Term paper
on
Chan's convex hull algorithm

by

Sunandita Patra

Roll number: 09CS3037

3rd year Dual degree student

Computer Science and Engineering

IIT Kharagpur

Email: sunandita.patra@gmail.com,

sunandita.patra@cse.iitkgp.ernet.in

The problem

The convex hull problem is, given a set of n points, P in the 2D plane, we have to find a representation of P's convex hull. The convex hull is a closed convex polygon. It can be represented simply with the help of counterclockwise or clockwise enumeration of the vertices of the convex hull. Ideally, the vertices of the hull should be extreme points, in the sense that if three points lie on an edge of the boundary of the convex hull, then the middle point should not be reported as a vertex of the hull. Thus, any algorithm to find the convex hull of a point said should have the following form:

Input A set of points, P in the 2D plane

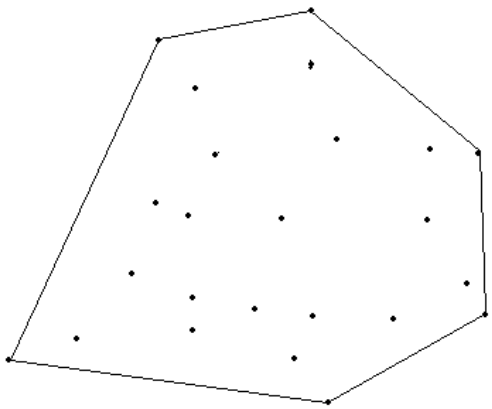
Output The vertices of the convex hull of P, in clockwise or counter clockwise order

Towards a solution

Here is a list of some well-known 2D hull algorithms (from reference [5]). The notation used is n = number of points in the input set, P and h = number of vertices on the convex hull. The list is ordered by date of first publication.

Algorithm	***Speed***	***Discovered By***
Brute Force	$O(n^4)$	[Anon, the dark ages]
Gift Wrapping	$O(nh)$	[Chand & Kapur, 1970]
Graham Scan	$O(n \log n)$	[Graham, 1972]
Jarvis March	$O(nh)$	[Jarvis, 1973]

Quick Hull	O(nh)	[Eddy, 1977], [Bykat, 1978]
Divide-and-Conquer	O(n log n)	[Preparata & Hong, 1977]
Monotone Chain	O(n log n)	[Andrew, 1979]
Incremental	O(n log n)	[Kallay, 1984]
Marriage-before-Conquest	O(n log h)	[Kirkpatrick & Seidel, 1986]
Chan's algorithm	O(n log h)	[T. M. Chan, 1996]



A set of points and its convex hull

Observations about convex hull

Observation 1

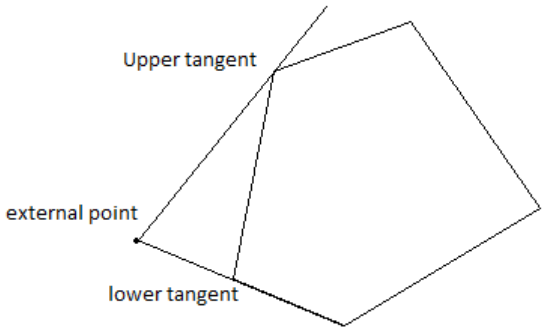
Suppose a point, p lies on the convex hull of a set of n points, P. Then, p will also be a point on the convex hull of any subset of P containing p. Thus, a point that does not lie on any subhull of any subset of P containing p, it can never lie on the convex hull of P.

Observation 2

We reduce the number of candidates by grouping the points into several groups and finding their convex hulls. Now, only these hull vertices are candidates to be vertices of the convex hull of P. The convex hull of these smaller convex hulls is the convex hull of P. The interior points of these subhulls need not be considered in future steps.

Observation 3

The point of tangency to a convex hull from an external point will be a vertex of the convex hull or the tangent will coincide with an edge, in which case the second point is considered to be the point of tangency.



Output sensitive convex hull algorithms

It turns out that in the worst-case, convex hulls cannot be computed faster than in O($n$log$n$) time. One way to see this intuitively is to observe that the convex hull itself is sorted along its boundary, and hence if every point lies on the hull, then computing the hull requires sorting of some form. Yao proved the much harder result that determining which points are on the hull (without sorting them along the boundary) still requires O($n$log$n$) time. However, both of these results rely on the fact that all (or atleast a constant fraction) of the points lie on the convex hull. This is often not true in practice.

The Quick Hull and Jarvis's March algorithms suggest the question of how fast can convex hulls be computed if we allow the running time to be described in terms of both the input size n and the output size h. Many geometric algorithms have the property that the output size can be a widely varying function of the input size, and worst-case output size may not be a good indicator of what happens typically. An algorithm which attempts to be more efficient for small output sizes is called an *output sensitive algorithm*, and running time is described as an asymptotic function of both input size and output size.

Lead up to Chan's algorithm

We can observe that any convex hull algorithm must take at least $O(n)$ time (we have to look at each point atleast once), and given that "log n" factor arises from the fact that you need to sort the at most n points on the hull, if you were told that there are only h points on the hull, then a reasonable target running time is $O(n \log h)$. (We will see that this is optimal.)

Kirkpatrick and Seidel discovered a relatively complicated $O(n \log h)$ time algorithm, based on a clever pruning method in 1986. About 10 years later, Timothy Chan came up with a much simpler algorithm with the same running time. One of the interesting aspects of Chan's algorithm is that it involves combining two slower algorithms (Graham's scan and Jarvis's March) together to form an algorithm that is faster than either one.

The problem with Graham's scan is that it sorts all the points, and hence will always have an Ω (n log n) running time, irrespective of the size of the hull. On the other hand, Jarvis's march can perform better if you have few vertices on the hull, but it takes Ω (n) time for each hull vertex.

Chan's approach and analysis

Chan's idea was to partition the points into groups of equal size. There are a maximum of m points in each group, and so the number of groups is k = ceiling (n/m). For each group, we compute its hull using Graham's scan, which takes $O(m \log m)$ time per group, for a total time of $O(km \log m) = O(n \log m)$.

Thus, after this step, we have k subhulls. Note that since the points in a set were selected arbitrarily, these subhulls may intersect with each other, one may completely lie within another, or they may be disjoint.

Next, we use the Jarvis march algorithm (to compute convex hull of a 2D point set) on the groups. Here we take advantage of the fact that you can compute the tangent between a point and a convex m-gon in $O(\log m)$ time.

The idea is to find the next vertex of the convex hull, p_{i+1} by knowing a current vertex, p_i. We select the leftmost point among all the vertices of all the subhulls. This point is guaranteed to be in the convex hull of P. We name this point as p_0, $i = 0$. Then, we find the tangents from p_i to each of the subhulls. The tangent which makes the lowest angle with $p_i p_{i-1}$, its point of contact with the subhull will surely be the next vertex of the convex hull of P.

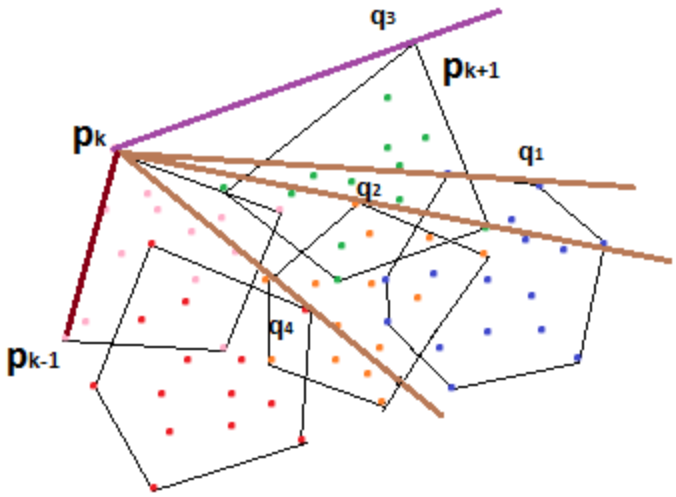
> **Note** Finding the tangent from an external point, q to a convex hull:
>
> Without loss of generality, we can assume that the vertices of the convex hull are ordered in clockwise direction. We perform a binary search on the list of vertices, choosing sides by calculating the angles made by the edge qp_i with the two edges which contain the point, p_i. p_i is the point of tangency if the two edges containing p_i lie on the same side of qp_i. This new line segment is the corresponding tangent.

We repeat until the newly found hull vertex coincides with p_0.

So, as before there are h steps of Jarvis's march, but because we are applying it to k convex hulls each step takes O (k log m) time. Each iteration consists of finding a tangent using binary search $O(\log m)$ and then finding the tangent which makes the least angle with the current edge of the convex hull ($O(k)$). Thus, the cost of each iteration = $O(klogm)$ and total running time = $O(hk \log m) = O((hn/m) \log m)$.

Combining the two parts, we get a total of $O((n + hn/m) \log m)$ running time.

We can observe that if we set $m = h$, we can get $O(n \log h)$ running time. There is only one small problem here. We do not know what h is in advance, and therefore we do not know what m should be when running the algorithm. We will see how to take care of this later. The algorithm works correctly as long as $m >= h$.



Chan's algorithm through different stages

Chan's Partial Convex Hull Algorithm (Given m) (from reference [6])

Partial Hull (P, m)

1. Let k = *ceiling (n/m)*. Partition P into disjoint subsets such that $\bigcup_{i=1}^{k} Pi = P$, each of size at most m.
2. For i = 1 to k do:
   (a) Compute convex hull(P_i) using Graham's scan and store the vertices in an ordered array.

3. Let p_0 = (-infinity, 0) and let p_1 be the bottommost point of P.
4. For k = 1 to m do:
   (a) For i = 1 to k do:
   Compute point $q_i \in P_i$ that maximizes the angle $p_{k-1}p_kq$.
   (b) Let p_{k+1} be the point $q \in \{q_1,\ldots,q_r\}$ that maximizes the angle $p_{k-1}p_kq$.
   (c) If $p_{k+1} = p_1$ then return $<p_1,\ldots,p_k>$

5. Return "*failure(m was too small)*."

Choosing the value of *m*

The remaining thing is choosing the value of *m* correctly. We can choose m to be 1, 2, 3 and so on, till we get *m* >= *h*. But this will increase the time complexity. Binary search will definitely be more efficient than this but if we use binary search, we may guess too large value of *m* (*for example, m = n/2 = O(n)),* which will in turn increase the time to $O(n \log n)$

So, Chan uses the trick that we start with a small value of *m* and increase it rapidly. Since the dependence on *m* is only in the logarithmic term, as long as our value of *m* is within a polynomial of *h*, that is, $m = ch$ for some constant *c*, (in other words *m = O(h))* then the running time will still be $O(n \log h)$. So, our approach will be to guess successively larger values of *m*, each time squaring the previous value, until the algorithm returns a successful result. This technique is often called *doubling search* (because the unknown parameter is successively doubled). The only difference is that in Chan's algorithm, we will be squaring the old value rather than doubling.

Chan's Complete Convex Hull Algorithm (from reference [6])

Convex Hull (*P*)

1. For $t = 1, 2, 3, \ldots$ do:
    a. Let $m = \min(2^{2^t}, n)$
    b. Invoke Partial Hull (*P, m*), returning the result in *L*.
    c. If $L \neq$ "failure", then return *L*.

We should note that 2^{2^t} has the effect of squaring the previous value of *m*.

Time and space complexity

For the *t*-th iteration, running time = $O(n \log(m_t)) = O(n \log 2^{2^t}) = O(n2^t)$. We know that it will stop as soon as $2^{2^t} > h$, that is if $t = \text{ceiling}(\lg(\lg n))$. So, the total running time is

$$\sum_{t=1}^{\lg \lg h} n2^t = n \sum_{t=1}^{\lg \lg h} 2^t \leq n2\text{^}(1 + \lg \lg h) = 2nlg\ h = O(n \lg h)$$

which is what we wanted to achieve.

The storage requirement is clearly linear because we are storing only the vertices of the convex hull which is linear to the total number of points in the input, *n*.

References:

1. Timothy M. Chan: *Optimal Output Sensitive Convex Hull Algorithms in Two and Three Dimensions. Discrete Computational Geometry 16:361-368 (1996)*
2. *http://en.wikipedia.org/wiki/Chan's_algorithm*
3. Chris Harrison. *An Investigation of Graham's Scan and Jarvis' March* URL:*http://www.chrisharrison.net/projects/convexHull/index.html*
4. *http://softsurfer.com/Archive/algorithm_0201/algorithm_0201.htm#tangent_PointPolyC()*
5. *http://www.tcs.fudan.edu.cn/rudolf/Courses/Algorithms/Alg_cs_07w/Webprojects/Zhaobo_hull/index.html*
6. Lecture notes by David M. Mount