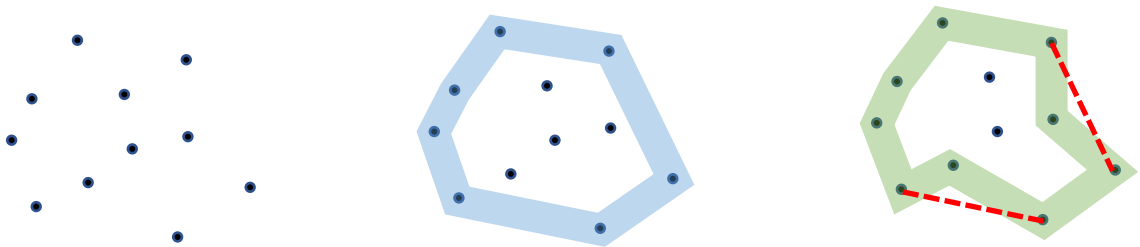
COMP0005 Algorithms
Academic Year 2020-21

Group Coursework: Convex Hull

Released: February 8th, 2021
Submission deadline: March 3rd, 2021 @ 4pm GMT

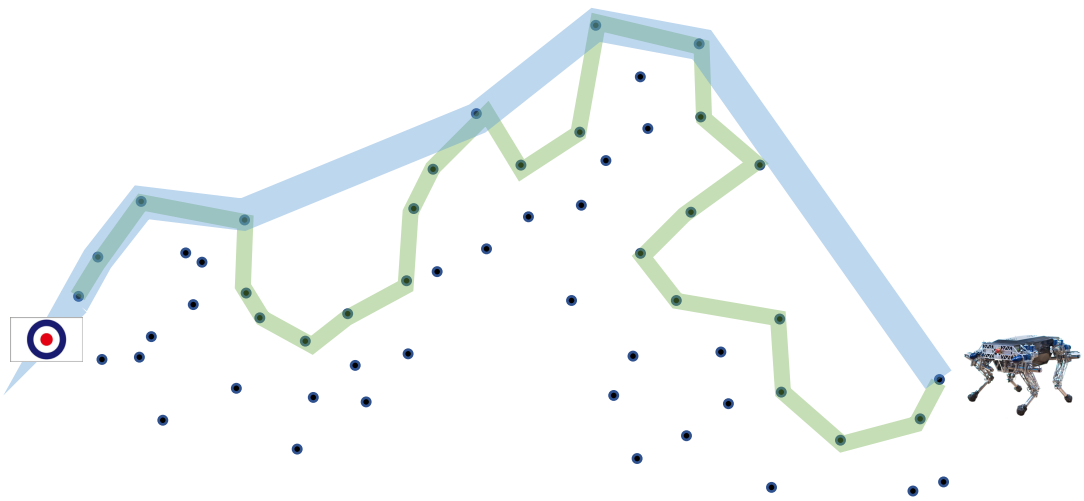
The problem.

Given a set S of N points (x_i, y_i) on a Cartesian plane, a convex hull is defined as the smallest convex polygon that contains all points in S. A polygon is said to be convex if it contains all the line segments connecting any pair of its points. For example, given the 11 points on the left:



the blue-highlighted polygon in the middle is a convex hull, while the green-highlighted polygon on the right is not, since the red-dashed segments are not contained in it.

Being able to efficiently compute the convex hull of a set of points on a plane has fundamental applications in a variety of domains, including Robotics and Autonomous Systems. Imagine, for example, a locomotive robot (depicted below) having to find the quickest path around a number of obstacles, as is the case for emergency/rescue robots having to navigate around a large number of debris when moving in post-disaster (e.g., earthquakes) areas. By following the convex hull (blue-highlighted line below), the robot can reach the target point much faster than if simply following the perimeter (green-highlighted line below).



Many algorithms exist to compute the convex hull of a set of Cartesian points, with varying computational complexity. Two such algorithms are: the Jarvis march algorithm (also known as the Gift wrapping algorithm) [1], and the Graham scan algorithm [2].

The task.

There are three parts to this coursework.

Part 1.

- Study the Jarvis march algorithm (Wikipedia link provided above, as well as reference to the original paper), and implement it in Python 3.
- Design and execute experiments to measure the computational cost (in terms of time) of your implementation, as the number N of points grows. Consider both the average case scenario (e.g., points randomly placed in the space), and the worst-case scenario for the Jarvis march algorithm.

Part 2.

- Study the Graham scan algorithm (Wikipedia link provided above, as well as reference to the original paper if interested), and implement it in Python 3.
- Design and execute experiments to measure the computational cost (in terms of time) of your implementation, as the number N of points grows. Consider both the average case scenario (e.g., points randomly placed in the space), and the worst-case scenario for the Graham scan algorithm.

Part 3.

- Design and implement heuristics to improve on the original Graham scan algorithm.
- Design and execute experiments to measure the computational cost of the modified Graham scan algorithm as the number N of points grows. Compare it to the original version.

Constraints. For this coursework, you are expected to implement your own algorithms and data structures. You are NOT allowed to use (import) python libraries, with the exception of `timeit` and `random` (as explained below). If in doubt about what you can and cannot use, ask in Moodle "Ask a question" forum.

Submission instructions.

This is a group-based coursework and only one submission per group is required. Using the Moodle course website, submit a single zip archive containing:

- A Jupyter notebook `jarvismarch.ipynb` containing:
  - Your python implementation of the Jarvis march algorithm.
  - Driver (test) code that first generates a set of N points **randomly placed** on a Cartesian plane (use the python `random` package to generate these points and limit your `x`-scale and `y`-scale to be integers in the range [0, 32767]). Then invoke the Jarvis march algorithm, passing in input these points, and returning as output the subset of points that lie on the convex hull. Repeat for N = 100, 500, 1000, 5000, 10000, 15000, and 20000. Use the python `timeit` library to measure the execution

time of your implementation. Do not include the time to generate test data in your measurement.
    - Driver (test) code that first generates a set of N points placed on a Cartesian plane, representing the **worst-case scenario** for the Jarvis march algorithm. Then invoke the algorithm, passing in input these points. Repeat for N = 100, 500, 1000, 5000, 10000, 15000, and 20000. Use the python `timeit` library to measure the algorithm execution time. Do not include the time to generate test data in your measurement.

- A Jupyter notebook `grahamscan.ipynb` containing:
    - Your python implementation of the Graham scan algorithm.
    - Driver (test) code that first generates a set of N points **randomly placed** on a Cartesian plane (use the python `random` package to generate these points and limit your `x`-scale and `y`-scale to be integers in the range [0, 32767]). Then invoke the Graham scan algorithm, passing in input these points, and returning as output the subset of points that lie on the convex hull. Repeat for N = 100, 500, 1000, 5000, 10000, 15000, and 20000. Use the python `timeit` library to measure the execution time of your implementation. Do not include the time to generate test data in your measurement.
    - Driver (test) code that first generates a set of N points placed on a Cartesian plane, representing the **worst-case scenario** for the Graham scan algorithm. Then invoke the algorithm, passing in input these points. Repeat for N = 100, 500, 1000, 5000, 10000, 15000, and 20000. Use the python `timeit` library to measure the algorithm execution time. Do not include the time to generate test data in your measurement.

- A Jupyter notebook `extendedgrahamscan.ipynb` containing:
    - Your python implementation of a faster, modified version of the original Graham scan algorithm.
    - Driver (test) code that repeatedly executes such algorithm and measures its execution time as N grows. Consider both randomly placed points on a plane and worst-case scenario, for N = 100, 500, 1000, 5000, 10000, 15000, and 20000.

- A PDF document of maximum 4 pages (font style: Ariel, font size: 12pt), where you describe, for each of the algorithms separately, the following key points:
    - The computational complexity achieved, both theoretically (as order-of-growth) and experimentally, for **random input**. Include scatter plots of the previously computed execution times (`x`-axis is N, `y`-axis is the measured execution time).
    - The computational complexity achieved, both theoretically (as order-of-growth) and experimentally, for input representing the **worst-case scenario**. Include scatter plots of the previously computed execution times (`x`-axis is N, `y`-axis is the measured execution time).

  For the extended Graham scan algorithm, also describe the heuristics you have implemented and discuss the improvements achieved with respect to the original version.

You must use the skeleton Jupyter code provided for each of your algorithms.

Submission deadline: Wednesday, March 3rd 2021, 4pm GMT.

Assessment.

This coursework represents 40% of the final mark for COMP0005 Algorithms.

Submissions will be evaluated in terms of the following marking criteria:

- Correctness and readability of your code (e.g., is your implementation of the above algorithms computing the correct convex hull? Is your code readable and well documented?) [20 points]
- Efficiency of your code (e.g., have appropriate data structures and algorithms been chosen? Have effective heuristics been implemented to improve over the original Graham scan algorithm?) [30 points]
- Depth of your analysis (e.g., is the theoretical analysis of the computational complexity of your code correct? Is the analysis of the experimental cost of your code comprehensive? Has the correct worst-case scenario been identified and thoroughly analysed?) [50 points]

Marks for each criterion will be assigned following the UCL CS Grade Descriptor (criteria 4, 5 and 6).

Note: students within the same group should not expect to receive the same mark. Marks will also depend on each student's individual contribution to the work of the group. This will be determined based on active engagement during the weekly TA lab sessions.

Academic Integrity.

UCL has a very clear position about academic integrity – what it is, why it is important, and what happens if you breach it. Make sure you have read UCL position on this matter before attempting this coursework.

References.

[1] Jarvis, R. A. "On the identification of the convex hull of a finite set of points in the plane". Information Processing Letters. 2 (1): 18–21. March 1973. doi:10.1016/0020-0190(73)90020-3.
Wikipedia link: https://en.wikipedia.org/wiki/Gift_wrapping_algorithm

[2] Graham, R.L. "An Efficient Algorithm for Determining the Convex Hull of a Finite Planar Set". Information Processing Letters. 1 (4): 132–133. doi:10.1016/0020-0190(72)90045-2.
Wikipedia link: https://en.wikipedia.org/wiki/Graham_scan