CS-GY 6703 Computational Geometry

Dongzi Qu (dq394)

Project Report (Chan's ALGO)

Abstract:

In this final project, I learned a new convex hull algorithm (Chan's Algorithm) from the original paper and then implement this algorithm using `python` and several useful libraries. Furthermore, I did some analysis based on the performance of several different algorithms. Last but not the least, I also built a simple demo to show how this algorithm work in detail.

Introduction:

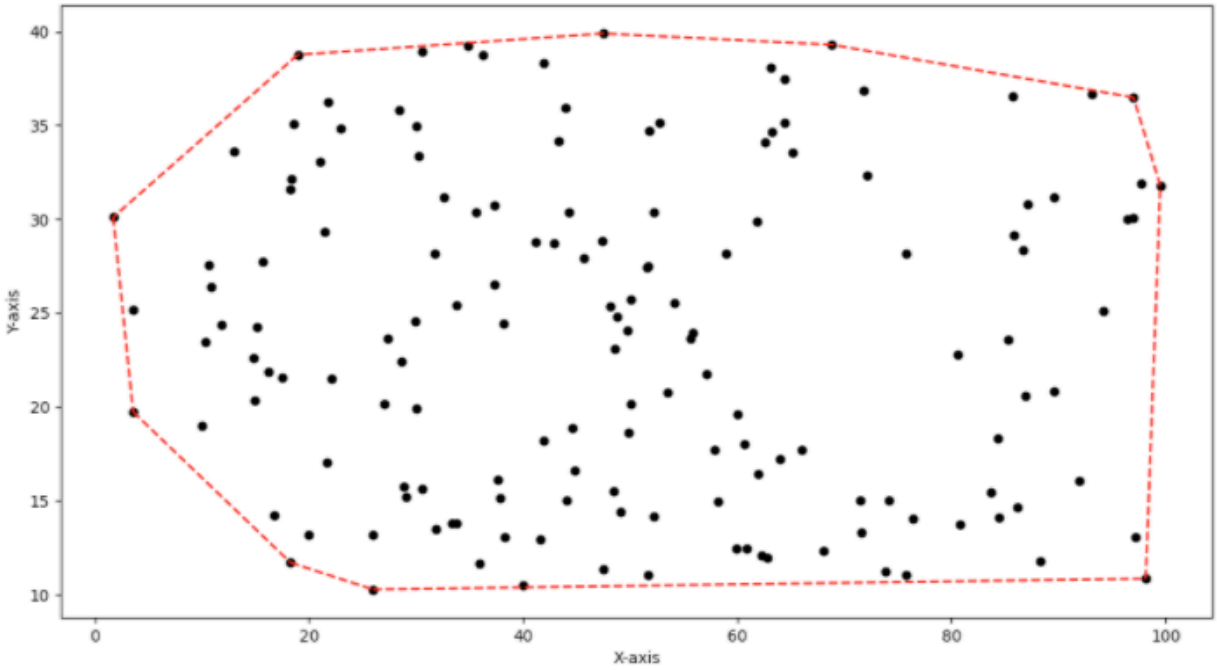
What is Chan's algorithm:

- Chan's algorithm is an output-sensitive algorithm that constructs the convex hull of a set of n points in worst-case optimal $O(n \log h)$ time and $O(n)$ space, where h denotes the number of vertices of the convex hull.

Basically, "output-sensitive" means the performance of Chan's ALGO depends on the final h. Based on the original paper, in order to achieve such time/space complexity, it combines two simpler convex hull algorithms, which are Jarvis's March (Gift Wrapping) and Graham's Scan.

The reason that why we need Chan's algorithm rather than these two simpler algorithms:

- Obviously, it's way better than Jarvis's March, which takes $O(nh)$ time to compute the hull. Because for each p_k on the convex hull, we need to compute the slopes between it and all the other $n-1$ points.
- As for Graham's Scan, if we are given a set of n points where most points are inside of the convex hull, then $O(n \log h)$ will be much smaller than $O(n \log n)$ if n is very large. In that case, we don't want to waste a lot of time on sorting the entire point set. (See the below image where $n = 150, h = 10$)



Algorithm:

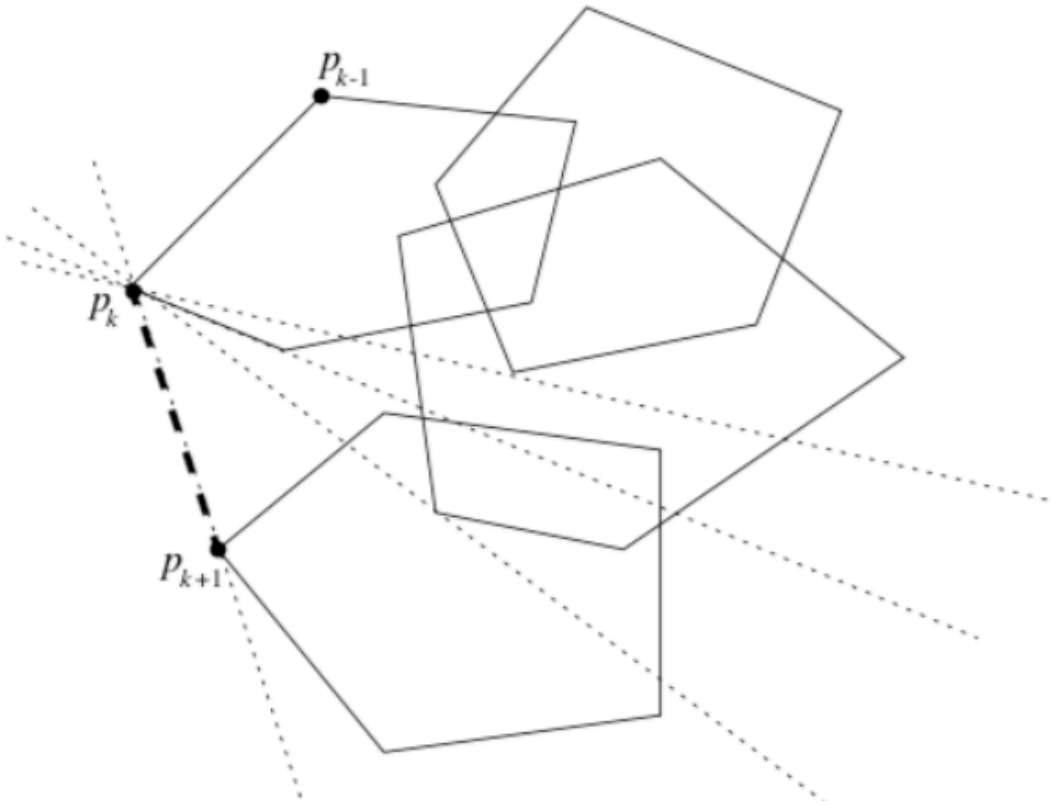
First of all, the input if a set of n points (P) in general position, where there are no colinear and two points sharing the same x-coord. The below algorithm is **almost** what we need (only some small modifications needed to change it to Chan's ALGO):

Basically, there are two main steps: Partition and Wrapping:

- Partition Step:
  - We first choose a parameter m from 1 and n and then partition P into n/m sub-groups where each group contains at most m data points. (This takes linear time $O(n)$)
  - The, we compute the convex hull of each group in $O(m \log m)$ time by Graham's Scan. This gives us n/m possibly overlapping convex polygons. (Total time: $O(\frac{n}{m} \times m \log m) = O(n \log m)$)
- Wrapping Step:
  - We still start from the leftmost point among all data points as what we did in Gift Wrapping. ($O(n)$)
  - Then, we scan all $O(n/m)$ polygons and compute tangents or supporting lines of the polygons through the current vertex p_k, which is the most recent point we found on the convex hull. Here, we can apply **binary search** on tangent finding, which takes only $O(\frac{n}{m} \times \log m)$ time for each p_k.

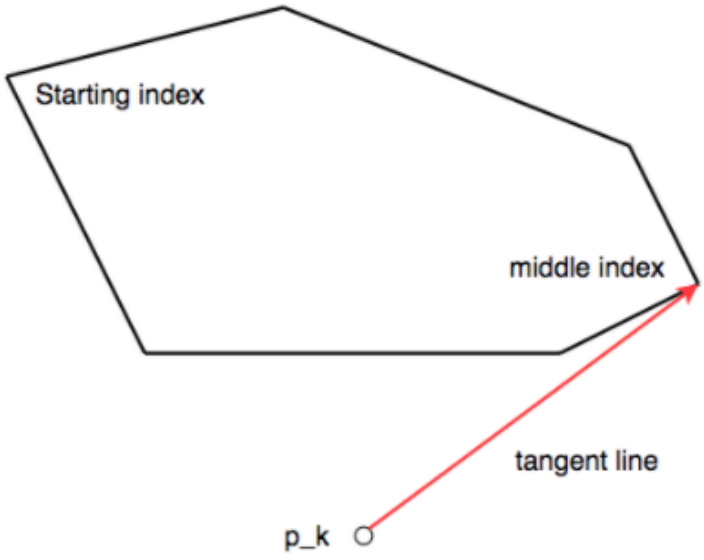
Here I use two images to illustrate these two steps more clearly:

- In the first one, the four convex polygons corresponds to what we get after partion step. Then, the four dash lines are four tangent (supporting) lines drawn from p_k to all these four polygons. After that, we choose the most suitable (largest angle) as the next candidate point p_{k+1}.



- The second image shows how to find tangent line using binary search. As we can see, the red line is the tangent line drawn from p_k to this sub-polygon. The vertices are stored in counter-clockwise order by Graham's Scan and we can check p_k and the point with middle index on this polygon to know which side we should go in order to find the tangent point.

  (*The detailed logic of binary search will be found inside the code*)



Time complexity until now:

In the second part of Wrapping Step, the reason for the time $O(\frac{n}{m} \times \log m)$ is because we need to draw the tangent lines from p_k to all the n/m polygons to make sure which one is the next point p_{k+1}. If we combine all these steps together, then the total time T is:

$$T = O(n) + O(n \log m) + O(h \times \frac{n}{m} \times \log m) = O(n \times (1 + h/m) \times \log m)$$

where the third term comes from we need $O(\frac{n}{m} \times \log m)$ for all h vertices of the convex hull. Notice that, if we replace m with h, then T becomes $O(n \log h)$, which is exactly what we mention in the beginning.

It seems that we have already finished this algorithm. However, this is not the end of Chan's ALGO because we don't know the number of vertices (h) on the hull before running the algorithm. That is, we don't know the specific m we should use on the partition step. So, we can't replace the m with h in the real algorithm complexity analysis.

Based on such condition, what to do next? We need to try/guess different m and make some modifications on wrapping step.

The modifications on the previous algorithm:

- We still scan all n/m polygons and compute tangent lines through the current vertex p_k. However, we **only do** this m times rather than h times. This will change the time of entire algorithm to $O(n \log m)$ because we just replace h in $O(n \times (1 + h/m) \times \log m)$ with m.
- Obviously, this change will lead to another new issue: if the current m is smaller than h, then the convex hull we find is **incomplete**. So, to output the correct hull, we should keep trying different m until it is larger than or equal to h.

So, the remaining question is: how to find this suitable m. There are several possible ways:

- We can iterate from $m = 1, 2, 3, \ldots, h$ naively, but it takes too much time.
- A smarter way from the paper to save time: each time we let $m = \min(n, 2^{2^t})$, where $t = 1$ initially. If such m is not large enough to return a complete convex hull, then we change $t = t + 1$; otherwise, we know that the current m is already greater than or equal to h, so we can return the result.

New time complexity:

In each iteration we run the entire algorithm, we need $O(n \log m) = O(n \times \log(2^{2^t})) = O(n \times 2^t)$. Also, we know that the number of iterations in the loop is $\log \log h$. The reason is once the current m is larger than or equal to h, we can break from the loop. So, if we let $2^{2^t} = h$, then t is $\log \log h$.

To sum up, the new time T should be:

$$T = O(\sum_{t=1}^{\log \log h} n \times 2^t) = O(n \times 2^{(\log \log h)+1}) = O(n \log h)$$

which is exactly we mentioned in the beginning.

Implementation & Result:

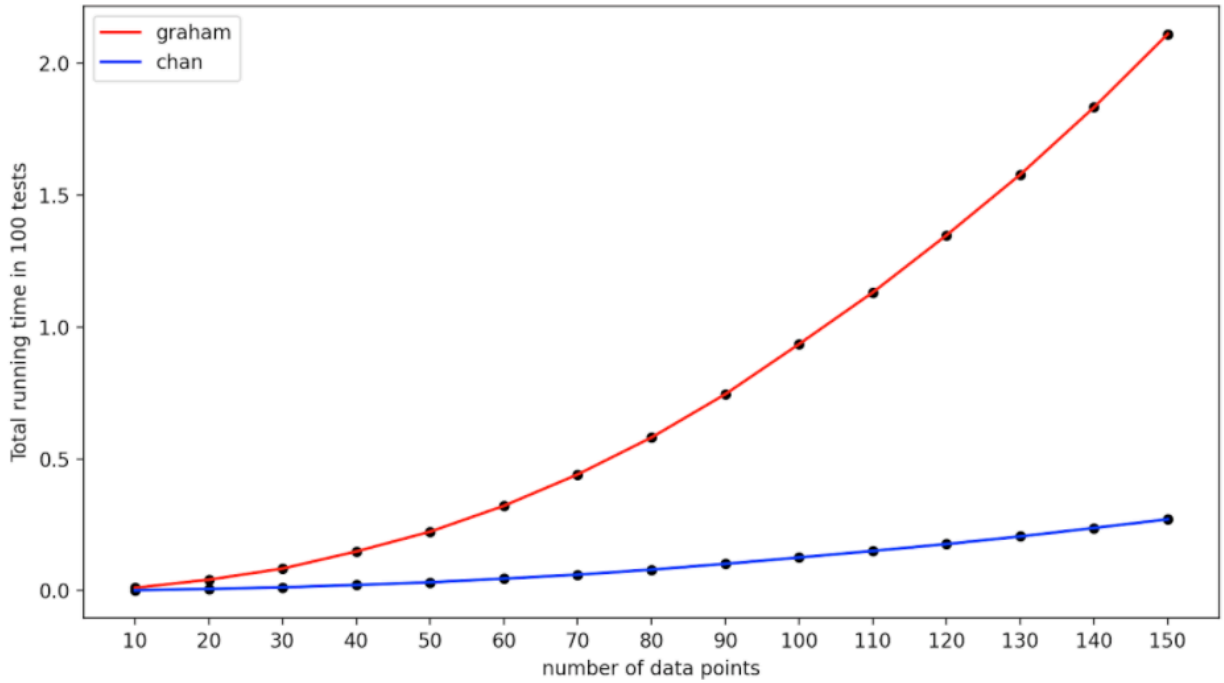
Implementation overview

Based on the previous discussion, I first implement a generator to randomly generate different date points. Then, I implement the Jarvis's March and Graham's Scan respectively using sorting and simple slope trick. Also, I implement a binary search block to find the tangent lines. Finally, I combine all these parts together and try dfferent m starting from 4 (1<<(1<<1)=4).

All the code for algorithm and auxiliary plotting functions will be found in my GitHub personal page. During the course presentation, I have already used the simple demo to show Chan's algorithm step by step, including switch from different m, partition and wrapping, which is also embedding inside my code files.

Some results:

Here, I use the below image to show the performance of Chan's algorithm compared with Graham's Scan. As shown, I run the test with different size of the data set. What's more, to remove the randomness from generation, I run 100 indepent test for each size. So, the x-axis is the size and y-axis is the total running time.



Obviously, as the size of data set is larger (n), the gap of running time between Graham's Scan and Chan's algorithm is larger. This is what I expected because the difference between h and n is larger and n is also increasing at the same time.

My own work statement:

In this project, I use `Python` for all the code part. Furthermore, I took use of the following useful libraries:

- `numpy`: to randomly generate the data points.
- `matplotlib`: all the plotting functions.
- `sys`, `math`, `tqdm`, `time`: auxiliary libraries to make code more clean.

For the entire code, including sub-functions and the first demo I showed in lecture, these are all my own work (annotations included). The second demo I showed in my presentation is from outside resource (Chris Gregory). Also, the slides for the presentation and this report are also totally my own work.

Except as described above, all the work on this project is my own. ***Signed: [Dongzi Qu]***