Comparison of algorithms for solving convex hull

Ajinkya Shinde

CS 584/ 684 – Algorithm Design and Analysis

Spring Term, 2019

Introduction

The purpose of this project is to compare two efficient algorithms for solving the convex hull problem. For comparison, we evaluate the algorithms on a variety of input types like random scatter, circle. The algorithms which we evaluate for comparison are Jarvis March and Graham Scan. We perform different tests on the variety of inputs mentioned above and then evaluate the efficiency of the two algorithms. Below is the overview of convex hull problem

Convex Hull Overview

Convex Hull of a set Q of points in the Euclidean space is the smallest convex polygon P that contains Q such that each point in Q is either on the boundary of polygon P or in its interior. In this project, the set Q of points lies within a finite set. Solving convex hull is a type of computational geometry problem. Convex hull problem can also be visualized as given the points in Q ;we can think each point as being a nail sticking out from a board. The convex hull is then the shape formed by a tight rubber band that surrounds all the nails. Below is the convex hull for random set of points generated by Graham scan

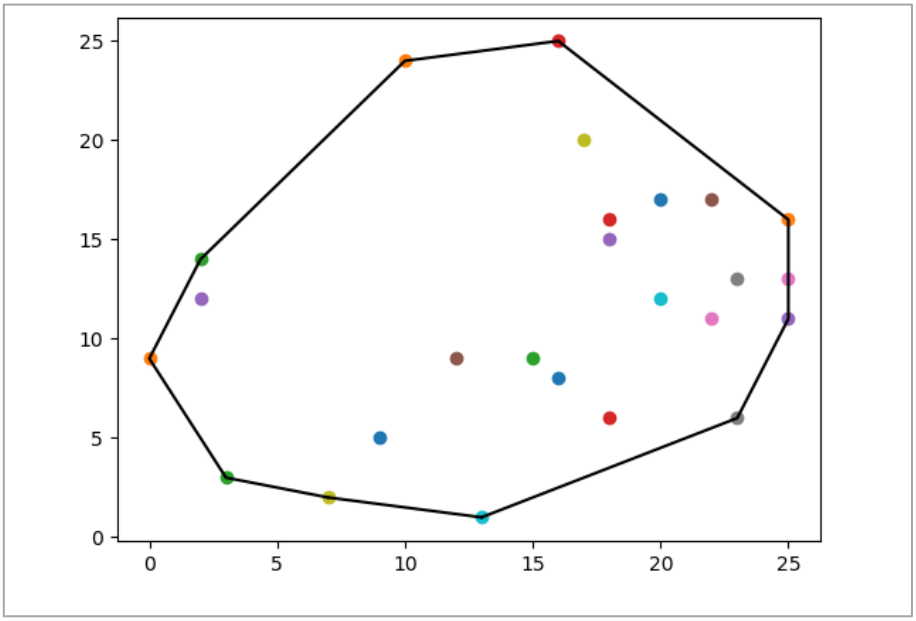

Figure 1: Convex hull of a finite set of points – for random points N= 25 (Graham Scan)

Motivation behind project

Creating art is one of the hobbies that I pursue. Being a computer science major, one of the areas that excites me is the algorithmic art generation. Since, a lot of computational geometry problems like Voronoi diagrams, fractals, convex hull involve using algorithms to create art, it is one of the reasons I decided to pursue this project.

Algorithms Overview

Graham Scan Algorithm - This algorithm[1] solves the convex hull problem by maintaining a stack S of candidate points. It is named after Ronald Graham, who published the algorithm in 1972. Briefly, the algorithm pushes each point of the input set Q of points onto the stack at one time and pops the points that is not a possible vertex of convex hull. Below is the pseudo-code for graham scan algorithm

```
GRAHAM-SCAN(Q)

(1)  let p0 be the point in Q with the minimum y-coordinate,
       or the leftmost such point in case of a tie

(2)  let  p1, p2, . . . , pm  be the remaining points in Q,
       sorted by polar angle in counterclockwise order around p0
       (if more than point has the same angle, remove all but
       the one that is farthest from p0)

(3)        let S be an empty stack

(4)        PUSH(p0,S)
(5)        PUSH(p1,S)
(6)        PUSH(p2,S)

(7)        for i = 3 to m
(8)           while the angle formed by points NEXT-TO-TOP(S),  TOP(S),
                  and pi makes a nonleft turn
(9)              POP(S)
(10)         PUSH(pi,S)
(11)    return S
```

Figure 2: Graham Scan Pseudo-Code

Explanation of Pseudo-code

The above procedure GRAHAM-SCAN(Q) takes an input set Q of points. Line (1) first finds point with minimum y co-ordinate and in case of ties choose the point with minimum x co-ordinate or the leftmost

point which is labelled as p0 . The point p0 is the vertex of convex hull since it has the lowest x and y co-ordinate. Line (2) then sorts the remaining set of points p1,p2 till pm(where m is the number of points other than p0) in Q using polar angle with respect to p0. The polar angle is calculated using the cross product of two vectors. For sorting, any sorting-based algorithm with O(n.log n) is used. We sort the points from smallest to largest polar angle and label them correspondingly from p1, p2 till pm. Line(3)-(6) creates an empty stack S and pushes the points p0, p1 and p2 onto the stack. Lines (7)-(10) then performs the test on each candidate point pi to see if it a possible vertex on convex hull. For this, it calls the procedure NEXT-TO-TOP(S) and TOP(S) which returns the second element and first element from top of the stack respectively. Then, it takes the next candidate point pi and checks if pi is on the left side of line segment formed by points NEXT-TOP(S) and TOP(S). If it is on the left side, we simply push pi on to the stack and do the test for the next candidate point from Q. If it is not on the left side , then the point returned by TOP(S) lies in the interior. We then pop the elements from the stack until the point pi is on the left side of points with respect to TOP(S) and NEXT-TO-TOP(S) points and then push pi onto the stack. We perform this process till the point pi is the same as P0. And then finally return the stack S in Line (11).

Algorithmic Complexity of Graham Scan

The graham scan algorithm's complexity is O(n . log n). This complexity comes from the sorting done at Line (2) and it predominates all the cost of other steps in the graham scan algorithm. For sorting, any sorting-based algorithm like insertion sort, merge sort etc can be used. Although there is a nested loop in Line (7)-(10) and it seems that the algorithm's complexity should be $O(n^2)$, the actual inner loop executes at most once for every outer iteration. Since, we pop the elements from the stack S only when addition of the candidate point pi causes a non-left turn, it is not always that case. Thus, the inner loop executes at most once and the overall complexity of Line (7) – (10) is O(n) which is dominated by sorting cost. The cost of other lines in the algorithm is almost cost and can be ignored. Hence, the overall complexity of graham scan comes to O(n. log n)

Jarvis March Algorithm

This algorithm[2] also computes the convex hull of a set of points – Q by simulating the technique of package or gift wrapping around the convex hull vertices. Visually, we start by taping the end of paper to the lowest end point , in our case it is p0. Then, we pull the paper to the right to make it taut and pull it higher until it touches a point. This point should also be the vertex of the convex hull. We keep this process until we reach the endpoint from where we started i.e. p0. Below is the pseudo-code for the algorithm.

```
JARVIS-MARCH(Q)

(1)   Create a temp sequence hullV = [] to store the vertices of convex hull

(2)   let p0 be the point in Q with the minimum y-coordinate,
        or the leftmost such point in case of a tie. Append
        p0 to the temp sequence H

(3)   Initialize pointonHull to store the candidate vertex point and initialize
        it to p0

(4)         i = 0
(5)         while True
(6)       Store pointonHull at hullV[i]
(7)       Initialize endpoint to p0

(8)       for j = 1 to m
(9)           if endpoint == P0 or p[j] is on the left side from line pi to endpoint
(10)              endpoint = p[j]

(11)          pointonHull = p[j] i.e. is the next vertex of convex hull

(12)          i = i + 1

(13)  until endpoint == P0
(14)  return hullV[]
```

Figure 3: Jarvis-March Pseudo Code

Explanation of Pseudo-code

The above procedure JARVIS-MARCH(Q) takes an input set of points Q and returns the convex hull stored in temporary sequence/ array hullV[] which is created at Line (1) . Similarly, to graham scan in Line(2), we let p0 to be the minimum point which is leftmost among all the points. At Line(3), we initialize pointOnHull to store a candidate vertex and then using it we perform test on other points. At Line (3), we first set it to point p0, the test then checks for rest points p[j] if point[j] is the right-most point with respect to other points in set Q. If point p[j] is the leftmost point then it must be the vertex of convex hull. Thus, for every point p[j], we check it with all the points from p[j-1] till rest of points in Q. At Line(11), we then take the leftmost point p[j] and then find the next potential vertex of the convex hull in the next iteration. We also store the point p[j] at start-of next iteration in the hullV[] at index i+1 where i is the index from previous iteration.

Algorithmic Complexity of Jarvis March

We perform the check on point p[j] with respect to all other points in the set Q. Thus, if there are h number of hull vertices of the convex polygon, we check if there are rightmost with respect to all other points or in other words, all the points other than p[j] are on the left side. Since there are n vertices in

the set Q, the time complexity of Jarvis March algorithm is O(n*h) where n is the number of points in the set Q and h is the number of hull vertices of convex polygon. In other words, for every 'h' operation, we perform O(n) operations resulting in O(n*h) time complexity

Experimental Procedure

To compare the efficiency of these two algorithms, we use three test suites I – III to evaluate the execution time of the algorithms. For the experimental procedure, we take input type as points in which we have a certain percentage of it distributed around the circle boundary and the remaining points are randomly distributed inside the circle. An input type of random distribution without any specific shape is also available but is not considered for any test cases. Based on this input type selection, we look at three different test suites with varying size of input points. The different size of input points for each of the test suites is 5000, 10,000 and 20,000 points. Also, another important objective is to check the which algorithm beats another algorithm.

We know that Jarvis March algorithm's complexity is O(n*h) whereas the complexity of Graham Scan is O(n*log n). The test suites for which percentage of input size does the Jarvis March performs better than the Graham Scan or in other words, we analyse the best case for Jarvis March algorithm.

Data Generation Process

Each of the algorithm starts with menu-styled input-based system. For the choice of 3. Circle with min. hull points, the user is asked to enter the size of input i.e. number of input points. Then, the percentage of those points which should lie on the hull. After this, the radius of the circle and the co-ordinates of the centre of the circle. Based on this user inputs, the function points_on_circumference_with_per then calculates the total number of points on the circumference of the circle from the percentage input and distributes points around the circle evenly. Then the remaining points are drawn randomly inside the circle.

Implementation and Results

For generating output value that can be compared across the test suite, the size of input i.e. 5000 is kept constant. However, for every execution of algorithm, the percentage of boundary points are varied. Initially, we start with all the points on the boundary of the circle i.e. 100% of the total input points. Then, we decrease the percentage as - 70% , 40%, 10% (large gaps) and 5%,1%,0.5%(slow gaps). The decision to vary the percentage by this amount in terms of large gaps and slow gaps is because as the percentage of hull points decrease, Jarvis March will perform better than Graham Scan. Thus, for

percentage less than 5%, it was observed that Jarvis March took less time as compared to Graham Scan. The results for each execution are saved as a .png plots and the execution time along with user inputs is exported to csv file. For comparison, the results in the csv are plotted in Tableau to compare the algorithm's efficiency.

Test Suite I – Implementation and Results

Here, the size of input points is 5000. Below sample plot shows the output for the Jarvis March algorithm for execution with inputs specified in the title

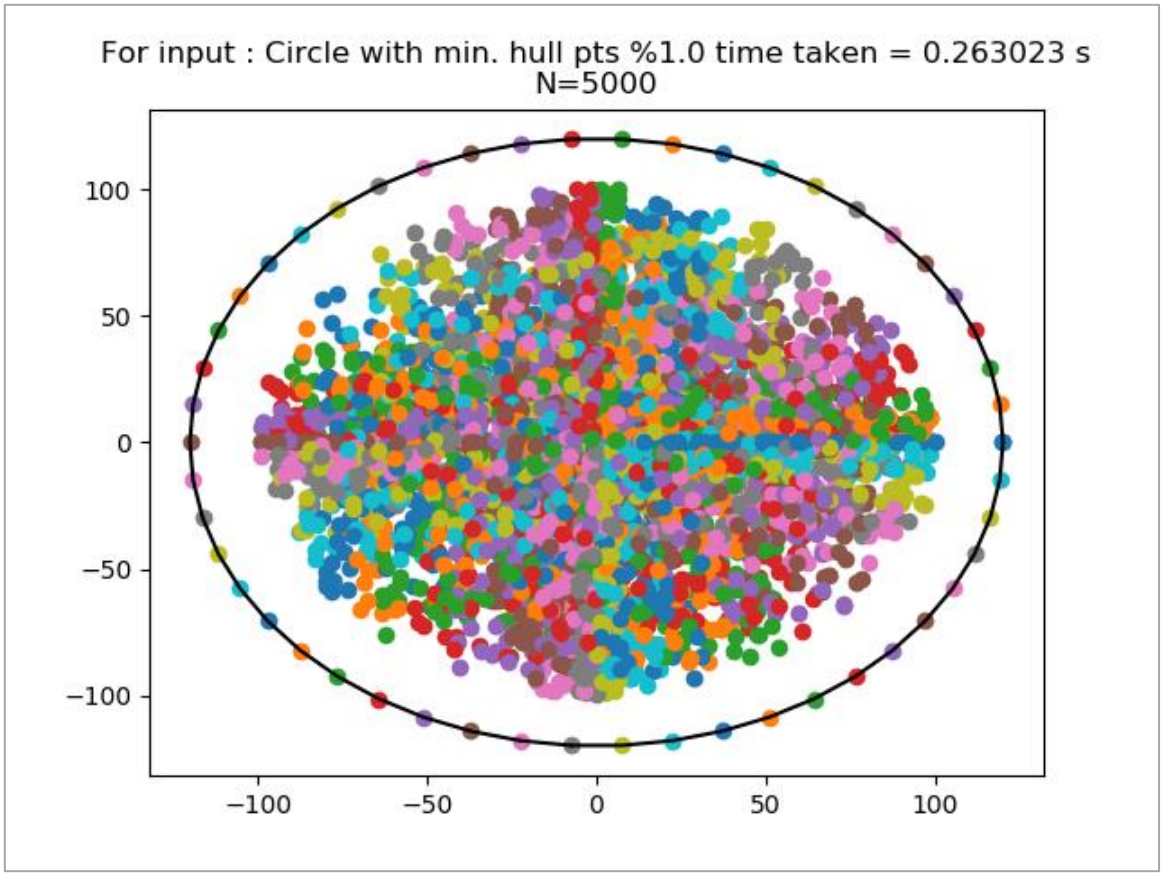

Figure 4: Jarvis March – Circle with min hull points as 1% for N = 5000

Similarly, we run the same input size and percentage input for Graham Scan. Below is the output for Graham Scan algorithm.

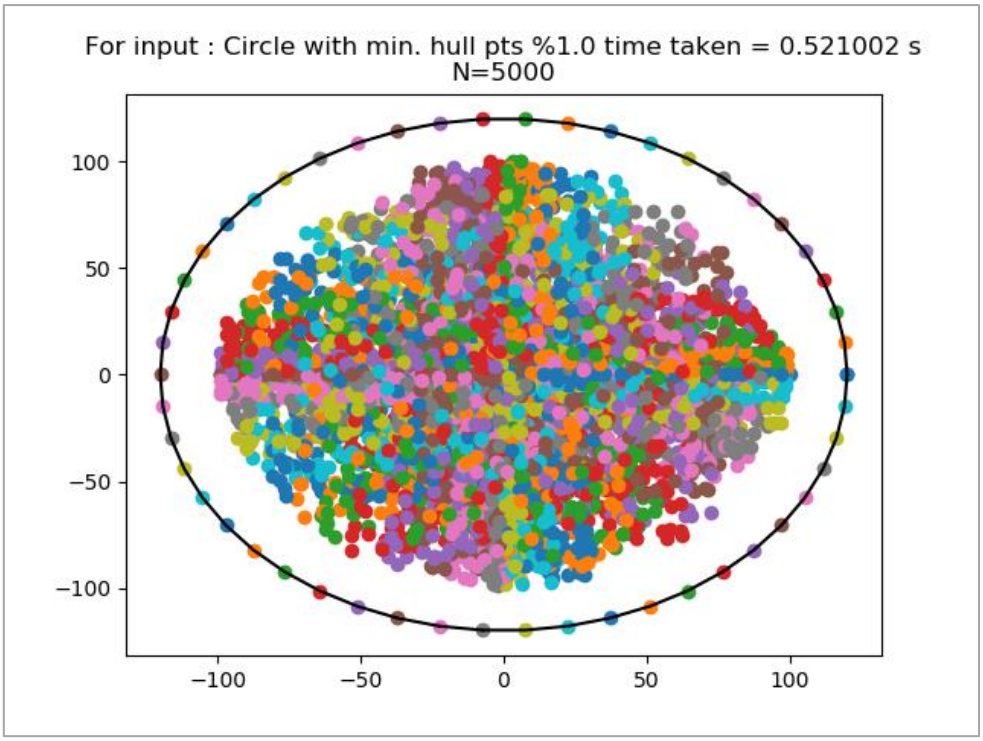

Figure 5: Graham Scan – Circle with min hull points as 1% for N= 5000

Similarly, we obtain the results for rest of the percentage of points for same input size – 5000. The execution time of this test suite is exported in csv. Below plot shows the comparison of algorithm's execution time

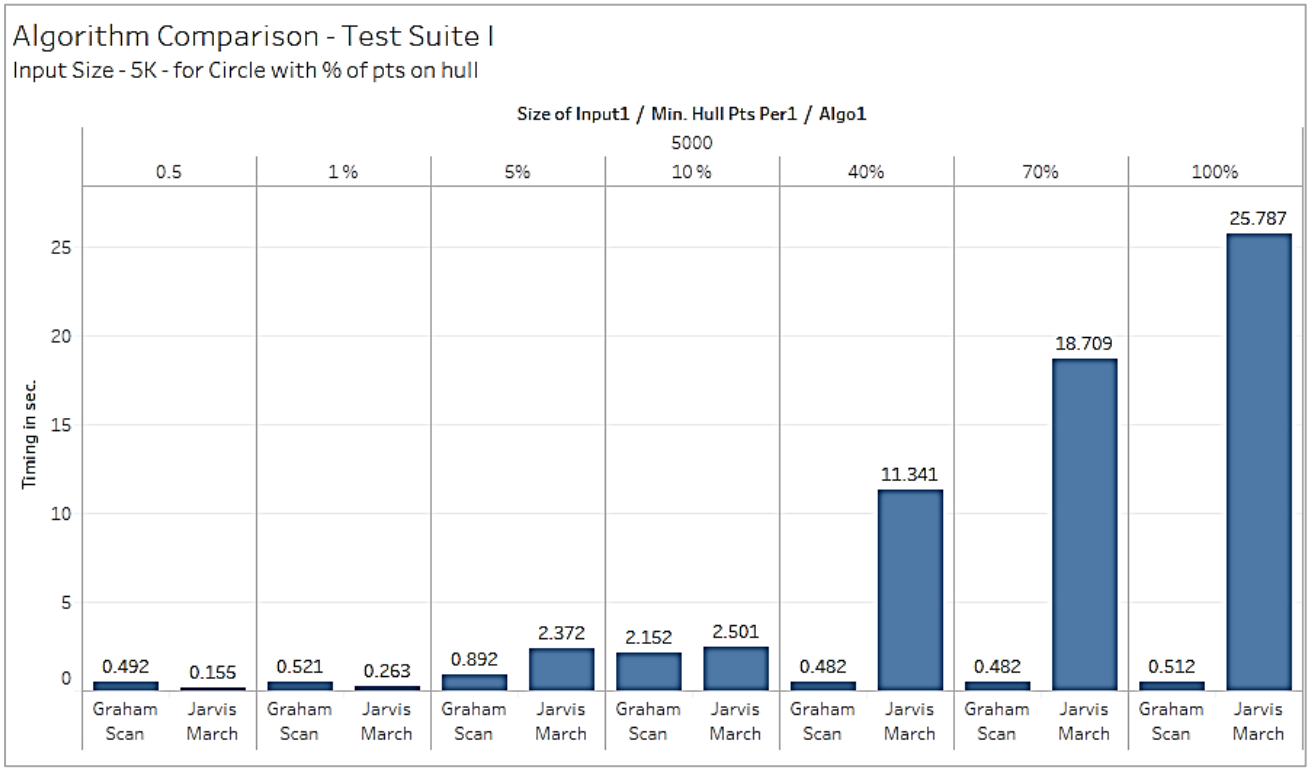

Figure 6: Algorithm comparison – Test Suite I

Test Suite II – Implementation and Results

For this test suite, we keep the size of input points as 10K. Below figures 7.) and 8.) show the output for Jarvis March and Graham Scan algorithms when the percentage is set to 1%

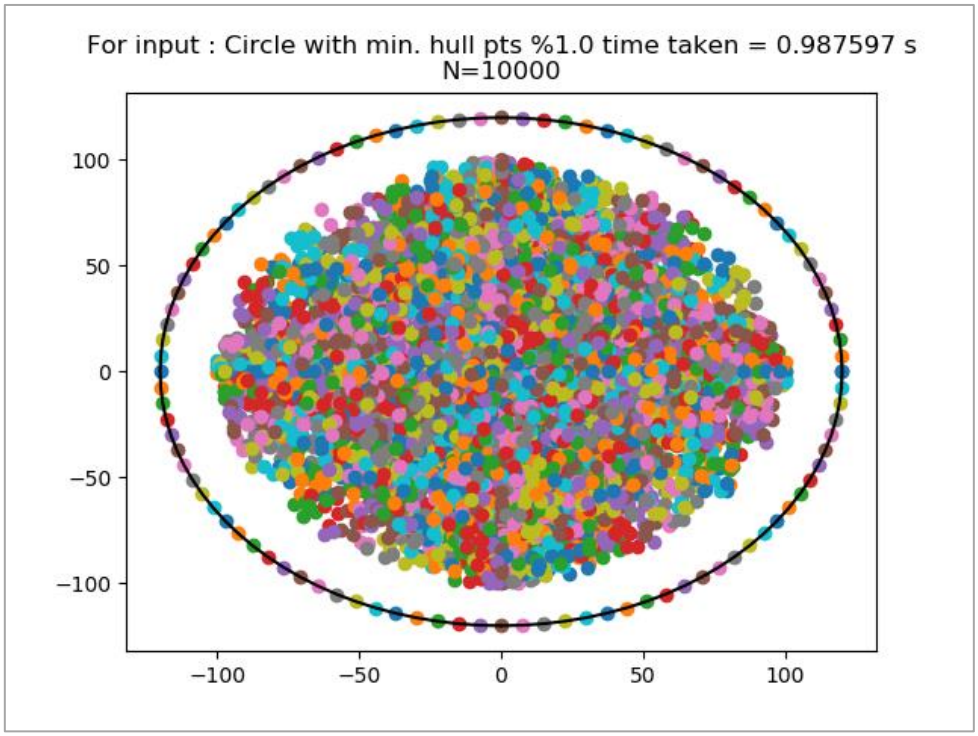

Figure 7: Jarvis March – Circle with min hull points as 1% for N = 10000

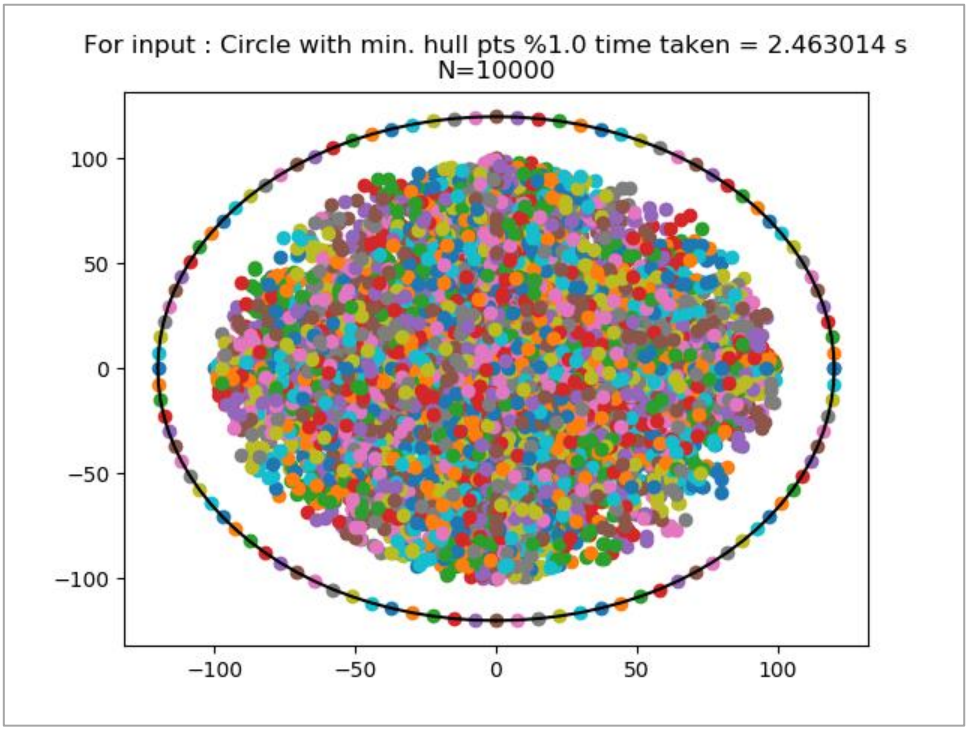

Figure 8: Graham Scan – Circle with min hull points as 1% for N = 10000

Below is the algorithm comparison plot for Test Suite II

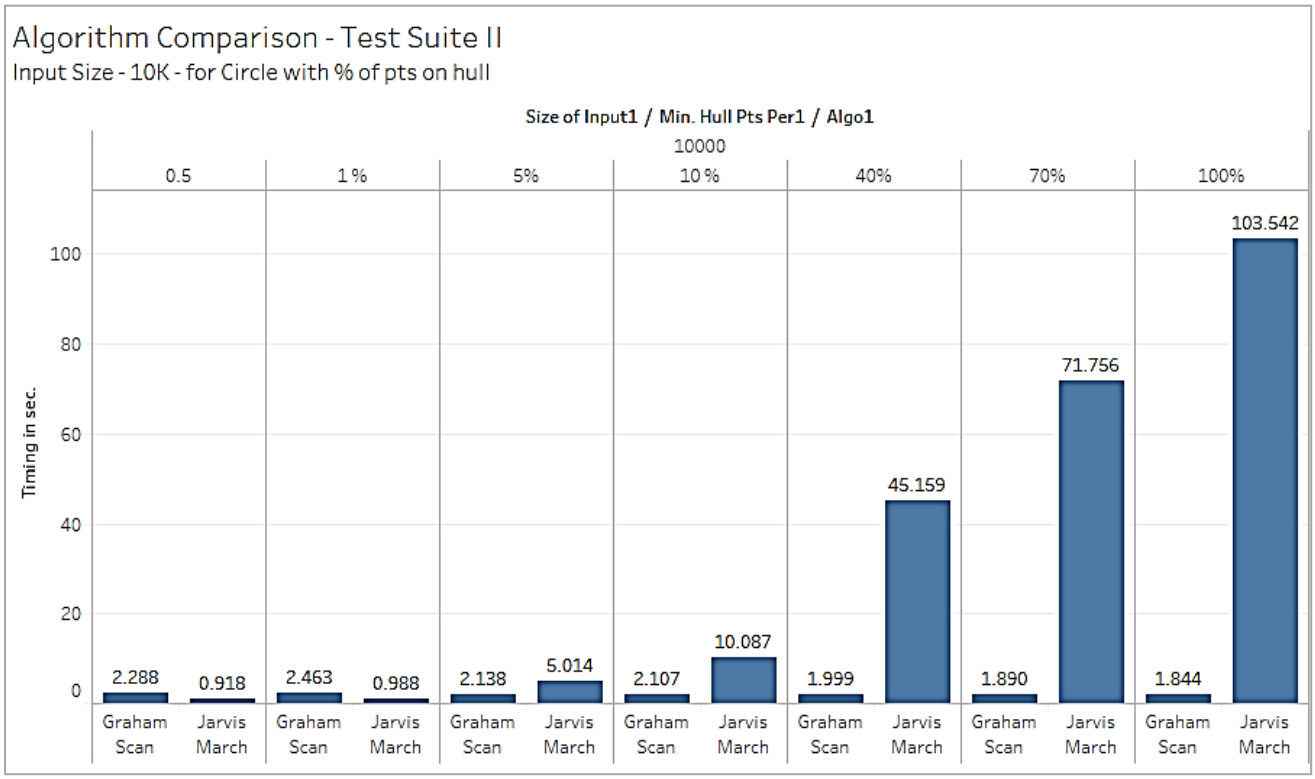

Figure 9: Algorithm comparison – Test Suite II

Test Suite III – Implementation and Results

For this test suite, we keep the size of input points as 20K. Below figures 10.) and 11.) show the output for Jarvis March and Graham Scan algorithms when the percentage is set to 1%

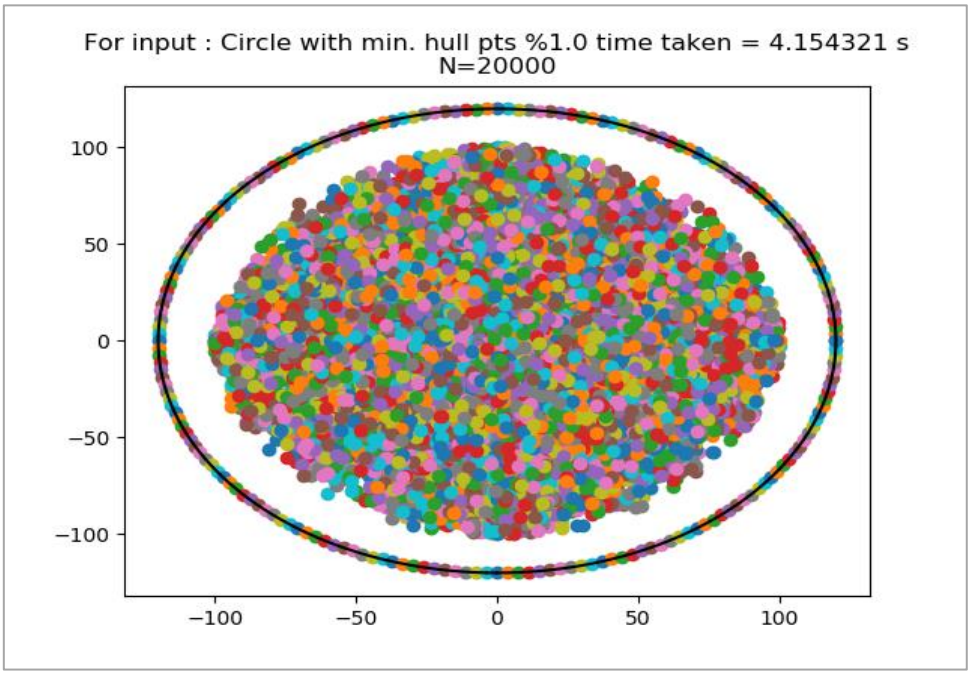

Figure 10: Jarvis March – Circle with min hull points as 1% for N = 20000

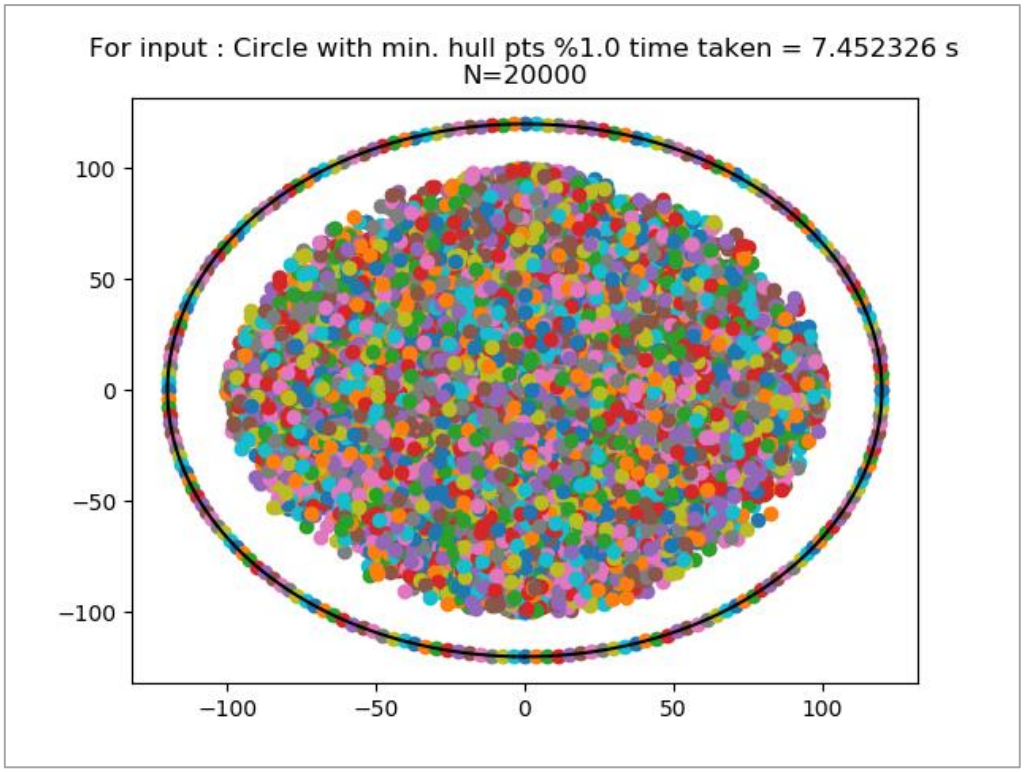

Figure 11: Graham Scan– Circle with min hull points as 1% for N = 20000

Below is the algorithm comparison plot for Test Suite III

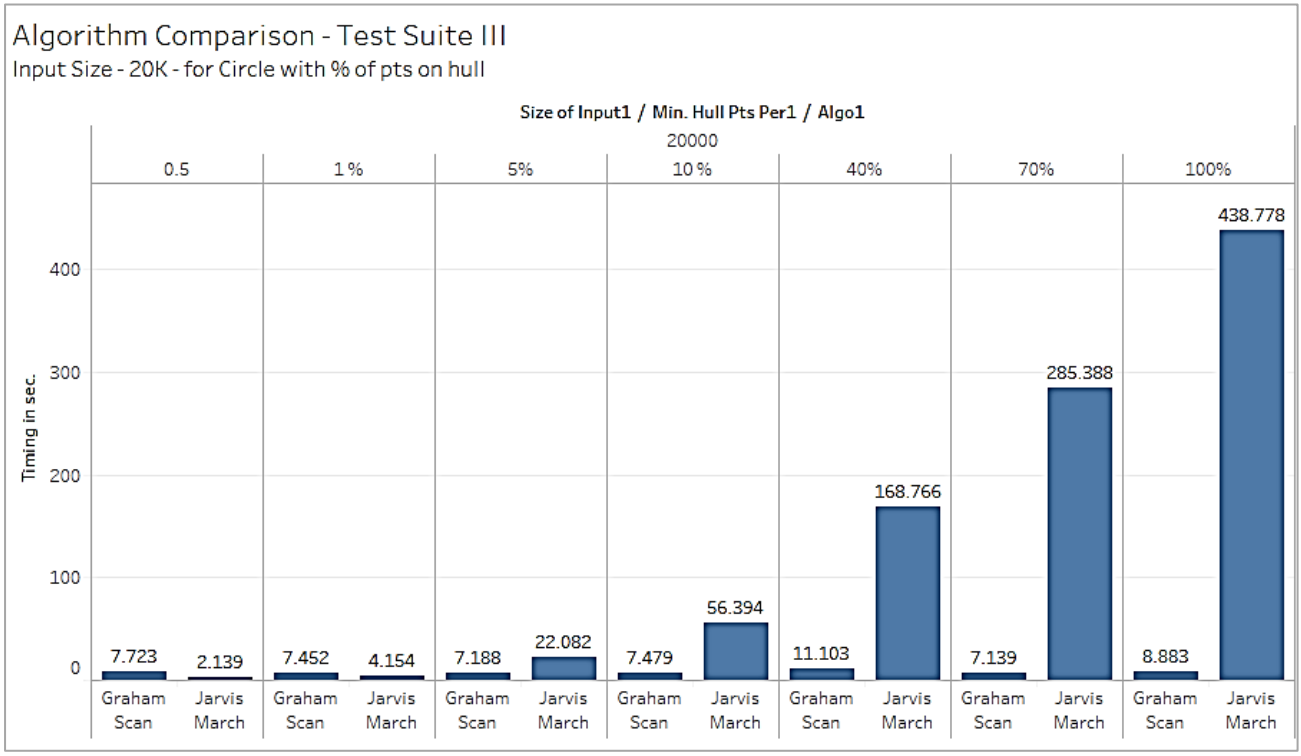

Figure 12: Algorithm comparison – Test Suite III

Interpreting Results

As per the test suite results, it can be seen after 5%, the performance of Jarvis March improves for 1% and 0.5%. The Graham Scan performance remains consistent throughout all the experiments. Jarvis March being an output sensitive , the results are exponential as we vary the percentage of points. Theoretically , for smaller values of h , Jarvis March beats Graham Scan. The experiment results also seem to align with the theoretical expectation. However, large test cases need to be obtained for smaller values of h to see the performance of Jarvis March i.e. the lower bound for Jarvis March algorithm

Limitations

A limitation of the test suite was the time taken by the Jarvis March algorithm to generate the results, specifically the case where all the points are on the hull. In addition to this, the time taken to generate the plots was large. As compared to the theoretical inference of Graham Scan[3], my implementation is faster as a result of using optimized python libraries. Also, its pointed in the article published[4] , that using trigonometric functions for polar angle co-ordinates decreases the speed of Graham Scan greatly.

Conclusion

The percentage of points on the hull was too specific too evaluate the best case for Jarvis March. Determining the exact value for the minimum hull points as a percentage needs rigorous test case. It would be interesting to see the exact percentage where the Jarvis March perform better than Graham Scan. Also, the test suites for really very large inputs like 10^5 , 10^6 need to be checked. However, due to large time for result generation and then reconciliation of it, it became difficult to extend the test suites to really very large inputs. The areas for improvements would be extending test suites for large input size and handling memory for corruption of test results if any.

References

[1] Graham, R.L. (1972). "An Efficient Algorithm for Determining the Convex Hull of a Finite Planar Set" - Information Processing Letters. https://doi.org/10.1016%2F0020-0190%2872%2990045-2

[2] Jarvis, R. A. (1973). "On the identification of the convex hull of a finite set of points in the plane". Information Processing Letters. https://doi:10.1016/0020-0190(73)90020-3

[3] Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L.; Stein, Clifford (2001) [1990]. "33.3: Finding the convex hull". Introduction to Algorithms (2nd ed.). MIT Press and McGraw-Hill. pp. 955–956. ISBN 0-262-03293-7.

[4] R.V. Chadnov ; A.V. Skvortsov [2005]“Convex hull algorithms review”. Proceedings. The 8th Russian-Korean International Symposium on Science and Technology, 2004. KORUS 2004.