Pattern Recognition Letters 3 (1985) 29–34
North-Holland

January 1985

On the ultimate convex hull algorithm in practice

Mary M. McQUEEN and Godfried T. TOUSSAINT
School of Computer Science, McGill University, 805 Sherbrooke Street West, Montreal, Quebec H3A 2K6, Canada

Received 2 April 1984

Abstract: Kirkpatrick and Seidel [13,14] recently proposed an algorithm for computing the convex hull of n points in the plane that runs in $O(n \log h)$ worst case time, where h denotes the number of points on the convex hull of the set. Here a modification of their algorithm is proposed that is believed to run in $O(n)$ expected time for many reasonable distributions of points. The above $O(n \log h)$ algorithms are experimentally compared to the $O(n \log n)$ 'throw-away' algorithms of Akl, Devroye and Toussaint [2,8,20]. The results suggest that although the $O(n \log h)$ algorithms may be the 'ultimate' ones in theory, they are of little practical value from the point of view of running time.

Key words: Convex hull, algorithms, complexity, computational geometry.

1. Introduction

The convex hull of a finite set of points in the plane is defined as the minimum area convex polygon enclosing the set. As one of the earliest problems studied extensively in computational geometry, algorithms for computing the convex hull abound. Besides theoretical importance, the convex hull is of practical relevance as a tool in pattern recognition [19]. Hence, improved time bounds and faster running time have been the focus of research.

Many algorithms with $O(n \log n)$ worst case time bounds have been described recently [3,5,11,15,16,17]. Several papers have proved an $\Omega(n \log n)$ lower bound for finding the convex hull [4,10,16,17,21]. Another pair of algorithms [9,12] have been proposed with a worst case time bound of $O(nh)$, where h is the number of convex hull vertices.

Kirkpatrick and Seidel [13] have presented an algorithm for determining the planar convex hull with worst case time complexity $O(n \log h)$, sensitive to both n and h. A second more comprehensive paper [14] proves $\Omega(n \log h)$ the lower bound for the problem.

Algorithms with linear expected times have also been described [5,18]. Furthermore, Akl, Toussaint, and Devroye [2,8,20] have proved that an increasing 'throw-away' preprocessing step will cause any of the previously mentioned algorithms to run in $O(n)$ expected time for certain distributions of the input. Actual running times have also been reported in [6].

In this paper we present a modification of Kirkpatrick and Seidel's ultimate planar convex hull algorithm [13,14] which is believed to run in linear expected time, for some distributions of points. Implementations of this and two other algorithms are described. The first algorithm is Kirkpatrick and Seidel's in its original form. The other is Kirkpatrick and Seidel's with 'throw-away' preprocessing. The running times of these algorithms are compared for several distributions. This analysis is especially valuable given that no proof presently exists of the linear expected time of the modified algorithm. A second reason for implementing Kirkpatrick and Seidel's convex hull algorithm is to check whether the algorithm, which is perhaps the theoretical ultimate, is in fact practical.

Research supported by N.S.E.R.C. grant no. A9293 and F.C.A.C. grant no. EQ1678.

2. Description of the algorithms

Kirkpatrick and Seidel's algorithm is described briefly for completeness and to facilitate the description of the modified version of the algorithm. A more extensive description of the algorithm appears in the original paper [13].

Algorithm 1. Kirkpatrick and Seidel's original algorithm

```
Procedure UPPER HULL(S)
Input  : A set $S=\{P_1,\dots,P_n\}$ of $n$ points in the
         plane, where $x(P)$ and $y(P)$ denote standard Cartesian coordinates.
Output : The sequence of indices of the vertices on
         the upper hull of $S$.
1.1. Let $P_{x\min}$ and $P_{x\max}$ be the points with minimum and maximum $x$ coordinate.
1.2. Let
         $T=\{P_{x\min},P_{x\max}\}$
             $\cup\{p \text{ in } S \mid x(P_{x\min})<x(P)<x(P_{x\max})\}$.
1.3. CONNECT($x$min, $x$max, $T$).

Procedure CONNECT(l, r, S)
2.1. Find a real number $a$ such that $a$ is the mean
     of the median $x$ coordinate and the next largest $x$ coordinate.
2.2. Find the 'bridge' over the vertical line
         $A=\{(x,y)\mid x=a\}$; $(i,j):=$ BRIDGE$(S,a)$.
2.3. Let $S_{\text{left}}=\{P \text{ in } S\mid x(P)\le x(P_i)\}$.
     Let $S_{\text{right}}=\{P \text{ in } S\mid x(P)\ge x(P_j)\}$.
2.4. If $i=l$ then print $(i)$.
         else CONNECT$(l,i,S_{\text{left}})$.
     If $j=r$ then print $(i)$.
         else CONNECT$(j,r,S_{\text{right}})$.

Function BRIDGE(S, a)
Input  : A set $S=\{P_1,\dots,P_n\}$ of points and a real
         $a$ representing the line $A=\{(x,y)\mid x=a\}$.
Output : A pair $(i,j)$, where $P_i$ and $P_j$ are the left
         and right bridge points respectively.
```

The function BRIDGE pairs up all points in the set S and defines a line through each pair. The median slope of these lines is computed and the support line of the set having this slope is determined. If this support line contains points on each side of line A, then the vertices on this line with minimum and maximum x coordinate are the bridge. Otherwise, if the support line contains no points to the right of line A, then of the point pairs defining lines with slope less than or equal to the median slope, the point with the least x coordinate is discarded. Or, if the support line contains no points to the left of line A, then of the point pairs defining lines with slope greater than or equal to the median slope, the point with the greatest x coordinate is discarded. BRIDGE calls itself recursively with the remaining points, until the bridge is found.

Procedure LOWER HULL is defined analogously to Procedure UPPER HULL.

In Step 2.3, the algorithm removes from consideration the points under the bridge. In Figure 1, the shaded area under the upper hull represents the location of ignored points.

Algorithm 2. Kirkpatrick and Seidel's algorithm with modification

Algorithm 2 is defined in much the same way as Algorithm 1. However, Steps 1.2 and 2.3 are replaced by the following modified steps.

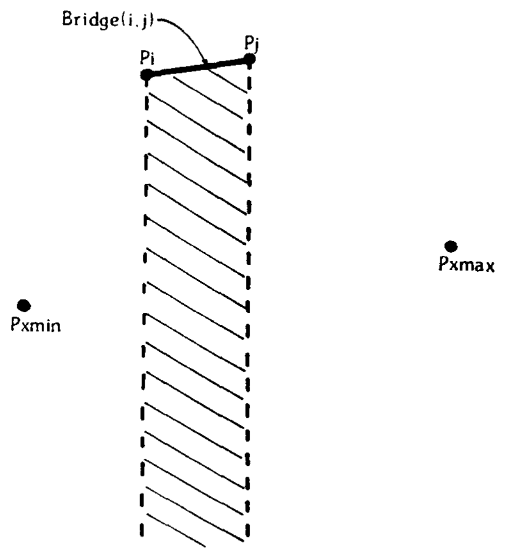

Figure 1. Points in the shaded area are discarded by Step 2.3 of Kirkpatrick and Seidel's original algorithm.

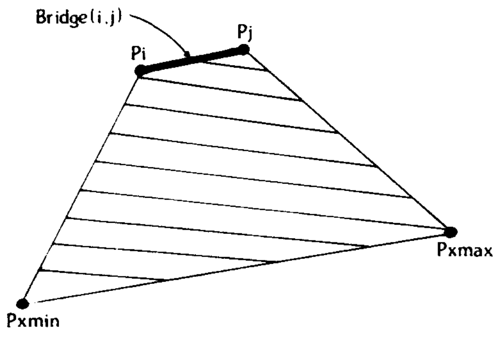

Figure 2. Points in the shaded area are discarded by the modified Step 2.3.

1.2. Let

$T = \{P \text{ in } S \mid x(P) \text{ above the line through } P_{x\min} \text{ and } P_{x\max}\}$.

⋮

2.3. Let

$S_{\text{left}} = \{P \text{ in } S \mid x(P) \le x(P_i) \text{ and } x(P) \text{ above the line through } P_k \text{ and } P_i\}$.

Let

$S_{\text{right}} = \{P \text{ in } S \mid x(P) \le x(P_i) \text{ and } x(P) \text{ above the line through } P_j \text{ and } P_m\}$.

In the modified Step 2.3, the algorithm removes from consideration points in a larger area than in the original Step 2.3. This new step cannot discard convex hull vertices, as these must lie outside any partial hull. In Figure 2, the shaded area under the upper hull represents the location of ignored points.

Algorithm 3. Kirkpatrick and Seidel's algorithm with 'throw-away' preprocessing

Algorithm 3 is defined in much the same as Algorithm 1. Step 1.2 is replaced by the following modified version. This method of 'throw-away' is fully documented elsewhere [2,20].

1.2. Let xymax, yxmax and $y$max be 3 other extremal points falling on the convex hull; where $P_{xy\max}$ maximizes

$(-x(P_i) + y(P_i))$,

$P_{yx\max}$ maximizes

$(x(P_i) + y(P_i))$

and $P_{y\max}$ is the point with maximum y coordinate. Let

$$T = \{P_{x\min}, P_{yx\max}, P_{y\max}, P_{xy\max}, P_{x\max}\} \cup \{P \text{ in } S \mid P \text{ is above the convex polygonal chain } (P_{x\min}, P_{yx\max}, P_{y\max}, P_{xy\max}, P_{x\max})\}.$$

3. Description of implementation

All these algorithms were implemented in Stanford PASCAL. The 'sets' were implemented using linked lists. CONNECT and BRIDGE were implemented as recursive procedures. Finding the median (such as the median line A or the median slope) was implemented using the algorithm of Blum et al. [7] as described in [1]. The code for ULTIMATE1, ULTIMATE2 and ULTIMATE3 are available from the authors.

ULTIMATE1. Implementation of Kirkpatrick and Seidel's algorithm

The lower hull is found by modifying the method used to find the upper hull. Since the points are generated in the unit square, the y coordinates were subtracted from 1 to 'flip' all the points. This enabled the UPPER HULL procedure to handle the case of finding the lower hull. Also, CONNECT was run from right to left, for procedure LOWER HULL, to allow the main program to produce all the points of the hull in order.

ULTIMATE2. Implementation of the modified algorithm

The condition 'P above the line through P_1 and P_i' is true if the crossproduct of P_1, P_i, P is positive.

ULTIMATE3. Implementation of the algorithm with 'throw-away'

The condition 'P above the polygonal chain

$(P_{x\min}, P_{yx\max}, P_{y\max}, P_{xy\max}, P_{x\max})$'

is tested as follows:

For each point P, find the edge (P_e, P_f) such that

$$x(P_e) \geq x(P) \geq x(P_f).$$

Retain the point P if the crossproduct of P_e, P_f, P is positive.

Note that the polygonal chain may have at most 4 edges, but may have fewer in the special case where two extremal points are equal. Duplicates can simply be 'deleted' from the chain and the method above is then applied.

4. Experimental results

The three algorithms implemented, ULTIMATE1, ULTIMATE2 and ULTIMATE3, were run on an AMDAHL 5850 computer using the Stanford PASCAL Compiler. Monte-Carlo simulations were carried out. There were two simple sizes; 1000 and 2000 points in the plane. Pseudo random samples were generated for four distributions:

(1) uniform in the unit square,
(2) circular normal,
(3) uniform in a unit circle, and
(4) uniform on the boundary of a unit circle.

The programs were timed using FORTRAN library subroutines TIMER1 and TIMER2.

Table 1
Mean running times (milliseconds) of three programs for five random samples of size n for four distributions

Distribution	PROGRAM	$N = 1000$	$N = 2000$
Uniform in the unit square	ULTIMATE1	1363.8	2597.6
	ULTIMATE2	524.7	1076.3
	ULTIMATE3	271.6	323.6
Circular normal	ULTIMATE1	1299.4	2726.2
	ULTIMATE2	473.1	994.0
	ULTIMATE3	178.3	313.1
Uniform in a unit circle	ULTIMATE1	2019.6	4757.3
	ULTIMATE2	543.5	1191.0
	ULTIMATE3	285.5	714.5
Uniform on the boundary of a unit circle	ULTIMATE1	4563.7 (1000)[a]	10848.3 (2000)[a]
	ULTIMATE2	2222.2 (993)[a]	5254.9 (1987)[a]
	ULTIMATE3	2187.9 (993)[a]	5181.3 (1988)[a]

[a] Mean number of convex hull points found.

The results of the experimental runs are shown in Table 1. Mean times for five random samples for each combination of program, distribution, and sample size are presented. It is evident that for every distribution ULTIMATE1 is the slowest, ULTIMATE2 is faster, and ULTIMATE3 is the fastest.

The case where all generated points lie on the boundary of a circle and also on the convex hull, ULTIMATE1 is slower than the other two programs. The gap between ULTIMATE2 and ULTIMATE3 is quite small for this distribution.

For points generated uniformly in the unit square, ULTIMATE2 is believed to run in $O(n)$ expected time and the 'throw-away' preprocessing used in ULTIMATE3 has been shown to enable algorithms with even $O(n^2)$ worst case time complexity to run in $O(n)$ expected time. The belief for ULTIMATE2 is substantiated by the fact that it runs two and a half times faster than the original algorithm, ULTIMATE1, which does not run in linear expected time for these distributions. The proof of ULTIMATE3's expected time is corroborated by its excellent performance here, showing how powerful the 'throw-away' procedure is [2,8,20]. Notice that for this distribution ULTIMATE3 can compute the convex hull of 2000 points before ULTIMATE2 has finished solving the problem for 1000 points.

Another interesting point that should be made is that all three programs are exceedingly slow. It has been shown by Bhattacharya and Toussaint [6] that Eddy's $O(n^2)$ algorithm [9] with 'throw-away' preprocessing computes the convex hull of 100 points on the boundary of a circle in 152.9 milliseconds. Though Eddy's algorithm appears much faster, the environments in which the Monte-Carlo simulations were run are very different (Stanford PASCAL vs. FORTRAN G1; AMDAHL 5850 vs. AMDAHL V-7), and are not directly comparable. Thus, no rigorous conclusion may be drawn.

All three implementations are reliable. The fact that ULTIMATE1 always finds n convex hull points for n points generated uniformly on the boundary of a unit circle is astounding. ULTIMATE2 and ULTIMATE3 are close behind finding 99% of the convex hull vertices. The figures for ULTIMATE2 and ULTIMATE3 differ from ULTIMATE1 due to the precision involved in the computations of the crossproducts computed in the former implementations.

It should be noted that almost all existing convex hull algorithms make use of crossproducts.

The reliability of these programs may be compared to that of the implementations of Eddy's [9] and Akl and Toussaint's [3] algorithms given in Bhattacharya and Toussaint [6]. On the average, the implementation of Eddy finds 944 out of 1000 points on the boundary of a circle, while that of Akl and Toussaint returns 971.

5. Conclusions

Implementations of various versions of Kirkpatrick and Seidel's algorithm were presented and were found to be very slow in general. Though having a worst case complexity of $O(n \log h)$ which is theoretically optimal, this complexity has a very large constant factor. A modified version of this original algorithm believed to run in linear expected time was shown to be about two and a half times faster than the implementation of Kirkpatrick and Seidel's algorithm. However, it could not keep up with Kirkpatrick and Seidel's algorithm with 'throw-away' preprocessing, showing how powerful this technique is. Proof of the $O(n)$ expected time performance of the modified algorithm, even for uniform distributions, remains an open problem. Hence, the theoretically 'ultimate' convex hull algorithm for points in the plane does not live up to expectations in practice, where the best algorithm to date (with respect to space and time) still appears to be that of Akl and Toussaint [2] as implemented by Bhattacharay and Toussaint [6]. However, as the experimental results show, the algorithm of Kirkpatrick and Seidel is the most accurate from the numerical point of view, of all algorithms tested so far by the authors. Thus, if the primary practical consideration is accuracy over and above running time then their algorithm may still be preferred.

References

[1] Aho, A.V., J.E. Hopcroft and J.D. Ullman (1974). *The design and Analysis of Computer Algorithms.* Addison-Wesley, Reading, MA.

[2] Akl, S.G. and G.T. Toussaint (1978). Efficient convex hull algorithms for pattern recognition applications. *Proceedings 4th International Joint Conference Pattern Recognition, Kyoto, Japan.* Nov. 1978, 483-487.

[3] Akl, S.G. and G.T. Toussaint (1978). A fast convex hull algorithm. *Information Processing Letters* 7, 219-222.

[4] Avis, D. (1979). On the complexity of finding the convex hull of a set of points. Technical Report No. SOCS 79.2, School of Computer Science, McGill University.

[5] Bentley, J.L. and M.I. Shamos (1978). Divide and conquer for linear expected time. *Information Processing Letters* 7, 87-91.

[6] Bhattacharya, B.K. and G.T. Toussaint (1981). A time-and-storage efficient implementation of an optimal planar convex hull algorithm. Technical Report No. SOCS-81.40, School of Computer Science, McGill University, December.

[7] Blum, M., R.W. Floyd, V.R. Pratt, R.L. Rivest and R.E. Tarjan (1972). Time bounds for selection. *Journal of Computer and System Sciences* 7, 448-461.

[8] Devroye, L. and G.T. Toussaint (1981). A note on linear expected time algorithms for finding convex hulls. *Computing* 26, 361-366.

[9] Eddy, W.F. (1977). A new convex hull algorithm for planar sets. *ACM Transactions on Mathematical Software* 3, 398-403 and 411-412.

[10] van Emde Boas, P. (1980). On the $O(n \log n)$ lower-bound for convex hull and maximal vector determination. *Information Processing Letters* 10, 132-136.

[11] Graham, R.L. (1972). An efficient algorithm for determining the convex hull of a finite planar set. *Information Processing Letters* 1, 132-133.

[12] Jarvis, R.A. (1973). On the identification of the convex hull of a finite set of points in the plane. *Information Processing Letters* 2, 18-21.

[13] Kirkpatrick, D.G. and R. Seidel (1982). The ultimate planar convex hull algorithm? *Proceedings 20th Annual Allerton Conference on Communications, Control and Computing,* Oct. 1982.

[14] Kirkpatrick, D.G. and R. Seidel (1982). The ultimate planar convex hull algorithm? Technical Report No. 83-577, Department of Computer Science, Cornell University, Ithaca, NY.

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

[16] Preparata, F.P. and S.J. Hong (1977). Convex hulls of finite sets of points in two and three dimensions. *Communications of the ACM* 20, 87-93.

[17] Shamos, M.I. (1978). *Computational Geometry.* Ph.D. Thesis, Yale University, New Haven, CT.

[18] Shamos, M.I. (1981). Personal Communication to G. Toussaint as described in Toussaint, G.T. 'Computational Geometric Problems in Pattern Recognition. In J. Kittler, K.S. Fu and L.F. Pau, Eds., *Pattern Recognition Theory and Applications. Nato Advanced Study Institute.* Oxford University, 73-91.

[19] Toussant, G.T. (1978). The convex hull as a tool in pattern

recognition. *Proceedings AFOSR Workshop in Communication Theory and Applications.* Provincetown, MA, September.

[20] Toussaint, G.T., S.G. Akl and L.P. Devroye (1978). Efficient convex hull algorithms for points in two and more dimensions. Technical Report No. SOCS 78.5, School of Computer Science, McGill University.

[21] Yao, A.C. (1981). A lower bound to finding convex hulls. *Journal of the ACM* 28, 780-789.