Graph Drawing with Minimum Edge Crossing

Jingyi LI
École Polytechnique, Institut Polytechnique de Paris
jingyi.li@polytechnique.edu

March 17, 2023

1 Introduction

1.1 Problem Definition

The challenge focuses on minimizing edge crossings on a given set of **N** points. The input therefore consists of a (undirected) graph of size **n** and a set of points (at least as many as vertices of the graph). The goal is to assign the **n** vertices of the graph to the given points, such that the number of crossings in the resulting drawing is as low as possible.

1.2 Input and Output Formats

The input instances are encoded in Json format and contain:
• a collection of input **points** (with integer coordinates): each point stores its **'x'** and **'y'** coordinates (integer values).
• a collection of **nodes** (of the input graph): nodes have indices in **[0..n-1]** and may be provided with initial coordinates or not (if not, both **x** and **y** coordinates are set to 0)
• a collection of **edges** (of the input graph): each (undirected) edge is encoded with a pair of indices (its **'source'** and **'target'** node).

The output is the resulting graph to a text file encoded in Json. The output format is the same as for the input: each node should be assigned with **x** and **y** integer coordinates (corresponding to the solution).

2 Algorithm and Analysis

2.1 Force Directed Method

Given an input graph without initial coordinates, we first employ the force-directed method to determine the initial position of each node.

The force-directed algorithm is a graph layout technique designed to visualize graphs or networks in a visually appealing and informative manner. It conceptualizes nodes as particles with mass and edges as springs connecting these particles. The algorithm simulates the physical forces between particles and iteratively adjusts their positions until reaching a stable state.

Assuming we have N nodes and E edges, and we perform I iterations, the time complexity of our implementation of the algorithm is $O(I(N^2 + E))$. In each iteration, calculating repulsive forces between all pairs of nodes takes $O(N^2)$ time, as there are $\frac{N(N-1)}{2}$ pairs. Meanwhile, computing attractive forces between connected nodes in each iteration takes $O(E)$ time since each edge must be considered.

We implemented a function **fruchterman_reingold** for it, but also used an existing implementation of the library NetworkX.

2.2 Evolutionary Algorithm

Taking into account that input graphs are often non-planar, and the potential positions of the nodes are both discrete and limited, it is difficult to employ existing theorems, such as Tutte's theorem or Schnyder's theorem, to obtain satisfactory results that fulfill all conditions. The problem is nonlinear, non-convex, and may lack gradient information and analytical solutions. Furthermore, several objectives must be optimized within a vast search space. Consequently, this situation is ideal for applying evolutionary algorithms.

Our method, outlined in Algorithm 1, involves selecting an edge at random during each iteration, assigning higher weights to edges with more intersections. We then randomly select two unique points as the new positions for the chosen edge's endpoints. This point selection process is repeated k times, and the best individual is chosen. If the number of edge crossings is reduced, we update the positions of the two nodes; otherwise, their positions remain unchanged. This procedure continues until the edge crossings reach zero or the allocated time elapses. Implementing the evolutionary algorithm is a straightforward process. We choose $k = 5$ in practice.

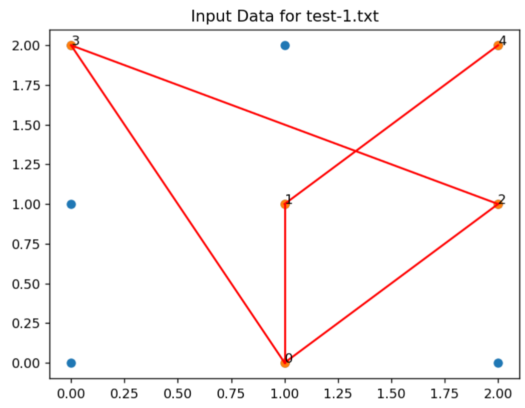
Algorithm 1 Evolutionary Algorithm

```
Input: A graph $G = (V, E)$ where each node has a position and a point set P
while true do
  Choose an edge $e \in E$ randomly with weights
  for $i = 0$ to $k$ do
    Choose 2 distinct points $a, b$ from P
    Put the endpoints of edge $e$ to $a$ and $b$ and get a new graph $G_i = (V_i, E)$
  end for
  Select $G'$ with minimum edge crossings from $G_1, ..., G_k$
  If num_edge_crossings($G'$) < num_edge_crossings($G$) then $G = G'$
end while
Output: the graph $G = (V, E)$
```

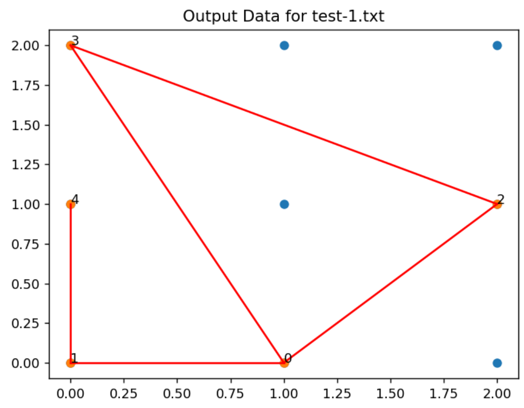
3 Results

3.1 Visualization

For the initial four instances, we only employed the aforementioned evolutionary algorithm, achieving very few edge crossings.

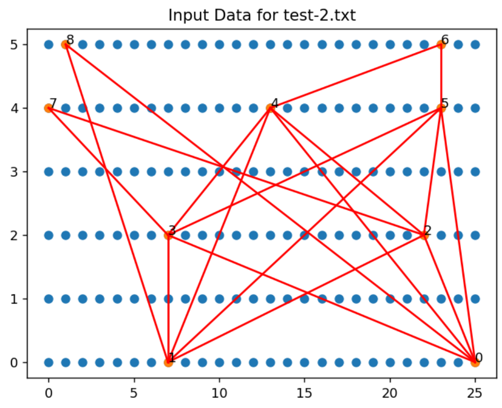


(a) Input for test-1 (1 crossing)

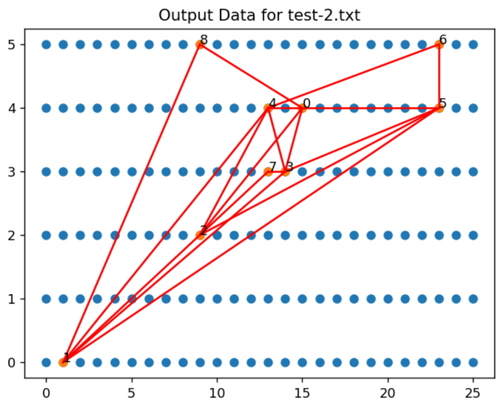


(b) Output for test-1 (0 crossing)

Figure 3.1: Result for test-1



(a) Input for test-2 (22 crossings)

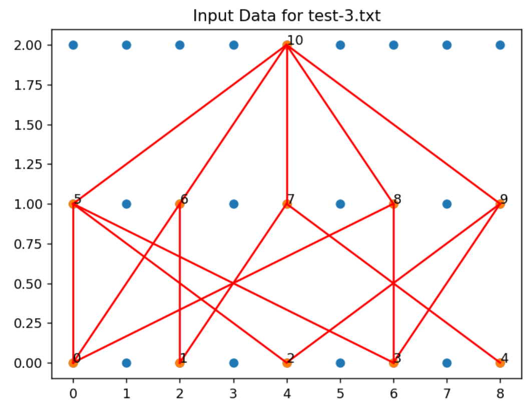


(b) Output for test-2 (3 crossings)

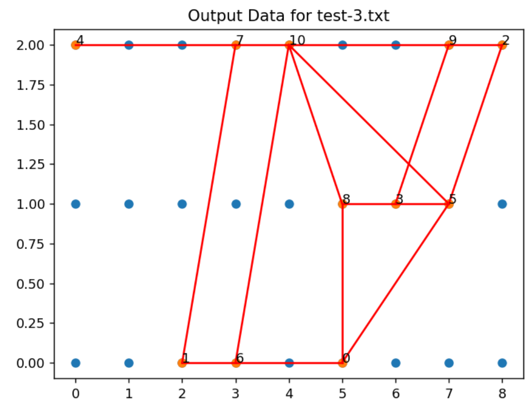
Figure 3.2: Result for test-2

For test-5 and test-6, the nodes initially lack optimal coordinates. Consequently, we utilize the force-directed method for initialization. In the case of test-5, employing the force-directed method alone yields an excellent result, as demonstrated in Figure 3.5.

For test-6, upon utilizing the force-directed method, we observe 6745 edge crossings, as illustrated in Figure 3.6. The graph appears quite cluttered, necessitating further optimization using the evolutionary algorithm. Following 815 iterations, the edge crossings are reduced to 1203, as depicted in 3.7.

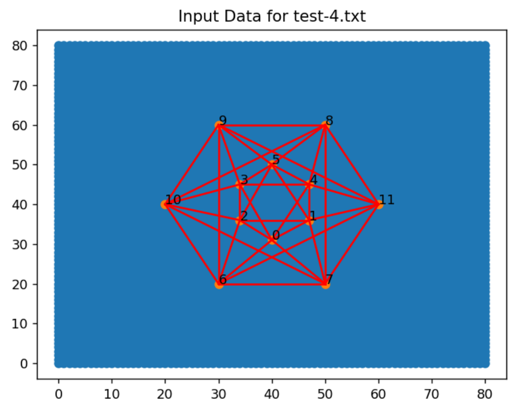


(a) Input for test-3 (16 crossings)

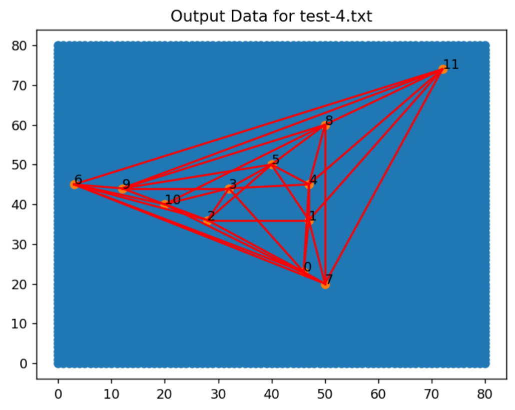


(b) Output for test-3 (1 crossings)

Figure 3.3: Result for test-3

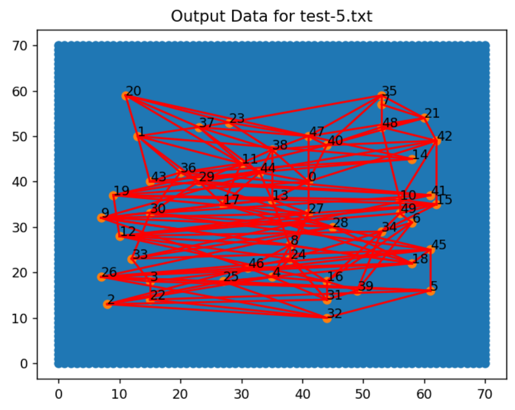


(a) Input for test-4 (24 crossings)

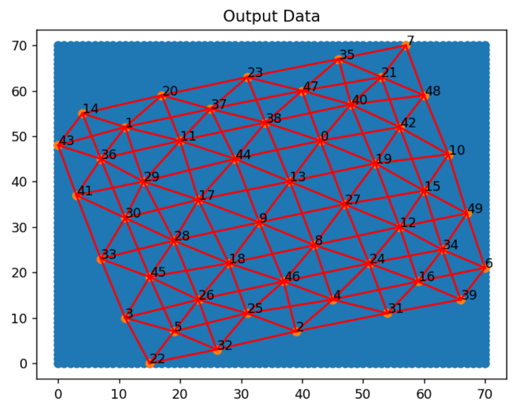


(b) Output for test-4 (12 crossings)

Figure 3.4: Result for test-4



(a) Input for test-5 (534 crossings)



(b) Output for test-5 (32 crossings)

Figure 3.5: Result for test-5

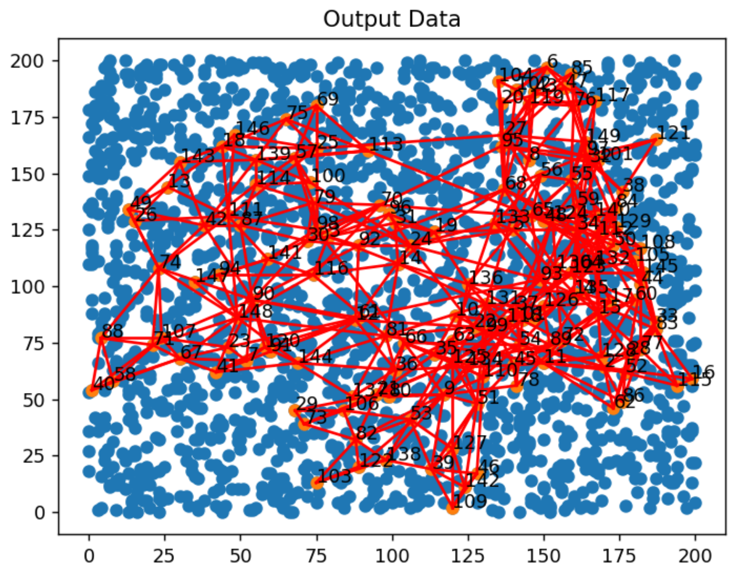
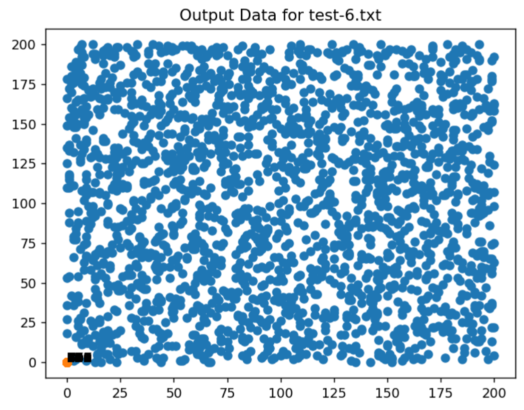


Figure 3.6: Initialization result (6745 crossings) for test-6


(a) Input for test-6

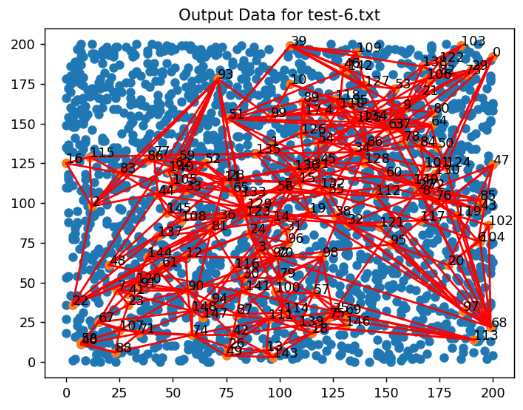

(b) Output for test-6 (1203 crossings)

Figure 3.7: Result for test-6

We show the iteration process for the test-6, as observed in Figure 3.8, the rate at which the number of crossing edges decreases becomes progressively slower.

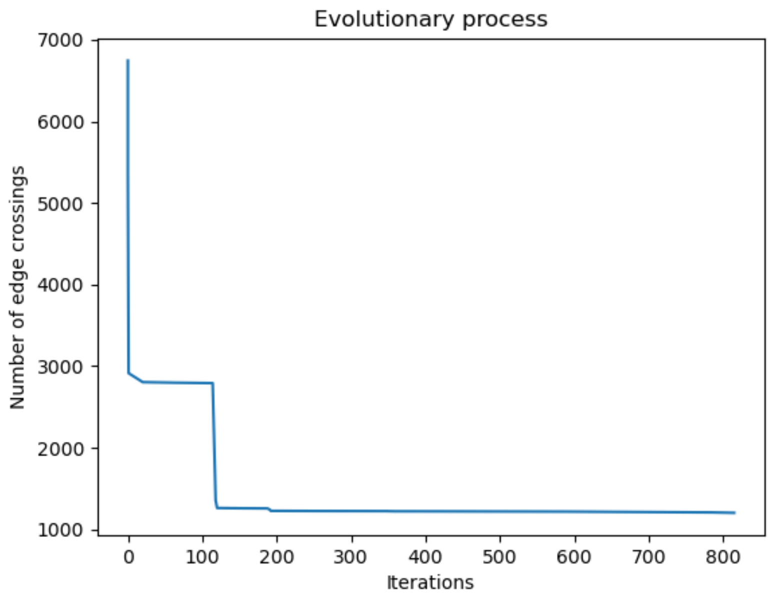


Figure 3.8: Evolutionary process for test-6.

3.2 Running Time

In the evolutionary algorithm, the most time-intensive aspect involves counting the number of edge crossings, which is proportional to the square of the number of edges. We tested our algorithm on a MacOS system equipped with 8GB memory and a 2.3 GHz Dual-Core Intel Core i5 processor, utilizing a single core. The Table 3.1 presents the running time and results for each instance:

INSTANCE	ITERATIONS	TIME(S)	NUM OF EDGES	INPUT CROSSINGS	OUTPUT CROSSINGS
TEST-1	2	0.007983	5	1	0
TEST-2	382	2.965	18	22	3
TEST-3	1896	14.70	16	16	1
TEST-4	3547	104.3	36	24	12
TEST-5	NO	NO	161	534	32
TEST-6	815	3269	476	6745	1203

Table 3.1: Statistics for each instance.

The code is available at `https://github.com/JeansLli/graph_drawing`.