Various Implementations of the Convex Hull Problem

Neelesh C. A.
Computer Science Engineering
PES University
Bengaluru, India
neeleshca26@gmail.com

Ninaad R. Rao
Computer Science Engineering
PES University
Bengaluru, India
ninaadrrao@gmail.com

Abhishikta Sai
Computer Science Engineering
PES University
Bengaluru, India
abhishikta.sai21@gmail.com

***Abstract*—Convex hull or convex envelope or convex closure of a set X of points in the Euclidean plane or in a Euclidean space (or, more generally, in an affine space over the reals) is the smallest convex set that contains all the points in X.In other words, given a set of points in the plane, the convex hull of the set is the smallest convex polygon that contains all the points given. We implement Quickhull, Divide and Conquer, Jarvis' Algorithm and Graham Scan algorithm to achieve the problem of generating the convex hull of the set of points in 2D plane and analyze the time complexity of each of these four algorithms with growing input size and the growing range of the input plane.**

***Index Terms*—Convex Hull, Graham Scan Algorithm, Quickhull Algorithm, Divide and Conquer, Jarvis' Algorithm**

I. Introduction

A set of points S isconvexif for any two points in S, the line segment joining them is also inside the set. A point p is anextreme pointof a convex set S if p is not interior to any line segment connecting two points in the set.

The convex hull approach of enclosing a set of 2D points can be used in various applications. It can be used for Robot Motion Planning. The set of points could represent obstacles for the robot and it has to not hit any of these obstacles and move around. In such a scenario, the convex hull algorithm can be used. It can be used in shortest perimeter fence enclosing the set of points. The convex hull algorithm also represents the smallest convex set containing all N points. Convex hull has an application of virtual reality which is used to draw the boundary of some object inside an image as done in

Jarvis' Algorithm for implementing the convex hull algorithm has a time complexity of O(nh) where h is the number of vertices in the hull. If h becomes equal to n, then the time com

The Divide and Conquer approach of the convex hull algorithm has a time complexity of O(nlog(n)) which merges the convex hull of the left half of the points with the convex hull of the right half of the points by finding the upper and lower tangents of the two polygons and joining them altogether.

The Graham Scan Algorithm for the convex hull problem has a time complexity of O(nlog(n)) and the problem is solved by finding the bottom most point of the set of points in the polygon and by sorting these points by the polar angle made with respect to the bottom most point in the counterclockwise direction.

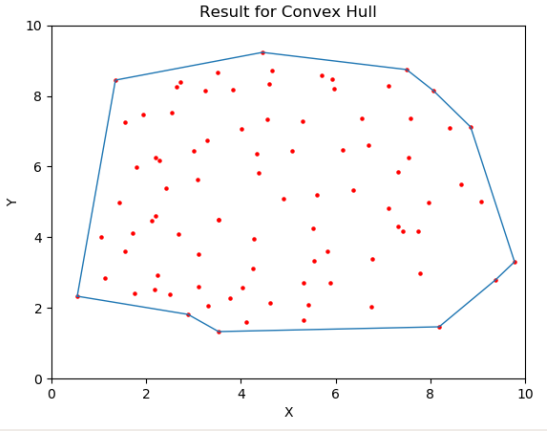

Fig. 1. An example of the Convex Hull Algorithm

The Quickhull algorithm for finding the convex hull has a average case time complexity of O(nlog(n)) but the worst case time complexity is $O(n^2$ which finds the minimum and the maximum values of the x coordinate, draws a line between the two points. This line will divide the entire plane into two parts and creates two subproblems.

II. The Divide and Conquer Approach

The key idea in the divide and conquer algorithm is that if we can have two convex hull, then we can merge the two convex hull and get the convex hull for a larger set of points. The two convex hull are merged by first finding the upper and lower tangents[1].

To find the two convex hulls, recursion is used. The number of points are divided until they are small, and a brute force approach is used to find the convex hull of the smaller set of points.

To find the upper and lower tangents, first the left most point of the right convex polygon and the right most point of the left convex polygon is taken. A line is drawn between these two points and it is checked if this passes through any of the polygons and if it does, then the points are moved upwards.

Algorithm 1 Divide_and_conquer(sorted_input)

```
0: procedure DIVIDE(INPUT)
0:     size ← size of input
0:     if size <= 10 then return Brute(input)
0:     lsize ← size/2
0:     rsize ← size/2
0:     lhull ← divide(input[0:lsize]
0:     rhull ← divide(input[lsize:size])
0:     procedure
       (RETURN )merge(lhull,rhull)
```

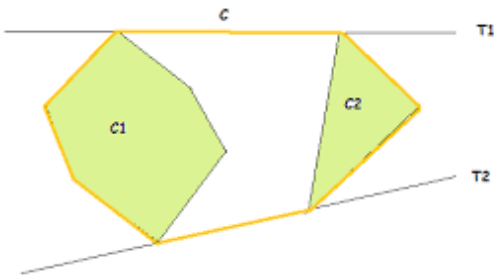

Fig. 2. An example of the merging of the two convex hull

If the line crossed the left convex polygon, then the next rightmost point is taken. Similarly if the line crossed the right convex polygon, then the next leftmost point on the polygon is taken and the upper tangent is found. Similarly, the lower tangent is found, except now the points move downwards and the hull is drawn as shown in the figure.

III. Graham's Scan

The idea is a common one in computational geometry, known as an incremental algorithm: add items one at a time to some structure, maintaining as you do a partial solution for the items added so far. The order in which items are added is often important. Graham Scan Algorithm begins with finding the base point which is the point with least distance from the origin. Then by using the base point we sort all the points based on their polar angle with the base point. So $O(n \log n)$ time is needed for an initial sorting stage, or less time if you want to assume e.g. that the points' coordinates are small integers.

In its second step the idea is to eliminate all the reflex vertices. A stack is initialized with first three points from the sorted array . The elimination procedure consists of a local test that takes the next vertex from the sorted array and two most recent points from the stack ,and checks whether the three constitute a left turn or right. If the turn is left we retain the vertex and push it into the stack. If the turn is right we pop the top vertex from the stack. This process happens till there are no points left to traverse in the sorted array.

The hull is obtained by converting the stack into an array.

The time complexity for sorting points in Graham Scan Algorithm is $O(n \log n)$ and the construction of hull takes $O(n)$ time as a point can either be pushed into the stack or discarded. Hence, the overall time complexity of constructing the convex hull using Gaham's scan algorithm is $O(n \log n)$.

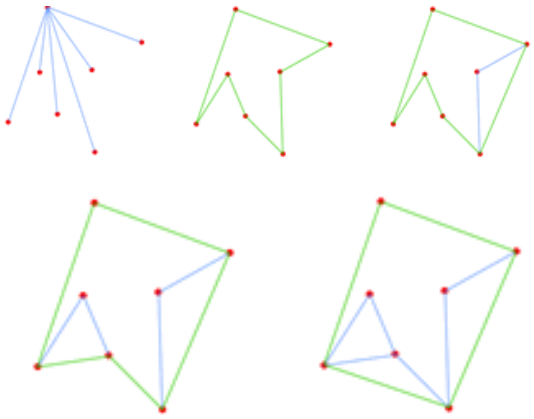

Fig. 3. Convex hull using Graham Scan

IV. Quickhull

The Quickhull algorithm[2] for finding the convex hull is one that is analogous to the well known quicksort algorithm. Like quicksort, it partitions the points into regions that can be solved recursively. It works by creating a line segment between the two extreme points (left and right). Since those two points are guaranteed to be a a part of the convex hull, we can insert them into the vector. We can now split it into two parts, the top half of the line and bottom half of the line. For each half, we find the point that is the farthest distance away from the line. This is like the pivot point in quicksort, which we partition around. Again, this point is guaranteed to be in the convex hull. If we join a triangle between the two extreme points and the farthest point, it can be observed that all the parts inside the triangle are not going to be in the convex hull. Hence, we can remove them from consideration. For the other two parts,

Algorithm 2 Graham_scan(input)

```
0: procedure GRAHAM(Q)
0:     size ← size of input
0:     Find p0 in Q with minimum y-coordinate (and mini-
       mum x-coordinate if there are ties).
0:     Sort the remaining points of Q thatis, Q{p0} by polar
       angle in counterclockwise order with respect to p0.
0:     TOP[S]=0
0:     PUSH(p0,S)
0:     PUSH(p1,S)
0:     PUSH(p2,S)
1: for i ∈ {3, . . . , n} do
2:     while orientationNEXT_TO_TOP[S], TOP[S], pi
       is counterclockwise do
2:         POP(S)
3:     end while
3:     PUSH(pi,S)
4: end for
   return S
```

i.e. those on either side of the triangle pointing outwards, we can solve those as separate sub problems.

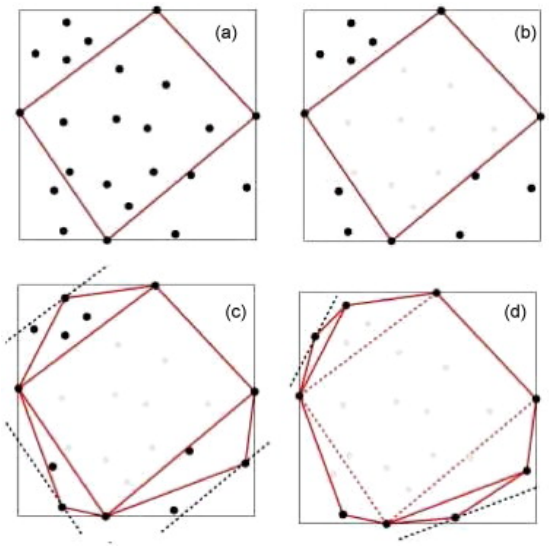

Fig. 4. Construction of convex hull using quickhull algorithm

Algorithm 3 Quickhull(input)

```
0: procedure QUICKHULL_RECUR(INPUT,LEFT,RIGHT)
0:   //Left and right are the leftmost and rightmost points
     for that input.
0:   Find the point 'c' which is at maximum distance from
     line left→right, taking all the points to the right when you
     are at point left and looking at point right.
0:   Insert c into the hull
0:   Using the point 'c', partition the input into those to the
     right of the line left→c. Insert these into a1. And to right
     of the line c→right. Insert these into a2.
     QUICKHULL_RECUR(a1,left,c)
     QUICKHULL_RECUR(a2,c,right)
```

The time complexity analysis of quickhull is similar to that of quicksort. In the worst case, i.e. a circle, where no points are inside the triangle to be discarded, the time complexity is $O(n^2)$. Since the recurring functions takes O(n), it has to process every point and there are n points in the convex hull, it can be seen that it is $O(n^2)$. For an average case, i.e. points are distributed evenly, then similarly to quicksort, the complexity is O(nlogn). In a realistic scenario, i.e. where the number of vertices of the convex hull is much smaller than input (as can be seen in Figure 7), the running time will be O(n+mlogm). O(n) for the first few iterations, where n comes into picture and mlogm for the later stages, which is similar to the average case. As m is much smaller than n, the algorithm is O(n).

V. Jarvis March

The jarvis march algorithm is one that finds the vertices of the convex hull starting from the leftmost point and proceeding in the counter-clockwise direction. It can be seen that the point which is counter-clockwise with respect to every other point will be in the convex hull. i.e. a line from the previous convex hull point point, current point and every other point is counter clockwise. Although all convex hulls algorithms do not work as well with lots of vertices in the convex hull, jarvis march is directly dependent on the number of vertices on the convex hull. The complexity is O(nh), where h is the number of convex hull vertices. For each vertex of the convex hull, an O(n) loop is run to find out the counter-clockwise point.

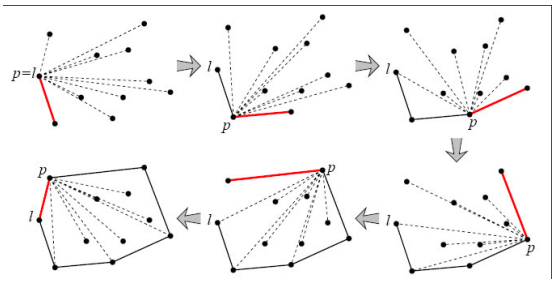

Fig. 5. Construction of convex hull using jarvis march algorithm

Algorithm 4 JarvisMarch(input)

```
0: procedure JARVIS(INPUT)
0:   //left is the leftmost point of input
0:   first_point ← left
1: repeat
1:     Insert first_point into output
1:     third_point ← first_point+1
2:   for i :← 0 →n-1 do
2:       if CCW(first_point,i,third_point) then
2:         third_point ← i
2:       first_point ← third_point
3: until first_point ≠left
```

VI. Analysis

While the overall trend of the time the algorithm takes is almost the same but the width in the difference changes. Overall, the divide and conquer algorithm has the worst asymptotic time complexity followed by Graham Scan, Jarvis march. Quickhull has the best asymptotic time complexity (for the general case) .

The increase at the end of jarvis hull is expected as the number of vertices of convex hull for the larger ranges are more (as can be seen in figure 7).

Graham scan is constant as the input is sorted, range of the points does not come into picture.

As can be seen from the trend of the graph, as the range increases, the number of points on the convex hull increases, hence, jarvis march is expected to take longer time than graham scan for a larger range.

The decrease for quickhull could possibly be due to more points being removed. When the range was (0,100) for both X and Y, the growth of the divide and conquer and the graham scan was similar but as the range of the points increased, divide

and conquer took longer time since divide and conquer has to sort the points and then it has to solve the subproblems.

All the graph's x axis values are the number of input points. The number of input points are 10, 100, 200, 300, 400, 500, 1000, 2000, 5000, 10000, 20000, 50000, 100000, 200000 and 1000000. To view the difference between the points, the range of x was change to 0 to 15.

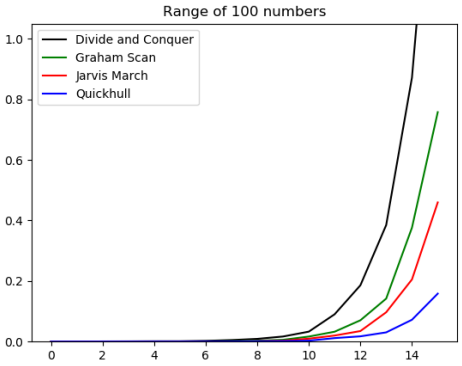

Fig. 6. Asymptotic time complexity when the range for the input was (0,100)

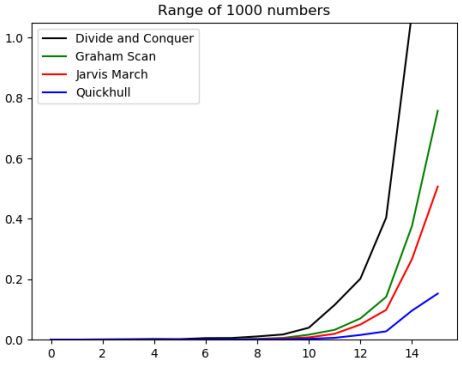

Fig. 7. Asymptotic time complexity when the range for the input was (0,1000)

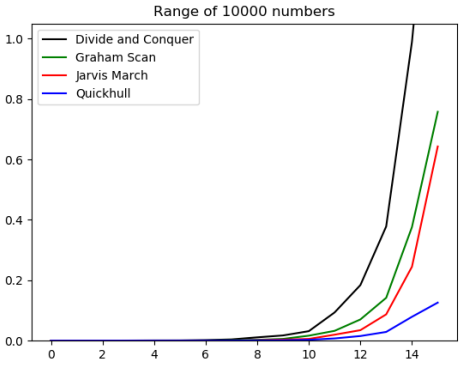

Fig. 8. Asymptotic time complexity when the range for the input was (0,10000)

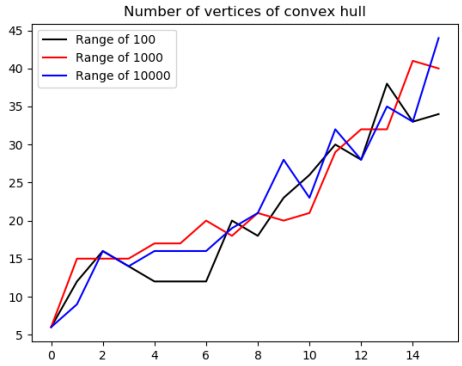

Fig. 9. Asymptotic time complexity when the range for the input was (0,100)

REFERENCES

[1] Suneeta Ramaswami, "Convex Hulls: Complexity and Applications (a Survey)", . December 1993.

[2] Greenfield,, Jonathan Scott, "A Proof for a QuickHull Algorithm"(1990).Electrical Engineering and Computer Science Technical Reports.65.