is given by the description of H. The size *of a certificate is the number of assertions contained in it.*

we also note that such a certificate not only justify the presence of each point in the output, but also justify the exclusion of the other points.

It not hard to see that all those easier cases showed at the beginning of this section have a certificate with small size, which indicates the size of the certificate is a good measurement of the difficulty of an instance. Alternatively, the so-called *convolution* can also be used as a measure of the difficulty.

Definition 2. *A* slicing *of the instance is a partition $(I_j)_{j\in[\delta]}$ of the domain of the x-coordinates such that, for each interval I_j of the partition, the part of the upper hull intersecting I_j can be certified in $k-1$ comparisons. The minimal size of a slicing of the instance measures the difficulty to certify this instance: we call it the* convolution *of the instance.*

2.2 Basic Operations

Computing the 2D convex hulls is considered as a natural extension of sorting problems. To compute the merged hull (/em convex union) of upper hulls is at once similar and distinct from computing the union and intersection of sorted arrays [8]:

- Whereas when computing the union of k sorted arrays the union of each pair of arrays matter, when computing the convex union of sorted upper hulls the interleaving from the lowest upper hulls can be ignored.
- Whereas when computing the intersection of k sorted upper hulls, only two arrays need to be considered if their intersection is empty, all the upper hulls need to be considered at least once to compute their convex union.

The analogy enables us to adapt some known techniques from 1D adaptive algorithms and also enlightens us to design some new basic operations for our new adaptive algorithms. We present them by increasing order of importance.

Eliminator Line The main similarity between the convex union of upper hulls and the intersection of sorted arrays is that in some cases a large section of one of the k components of the instance (sorted arrays for the intersection problem, sorted upper hulls here) can be eliminated by a simple operation.

Observation 1 *Given a line l and an upper hull U, if the point $U[p]$ is below l and the slope of $U[p]U[p+1]$ is smaller than l, then all points right to $U[p]$ is below l; if the point $U[p]$ is below l and the slope of $U[p-1]U[p]$ is greater than l, then all points left to $U[p]$ is below l.*

Doubling Search Another similarity between algorithms computing the union of sorted arrays and the various "wrapping" algorithms computing the convex union of upper hulls, is that they both compute the result from "left to right". In this context, it is not necessary to perform a binary search for the insertion rank of a value (resp. for the tangent of a point) every time on the whole array (resp. on the whole sorted upper hull) at each search: a doubling search [2] algorithm permits to amortize the sum of each search over the whole structure:

Lemma 1. *There is a deterministic algorithm in the comparison model which decides if there is an edge $[c_p, c_{p+1}]$ in an ordered upper hull $c_1, \ldots, c_n$ which intersects a line of (a, b), and finds it if it exists, in $\mathcal{O}(\log p)$ comparisons.*

Proof. This is simply a doubling search [2] on the x-coordinate of the points of the upper hull, performed in $2\log(p)$ operations. □

Lemma 2. *The tangent (x, c_p) of a point x with an ordered upper hull $c_1, \ldots, c_n$ can be found in $\mathcal{O}(\log p)$ comparisons.*

Proof. The tangent can be found using a doubling search algorithm [2] on the angle between (x, c_i) and (c_i, c_{i+1}) in $2\log(i)$ operations. □

Using these results yields a minor improvement of the "wrapping" algorithm proposed by Chan [4] to compute the convex union of upper hulls.

Hull to hull tangents While the gift-wrapping algorithm can take advantage of various features of the instances which make them easier, it is still performing in super-linear time to the size of the upper hulls. In the simple case of two side by side upper hulls, of respective sizes n_1 and n_2, there is a certificate of constant size, and as it can be solved in $\mathcal{O}(\log n_1 + \log n_2 + h)$ time, where h is only required to output the result: Kirpatrick and Seidel [10] showed that, if the n_1 points of a first upper hull are on the left of the n_2 points of a second upper hull, then the convex union of the two upper hulls can be computed in $\mathcal{O}(\log(n_1) + \log(n_2))$ time. Their algorithm is based on the search for a *tangent* of the two upper hulls, i.e. a line which touches each hull at a single point [13] (instead of zero or two, as most lines do). A slight modification of their algorithm makes it adaptive in the position of the point of the tangent in the second hull:

Lemma 3. *Given upper hulls A and B, each represented in the clockwise order in an array, with respective sizes m and n, such that all the points of A are at the left of all the points of B, the tangent $(A[i], B[j])$ from A to B can be computed in time $\mathcal{O}(\log i + \log j)$.*

Proof. This is just an adaptive variant of the original algorithm from Kirpatrick and Seidel [10], performing a doubling search [2] on both hulls instead of mere binary searches. □

2.3 Adaptive Convex Hull Algorithm

Theorem 2. *There is a deterministic algorithm in the comparison model which computes the convex union for an instance of convolution δ composed of k sorted upper hulls of respective sizes $n_1, \ldots, n_k$ in $\mathcal{O}(\delta \sum \log(n_i/\delta))$ comparisons, which is also in $\mathcal{O}(\delta k \log(n/\delta k))$.*

Proof (sketch). Consider Algorithm 1: after identifying the first point a of the convex hull A, it iteratively computes the tangent of a with any other hull using Lemma 2, and computes the intersection of these tangents with A using Lemma 1. Each of those tangents yields either an arc on which A does not intersect the concerned hull, so that some points of this hull are *certified* not to contribute to the convex hull; or a bridge from A to this hull, such that some points from A are certified not to contribute to the convex hull.

We prove the correctness of the algorithm by induction. Consider one iteration of the outer loop. The following invariant is kept: After each iteration, A is on the merged hull. In the base case, a is the first point on the merged hull, thus the invariant is kept.

In the induction step, we have two possible cases:

Algorithm 1 Convex Upper Hull algorithm

```
Identify the starting point a (from Hull A) of the convex union;
repeat
  for each other hull B do
    compute the tangent from a to B, touching B at b
    and the rightmost intersection a' of (a,b) with A;
    if the segment ab does not cut A (i.e. a' is right of b) then
      // B does not cut A in the slide [a,a']
      ignore further the points left of a' in B;
      memorize (a,b)'s slope;
    else
      // B cuts A exactly once in the slice [a,a']
      compute the bridge [c,d] between the points from a to a' in A and the points right of b in B;
      memorize the slope of this bridge;
      ignore further the predecessors of d in B;
    end if
  end for
  if the highest slope s corresponds to at least one bridge then
    output and further ignore points of A left of d;
    switch (a,A) to (d,B);
  else
    update a to the last point of A of higher slope than s
    (but don't ignore its predecessors);
  end if
until no point is left in any hull
```

- The highest slope s corresponds to the bridge between A and B. At the end of the iteration, a is the first point of the current hull and on the merged hull. Therefore, the invariant is kept.
- The highest slope s corresponds to the arc with the highest slope. In this case, any other hull is below this arc. Since a is the last point on A above this arc, a is on the merged hull. The invariant is kept. □

3 Adaptive Computational Lower Bound

In the previous section we proved that our algorithm performs better than a naive algorithm on many instances. To complete this result, we prove that our algorithm takes optimally advantage of the easiness of the instances as measured by our difficulty measure. For that we show that no randomized algorithm can perform better than our deterministic algorithm, in the worst case over instances of fixed size and difficulty, asymptotically in both the size and the difficulty of the instance.

The main work is done in Lemma 4, which defines a probability distribution which is "bad" for any deterministic algorithm. Theorem 3 merely translates the corresponding lower bound into a lower bound on the worst case performance of any randomized algorithm, through a direct application of the Yao principle [15].

Lemma 4. *For any fixed value of $k, n_1, \ldots, n_k, \delta$, there is a probability distribution over instances of convolution δ composed of k sorted upper hulls of respective sizes $(n_1, \ldots, n_k)$ such that any (deterministic or) randomized algorithm computing the sorted convex hull of these instances performs $\Omega(\delta \sum \log(n_i/\delta))$ comparisons on average. Also, under similar conditions except that the total number of points n of the instance is fixed (instead of the sizes $(n_1, \ldots, n_k)$ of each object composing*

the instance), there is a probability distribution over instances matching the criteria such that A performs $\Omega(\delta k \log(n/\delta k))$ comparisons on average.

Proof (sketch). The proof proceeds in two steps: first we define a distribution over "elementary" instances of arbitrary size but small difficulty, then we show how to combine them to form distribution over instances of arbitrary size and difficulty, by combining several elementary instances.

In our context, an "elementary" instance is an instance of convolution 1. We define a distribution over elementary instances composed of k of respective sizes $(n_1, \ldots, n_k)$ (resp. of total size n), such that any deterministic algorithm performs $\Omega(k)$ searches on average, corresponding to an average cost of $\Omega(\sum_{i=2}^{k} \log(n_i))$ comparisons.

We show how to combine δ elementary instances composed of k hulls of respective sizes $n_1/\delta, \ldots, n_k/\delta$ (resp. of total size n/δ) into a larger instance composed of sorted upper hulls of respective sizes $(n_1, \ldots, n_k)$ (resp. of total size n) and of convolution δ, such that any algorithm solving this instance has to solve each of the elementary instance composing it. Applying this combination to elementary instances randomly drawn from the probability distribution described above yields a distribution over instances of desired size and convolution, forcing any deterministic algorithm to perform $\Omega(\delta \sum \log(n_i/\delta))$ comparisons (resp. $\Omega(\delta k \log(n/\delta k))$ comparisons) on average. □

Theorem 3. *For any fixed value of $k, n_1, \ldots, n_k, \delta$, and for any (deterministic or) randomized algorithm A computing the sorted convex hull of sorted upper hulls, there is an instance of convolution δ composed of k sorted upper hulls of respective sizes $n_1, \ldots, n_k$ such that A performs $\Omega(\delta \sum \log(n_i/\delta))$ comparisons on it. Also, under similar conditions except that the total number of points n of the instance is fixed (instead of the sizes $(n_1, \ldots, n_k)$ of each object composing the instance), there is an instance matching the criteria such that A performs $\Omega(\delta k \log(n/\delta k))$ comparisons on it.*

Proof. Applying the minmax theorem [12] from game theory, the Yao principle [15] states that the average complexity of the best deterministic algorithm on the worst probability distribution of instances is equal to the worst case complexity of the best randomized algorithm. Applying this principle to Lemma 4 directly yields the two results. □

4 Conclusion

Very large set of points for which a convex hull is required will not appear "out of nowhere": most likely, they will be formed of several objects from a library, for which a convex hull can be precomputed. In this context, we have given an algorithm to compute the convex hull of a set of convex hulls which outputs a description of the convex hull in a write-only streams, use little working space, and take advantage of instances where the relative positions of the objects makes the convex hull easier to compute. While those improvements do *not* change the complexity of the algorithm *in the worst case* over instances of fixed input and output size, they change the complexity of many instances which are likely to happen in practice, and which we formally identify through the definition of the certificate of an instance, and of a measure of the *difficulty* of the instance. Our techniques can be applied to compute the intersection of convex upper hulls [5] (and hence the intersection of any convex object), the union of convex upper hulls (and hence the contour of the union of any set of convex objects), and the intermediate relaxations of those problems: given k

convex planar upper hulls and a parameter $t \leq k$, what is the region covered by at least t convex planar upper hulls? Clearly this is the intersection for $t = k$ and the union for $t = 1$: this relaxation truly generalizes its equivalent on sorted sets in one dimension [1].

As the basic operations are clearly identified in each algorithm, our results are easily generalizable to the transdichotomous computational model as well: each of the basic operation can be supported in time $\mathcal{O}(\log n / \log \log n)$ using a precomputed index [6].

The ideas presented in this paper also apply to other problems, such as the difference of convex objects, or some more artificial relaxation between the union and the convex hull, defined on the model of our relaxation between the intersection and the contour of the union. The concept of certificate is easily generalized to higher dimensions, but whereas in the plane the choice of the best bridge between two hulls is well defined, this choice is not well defined even in three dimensions. Hence, the generalisation of our approach to three dimensions (and above) is still an open problem. A first goal would be to perform an adaptive analysis of the two base cases: the convex hull of two totally disjoint convex polytopes, and of the certification that one convex polytope is totally included in another. Algorithms are known for both, but it is not clear how to analize them adaptively.

At a more general level, we showed that in the same way as the output-sensitive analysis is finer than the typical worst case analysis over instances of fixed size; an even finer analysis can be performed for some problems in computational geometry, in order to yield further improvements over algorithms suggested by the output-sensitive analysis.

Acknowledgements: Many thanks to Alejandro López-Ortiz for suggesting this direction of research, and to Timothy Chan and Alejandro Salinger for pointing to previous works. This work was supported by a discovery grant from NSERC.

Bibliography

[1] J. Barbay and C. Kenyon. Adaptive intersection and t-threshold problems. In *Proceedings of the 13th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 390–399. Society for Industrial and Applied Mathematics (SIAM), 2002.

[2] J. L. Bentley and A. C.-C. Yao. An almost optimal algorithm for unbounded searching. *Information processing letters*, 5(3):82–87, 1976.

[3] W. H. Burge. Sorting, trees, and measures of order. *Information and Control*, 1(3):181–197, 1958.

[4] T. M. Chan. Optimal output-sensitive convex hull algorithms in two and three dimensions. *GEOMETRY: Discrete & Computational Geometry*, 16, 1996.

[5] T. M. Chan. Deterministic algorithms for 2-d convex programming and 3-d online linear programming. In *SODA: ACM-SIAM Symposium on Discrete Algorithms (A Conference on Theoretical and Experimental Analysis of Discrete Algorithms)*, 1997.

[6] T. M. Chan. Point location in o(log n) time, voronoi diagrams in o(n log n) time, and other transdichotomous results in computational geometry. In *Proc. 47th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 333–342, 2006.

[7] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry, Algorithms and Applications*. Springer, 1997.

[8] E. D. Demaine, A. López-Ortiz, and J. I. Munro. Adaptive set intersections, unions, and differences. In *Proceedings of the 11th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 743–752, 2000.

[9] V. Estivill-Castro and D. Wood. A survey of adaptive sorting algorithms. *ACM Computing Surveys*, 24(4):441–476, 1992.

[10] D. G. Kirkpatrick and R. Seidel. The ultimate planar convex hull algorithm? *SIAM J. Comput.*, 1986. 15(1):287–299.

[11] H. Mannila. Measures of presortedness and optimal sorting algorithms. *IEEE Trans. Computers*, 34(4):318–325, 1985.

[12] J. V. Neumann and O. Morgenstern. *Theory of games and economic behavior*. 1st ed. Princeton University Press, 1944.

[13] J. O'Rourke. *Computational Geometry in C*. Cambridge University Press, 1994.

[14] O. Petersson and A. Moffat. A framework for adaptive sorting. *Discrete Applied Mathematics*, 59(2):153–179, 1995.

[15] A. C. Yao. Probabilistic computations: Toward a unified measure of complexity. In *Proc. 18th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 222–227, 1977.