Analysis on the Enclosing Circle Algorithms

Author: Caio Valente (https://github.com/valentecaio/geometry)

Date: 2023-09-05

Introduction

An enclosing circle algorithm, often referred to as the "smallest enclosing circle" or "minimum bounding circle" algorithm, is a computational geometry algorithm used to find the smallest circle that completely encloses a given set of points in a 2D space. The primary goal of an enclosing circle algorithm is to find the center (x, y coordinates) and the radius (r) of the smallest circle that contains all input points. This circle is called the "enclosing circle" or "bounding circle," and it has the property that no point lies outside of it.
The objective of this project is to compare and study several "Enclosing Circle" algorithms. To do so, some of these algorithms were implemented in the Lua language, which shall be a good fit for this type of job.

Dependencies

Since we are dealing with a geometry problem, we need an easy way to plot some simple data, such as points, circles and curves. I couldn't find a library as good as the popular Python library MatPlotLib, so I've developed a small bridge between Lua and Matplotlib, named MatPlotLua. This allowed me to generate or process data in Lua and then visualise it using Matplotlib in Python. As a consequence, this project needs python and matplotlib to run. We also need the lua-cjson Lua package for parsing json data.

```
# pip install matplotlib           # install python deps
# sudo apt-get install lua-cjson   # install lua deps
```

Project Structure

```
.
├── convex_hull/
│   └── convex_hull.lua       -- Contains Convex Hull algorithms that are used by one of
│                                the Enclosing Circle algorithms.
├── enclosing_circle/
│   ├── enclosing_circle.lua  -- Contains the developed Enclosing Circle algorithms.
│   └── main.lua              -- Script to test the developed algorithms, compare them
│                                and analyse their results.
├── matplotlua/               -- Our plot library, a bridge between Lua and matplotlib.
│   ├── example.json
│   ├── main.lua
│   ├── matplotlua.lua        -- Lua module to handle plot data and parse it to JSON.
│   └── matplotlua.py         -- Python script that parses JSON data and plots it.
│
└── utils/
    └── utils.lua             -- Contains useful methods reused by other modules.
```

EnclosingCircle module

The following algorithms were implemented in the EnclosingCircle module:

1. **Dumb**: A naive way to find ANY circle that encloses a given set of the points.
   It calculates the rectangle containing all the points, and returns a circle that encloses this rectangle.
   This method performs in O(n) but its results may be a very bad approximation for the smallest circle.

2. **Brute Force**: O(n^4) way to find the smallest circle that encloses all points.
   It calculates the centre and radius of each potential circle and selects the one with the smallest radius.
   It is simple but has a very bad time complexity.

3. **Heuristic**: O(n) (average) algorithm that aims to find an approximate solution more efficiently. It starts with a small circle containing just one point and iteratively expands it to include additional points until all points are enclosed. This method has a better average-case time complexity than brute force but may not guarantee the smallest circle in all cases.

4. **Welzl**: Welzl's algorithm is a recursive algorithm that efficiently computes the smallest enclosing circle by relying on the minimal enclosing circle of smaller point sets. It starts with a set of points and recursively reduces the problem by removing points that are not on the boundary of the smallest circle. It is widely used in practice since it has an expected time complexity O(n) when the input data is randomly distributed, and assures that the returned circle is the smallest possible. Although the Welzl algorithm is expected to grow linearly, it should have a bigger slope when compared to the Heuristic method. In the worst case, when the input data is ordered, this algorithm should perform with time complexity of O(n^2).

5. **Smolik**: Smolik's algorithm is a relatively recent approach that combines Welzl's algorithm with a pre-processing. It was proposed in 2022, in the paper "Efficient Speed-Up of the Smallest Enclosing Circle Algorithm" available at https://informatica.vu.lt/journal/INFORMATICA/article/1251.
   The algorithm starts by calculating the Convex Hull of the given points, using a linear algorithm proposed by Skala in 2016. Skala's method has a complexity of O(n+k) where k is the size of the Convex Hull (the output). The points that are not part of the convex hull can be discarded, as they cannot be part of the smallest enclosing circle. This reduces the data set from n to k. Then,we shuffle the remaining points, and apply the Welzl Enclosing Circle algorithm on them. Smolik's algorithm is expected to be O(n) in average, as Welzl's, but it should grow with a smaller slope, closer to the Heuristic method.

The module also contains a method named **validateCircle** that checks if all points in the data set are enclosed by a given circle. This method may be used to check the validity of results.

As the first two implementations are too simple, I'll study only the last three.

How to execute tests

We just need to execute the enclosing_circle/main.lua file using the Lua interpreter:

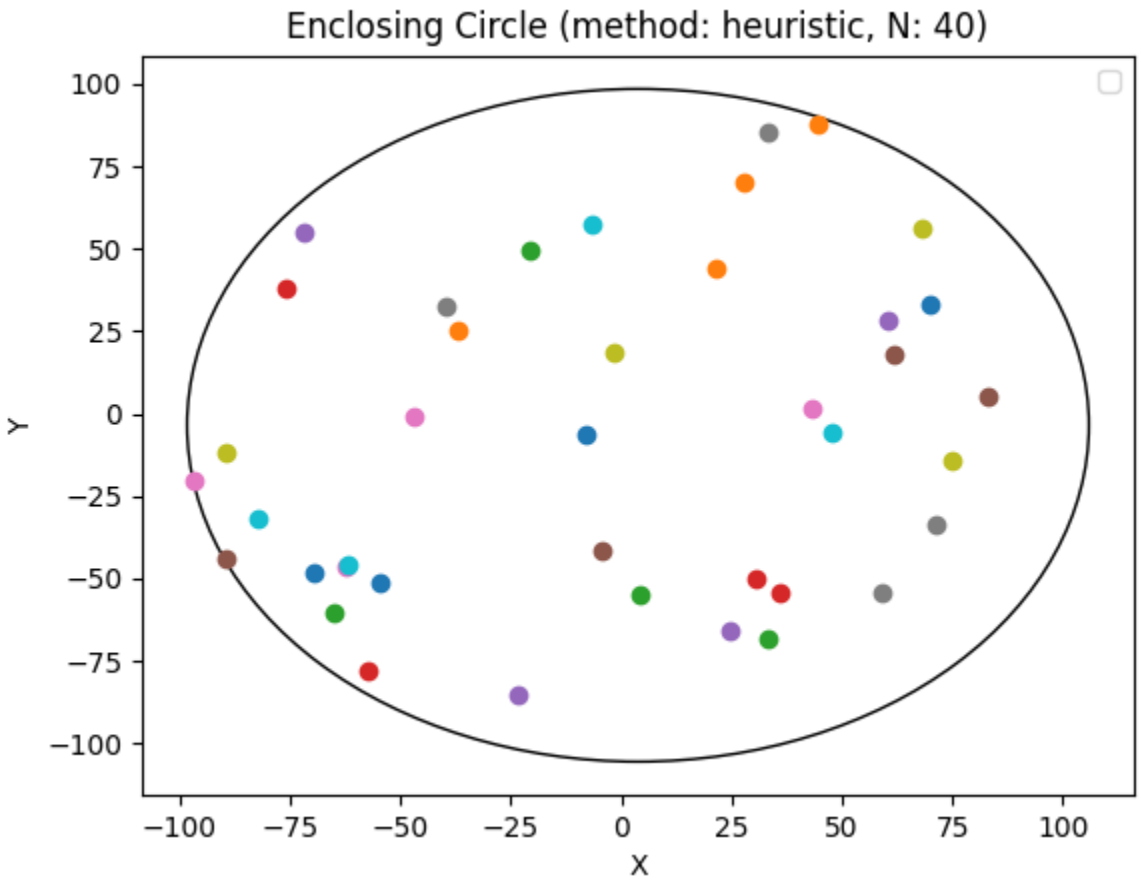
```
$ lua enclosing_circle/main.lua
 Usage: lua enclosing_circle/main.lua <method> [N]
      where N is the number of random points in the dataset (optional, default: 100)
      and <method> is one of [complexity|compare|dumb|heuristic|bruteforce|welzl|smolik]
```

By default, all points are randomly generated inside a circle centred at {0, 0} with radius 100.
This can be changed manually in the script:

```
local base_circle = {x = 0, y = 0, r = 100}
local rand_points = Utils.generateRandomPointsInCircle(N, base_circle)
```

Heuristic method

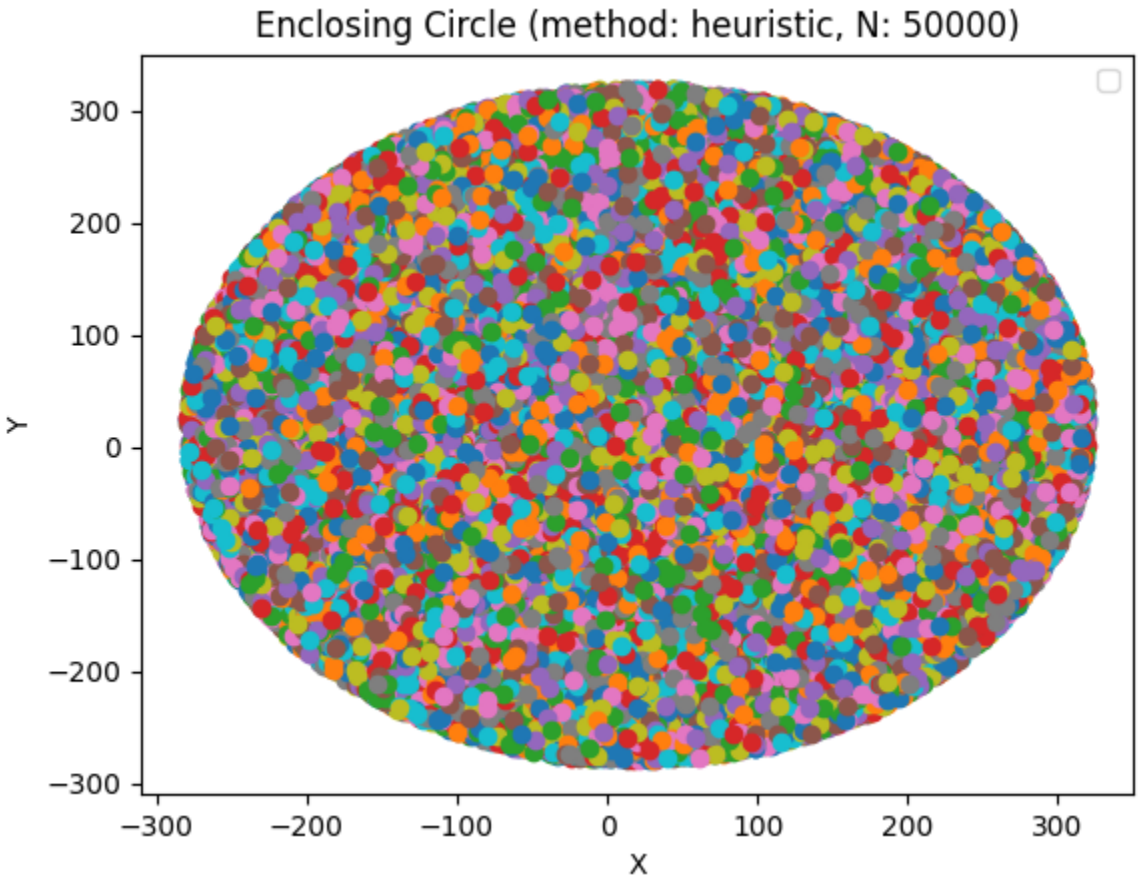
1) N = 40, base_circle = {x = 0, y = 0, r = 100}




```
# lua enclosing_circle/main.lua heuristic 40
 Found circle: { x= 3.8409872435194, r= 102.02700672891, y= -3.5096907994016, }
 Execution time: 2.9e-05 seconds
 Is this a valid result? true
```

Here we can see that the algorithm runs very quickly and finds a valid enclosing circle, but it is a bit off-centered and it's clearly not the smallest, as its radius is 102, which is bigger than the original one, 100. As expected, it is a good approximation.

2) N = 50000, base_circle = {x = 20, y = 20, r = 300}

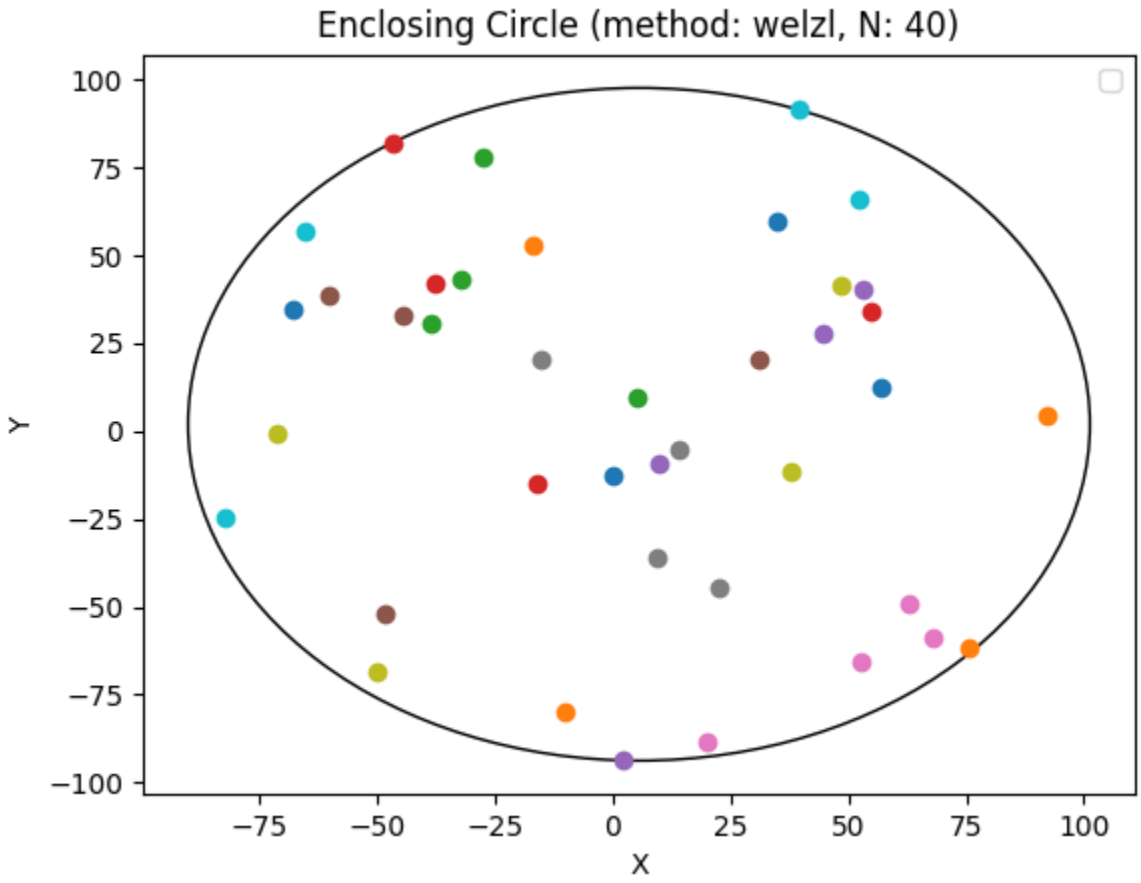



```
# lua enclosing_circle/main.lua heuristic 50000
 Found circle: { y= 19.981255872808, r= 300.09670109803, x= 19.887422816016, }
 Execution time: 0.01567 seconds
 Is this a valid result? true
```

Here we can see how the method performs with a very big data set (N=50k). As the circle is much more dense now, the heuristic method finds a very good fit, with radius ≃ 300.1. It also runs very quickly, still just a bit more than 0.01 second. Actually, by running it we can notice that the plot takes much more time to be drawn (around 5 seconds) than the algorithm takes to run.

Welzl's method

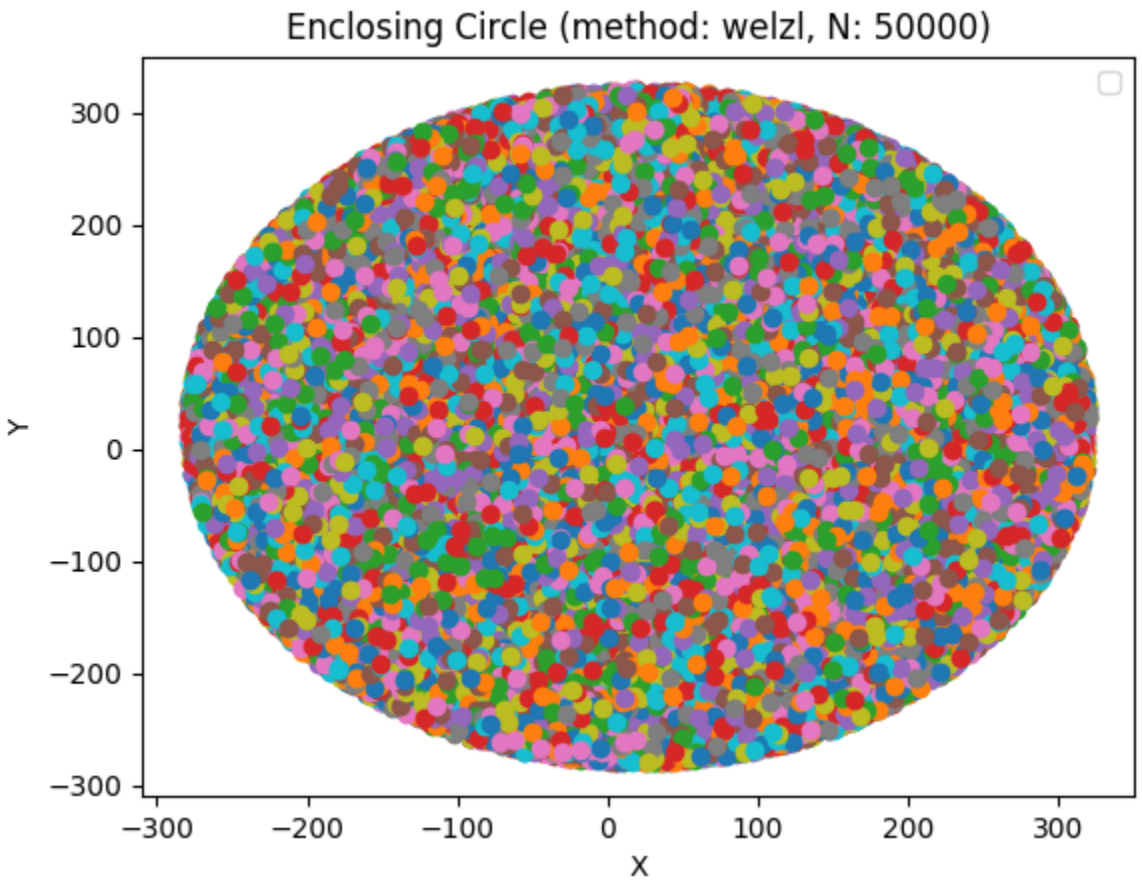
1) N = 40, base_circle = {x = 0, y = 0, r = 100}




```
# lua enclosing_circle/main.lua welzl 40
 Found circle: { y= 1.9533184228415, x= 5.5346455716859, r= 95.742357937499, }
 Execution time: 0.001507 seconds
 Is this a valid result? true
```

Welzl's algorithm performs as expected and finds an enclosing circle with a radius 95.7, which is smaller than the original one. We can see on the plot that there are 3 points in the frontier of the circle (a red and a blue on the top, and a purple at the bottom). Execution time is fast as expected.

2) N = 50000, base_circle = {x = 20, y = 20, r = 300}



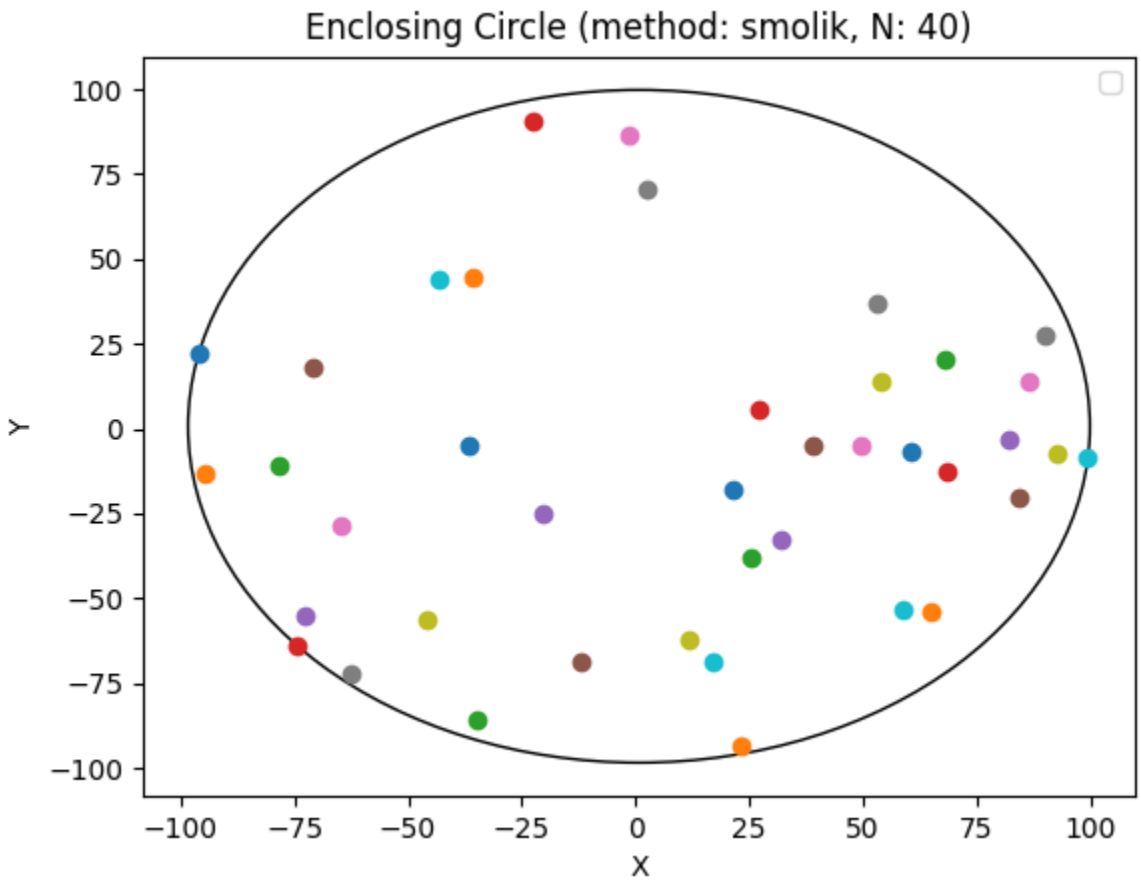

```
# lua enclosing_circle/main.lua welzl 50000
 Found circle: { r= 299.98333552735, y= 20.007720072314, x= 20.014413000266, }
 Execution time: 1.540309 seconds

 Is this a valid result? True
```

Here we try the Welzl algorithm with a big big data set. It runs in 1.54 s, which is still very fast, and finds an enclosing circle with radius 299.98, slightly smaller than the original one, as expected. The center of the circle {20.01, 20.01} is also very close to the original center {20, 20}.

Smolik's method

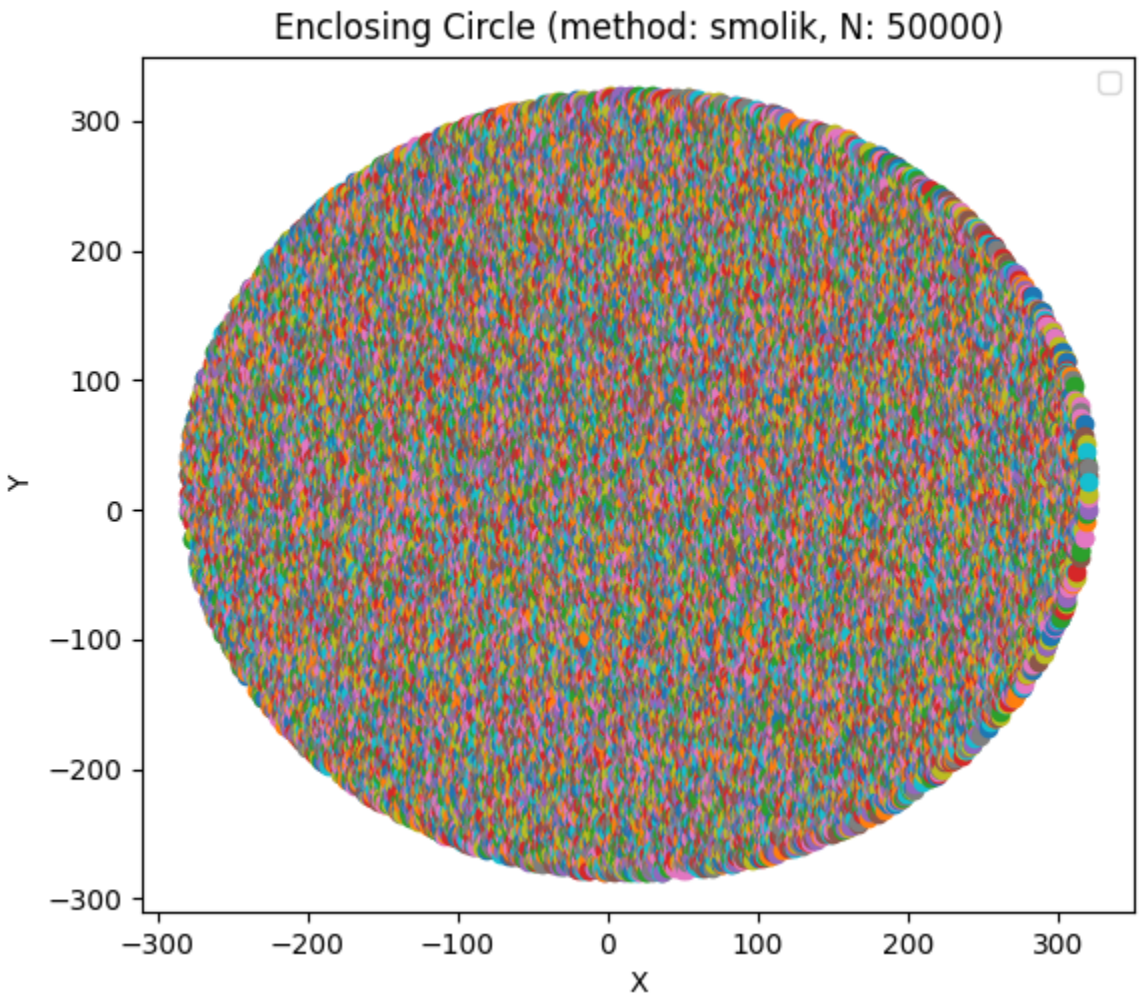
1) N = 40, base_circle = {x = 0, y = 0, r = 100}




```
# lua enclosing_circle/main.lua smolik 40
 Found circle: { r= 99.091264113435, x= 0.75279021077347, y= 0.69478918604595, }
 Execution time: 0.001366 seconds
 Is this a valid result? True
```

As the Smolik's algorithm is just a pre-processing followed by a Welzl algorithm, we can also expect to find two or three points in the frontier of the circle, which we can easily observe in this example (a blue on the left, a red on the left-bottom, and a blue on the right). We can see that the Smolik performed slightly better than the Welzl, having a time of 0.001366 against 0.001507. It also found a circle whose radius is smaller than the original one (r=100).

2) N = 50000, base_circle = {x = 20, y = 20, r = 300}

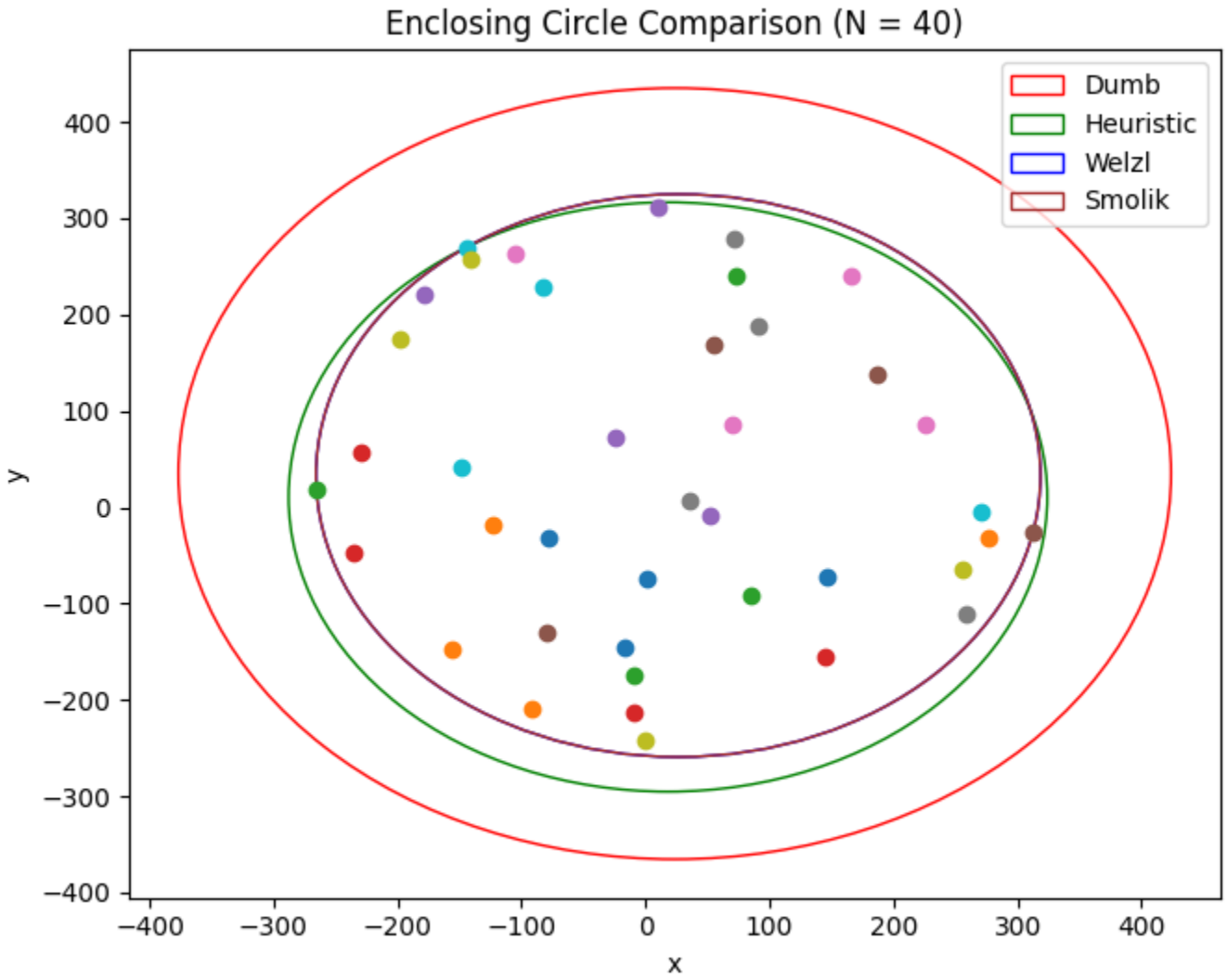



```
# lua enclosing_circle/main.lua smolik 50000
 Found circle: { r= 299.99838527239, x= 20.000260660971, y= 19.999666227166, }
 Execution time: 0.16171 seconds
 Is this a valid result? True
```

With a big data set, the Smolik's method performed very similarly to the Welzl's method. The results are valid, the radius is 299.998, very close to the original 300, and the execution time is slightly bigger than the Welzl one, which was unexpected. We will analyse them deeper in the following topic.

Methods comparison

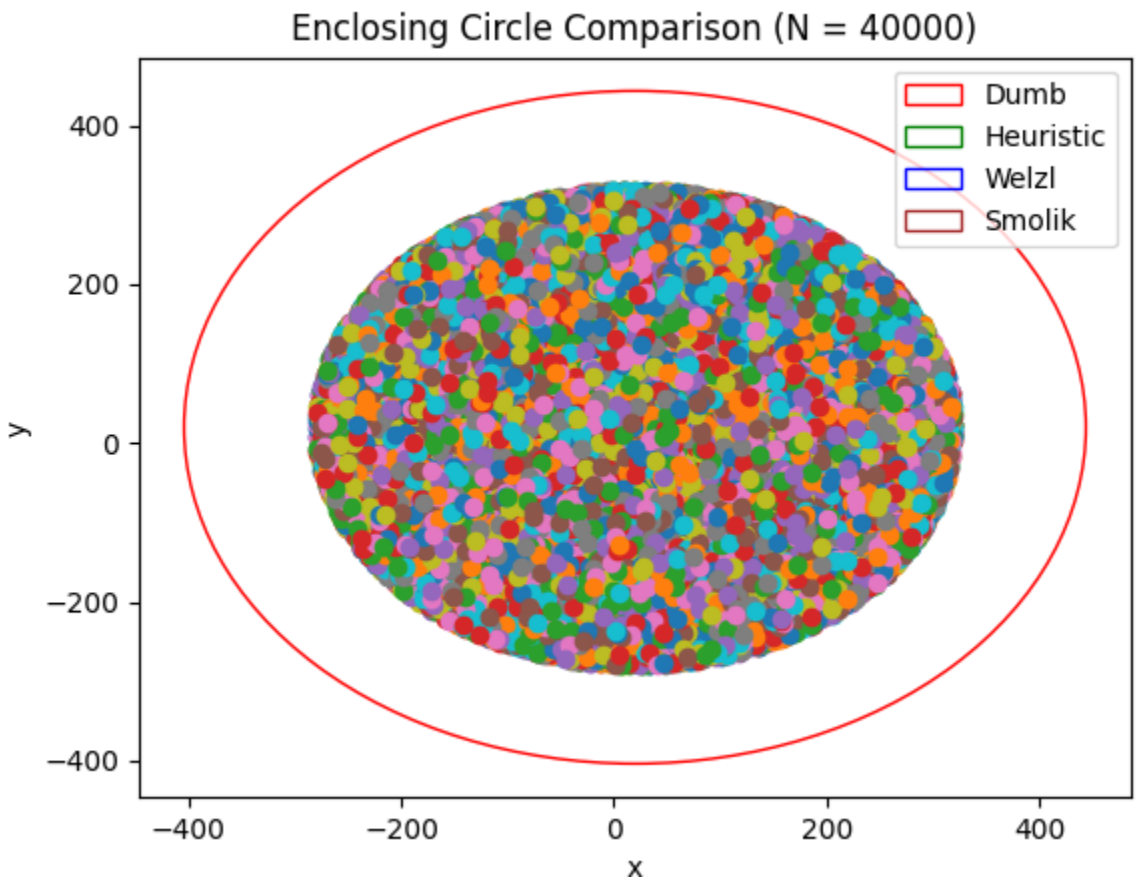
1) N = 40, base_circle = {x = 0, y = 0, r = 100}




```
# lua enclosing_circle/main.lua compare 40
 Dumb:      { y= 23.24167, x= 4.98556, r= 377.27595, }      in 1.7000000000001e-05 seconds
 Heuristic: { y= 45.86697, x= -26.7920, r= 320.6051, }      in 3.4999999999999e-05 seconds
 Welzl:     { y= 29.35402, x= 13.6090, r= 296.78592, }      in 0.000262 seconds
 Smolik:    { y= 29.35402, x= 13.6090, r= 296.78592, }      in 0.000183 seconds
```

By running all methods with the same data set, we can notice that the Welzl and Smolik methods found the exact same circle, which is expected, as the smallest enclosing circle can only have one right result, by definition. The Heuristic method found a bigger circle, but finished much faster. The Welzl circle is hidden in the plot as the Smolik circle was drawn last.

2) N = 40000, base_circle = {x = 0, y = 0, r = 100}

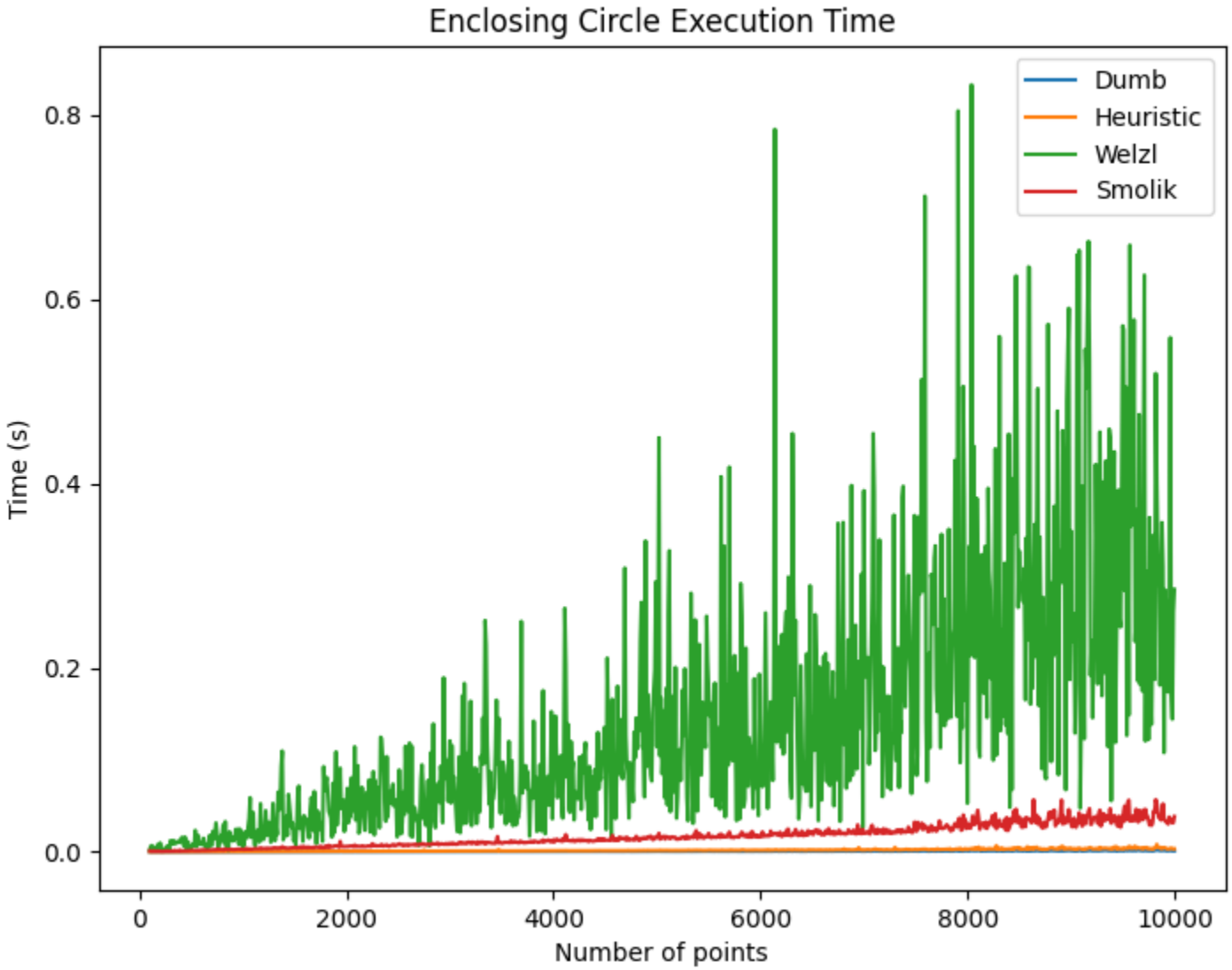



```
# lua enclosing_circle/main.lua compare 40000
 Dumb:      { r= 423.9118, y= 19.83133, x= 20.0358, }     in 0.004087 seconds
 Heuristic: { r= 301.4695, y= 21.52859, x= 20.0384, }     in 0.013835 seconds
 Welzl:     { r= 299.9835, y= 20.00845, x= 19.9882, }     in 0.178521 seconds
 Smolik:    { r= 299.9835, y= 20.00845, x= 19.9882, }     in 0.123716 seconds
```

Trying all methods with a large input of 40k points, we can confirm that the heuristic's circle gets (relatively) very close to the smallest circle of Welzl and Smolik. The Smolik method performed faster than the Welzl.

Time complexity comparison

To analyse the time complexity of these algorithms, we will measure their execution times for different sizes of data set. Specifically, we will try N=100 to N=10000, with a step of 10.



As the Smolik and Welzl algorithms are expected to be O(N) in a probabilistic measure, we can see that they form very noisy curves of time complexity, as expected. However, we can still identify their growth as linear. Smolik's has a smaller slope than the Welzl.

If we zoom on the plot, we can see that the Heuristic method has a very small slope, performing very closely to the the Dumb method: