**Problem context**

There are a lot of tools to measure distances while driving, or in an environment where you can only follow a path in a straight line between its connections, but there’re almost no apps that measure people path and distances in a open environment. This can be helpful in airports or in situations where persons need to know either the fastest way to go from one place to another walking or to know if its possible to go from one place to another in less than a given time.

**Solution**

For the nature of the problem, it can be solved by two different paradigms: logic and concurrent. Both paradigms offer advantages over the other.

Logic:

* It’s easier to have the fastest routes if the paths are measured with graph nodes. However, the solution turns more difficult if what we want is simulate width in the paths between the nodes.

Concurrent:

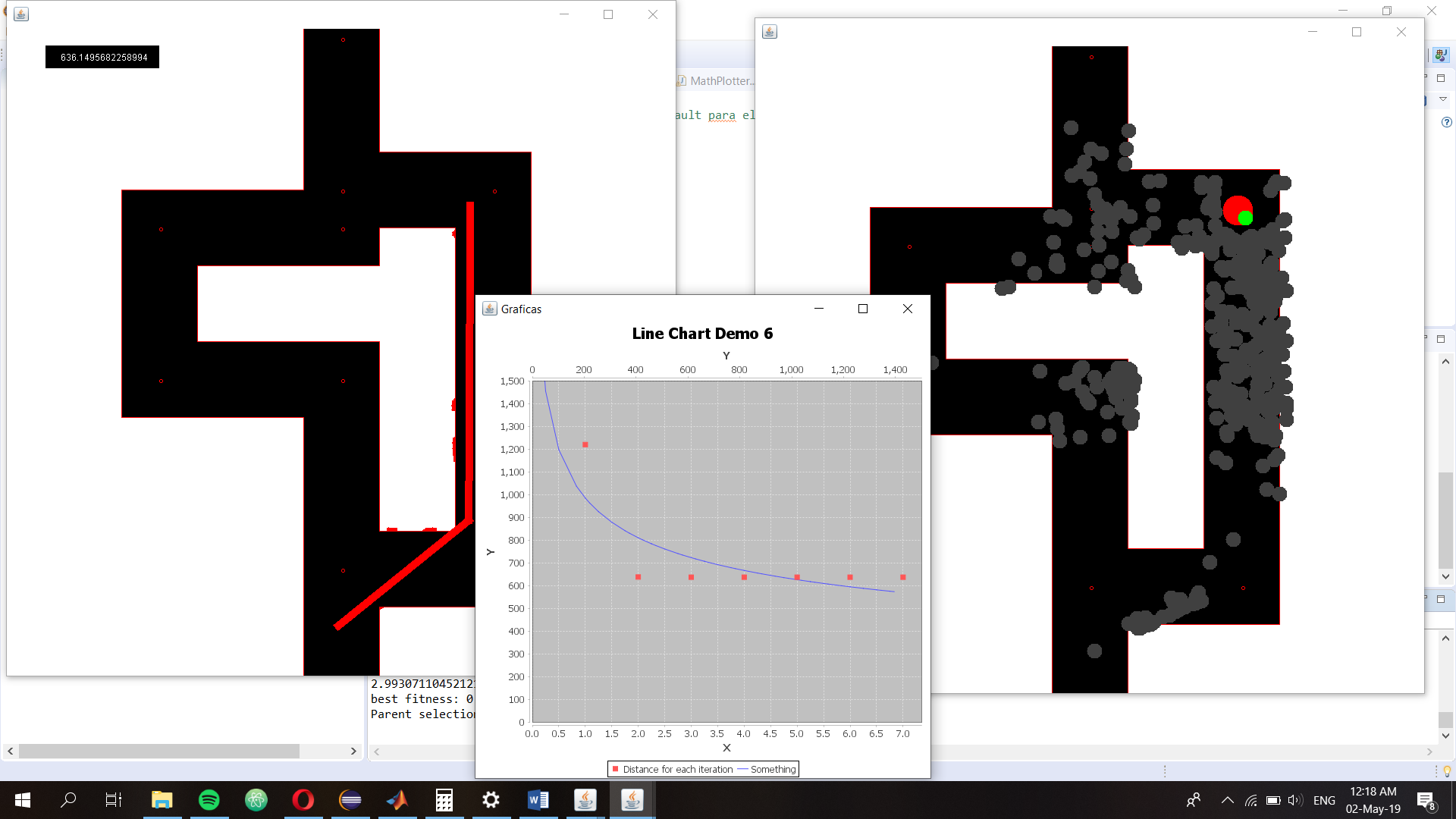
* Concurrency may not have anything that makes it easier to solve the problem, since it’s focus is only to compute faster. If we combine concurrency with OOP, we can achieve a better result since modeling the node connection’s width its easier. OOP by itself its more expensive to implement than logical programming, however if we consider concurrency, it can get better results than logical programming.

The approach of the solution was using a genetic algorithm. The genetic algorithm operations are supported and controlled with concurrency operations that make loading times shorter.

**Results**

The genetic algorithm at the beginning was vey expensive, the loading times for its operations varied between the 5 to the 25 seconds each (parent selection, mutation, calculate fitness) when using a population of 1000 dots. With concurrency the times using 5000 dots has dropped to:

* Calculate fitness: 1 millisecond
* Parent selection: 2 seconds
* Mutation: 1 millisecond



**Conclusions**

Concurrency is a very powerful tool, moreover if we combine it with other paradigms to solve a problem. It brings a lot of advantages; however, each advantage can turn into a problem by itself if it is not used wisely.

The principal problem I afford while doing this project was data incongruency, it is very important to use Thread safe structures or to know how to protect the data involved. There are a lot of things that you need to follow in order to have a concurrent system, but the best practices are: refactoring, documenting and verifying all the data.

Concurrency as we saw in class is currently the way to improve in computation, hopefully in the future it is easy to do graphical programming as it is using the cpu in concurrency.

**Setup instructions**

1. Download JFreeChart: <http://www.jfree.org/jfreechart/download.html>
2. Move the files inside the Zip to a route that its easy to access
3. Clone the git repository to your computer with the following command:
4. Git clone
5. Change the library directions from the project build path to the ones where you installed it in your PC.
6. One should reference \lib\jfreechart-1.0.19.jar and the other one \lib\jcommon-1.0.23.jar

**References**