Spatial random graphs generation — comparison of naive and evolutionary approach

Geometrische Zufallsgraphen — Vergleich zwischen naiver und evolutionärer Ansatz

Blagovest Achanov

MSc Student at Faculty of German Engineering and Industrial Management
Sofia, Bulgaria, blagovest.achanov@fdiba.tu-sofia.bg

***Abstract* —** This work introduces an evolutionary approach for generation of spatial random graphs based on Waxman graph and a naive approach for generation of undirected connected weighted graphs in given 2D space. Both approaches are experimentally compared by measuring their execution time, graph connectivity and maximal average length of edges, proving the practical usability of evolutionary algorithms combined with Waxman graphs for fast generation of connected undirected geometric graphs.

***Zusammenfassung* —** Diese Arbeit stellt ein naiver und ein evolutionärer Ansatz für Generierung von gewichtete ungerichtete verbundene Zufallsgraphen. Beide Ansätze werden mit verschiedenen Versuche gegen Ausführungszeit der Ecken und maximale Durschnittdistanz der Ecken. Die Versuche werden die Gelegenheit für praktische Anwendung der evolutionären Algorithmen für schnelle Generierung von geometrischen Graphen.

I. Introduction

Different situations such as the Seven Bridges of Königsberg problem [2], a network topology, an electric circuit or relationships in a social network can be displayed as graphs. When implementing such a system, it should be tested with different graphs that simulate some special conditions. Those graphs, which are generated using probability approaches are called random graphs.

The first studies in the area of random graph generation (RGG) are done in the late 1950s by Erdos, Renyi and Gilbert [3]. Both Erdos-Reniy and Gilbert models use an iterative way to generate graphs given number of nodes and don't support spatial graphs. Later, Waxman presented a model for generation of spatial random graphs. Tettamanzi [7], Cordella, De Stefano, Fontanella, and Marcelli, [5] worked on evolutionary algorithms applied to graphs. Shopov and Markova compared different approaches for RGG in [8] and implemented an evolutionary approach in [9].

The purpose of this work is to update the traditional Waxman graph to use an evolutionary approach and to compare it to a non-evolutionary approach, where all graphs meeting some requirements are generated (naive approach). Both approaches are going to be described in theory in part III and IV and practically compared in part V of this work. As result, the work should prove the practical usability of an evolutionary approach combined with Waxman graph.

II. Related works

Some of the publications, nominated in the introduction were combined and elaborated in this work. This part of the work introduces them in some detail.

A. Waxman graph

Waxman graphs are particular class of random graphs and are used for modelling network topologies for evaluating routing algorithms (see [1]). Vertices of the graph are randomly distributed over a rectangular area. Edges between nodes are added randomly using a probability mechanism, where the probability that two vertices are connected decreases exponentially depending on the Euclidean distance between the nodes. The number of vertices N is pre-determined [1]. The edge existence probability can be described on the following way:

$$P(\{u, v\}) = \beta e^{\frac{-d(u,v)}{L\alpha}} \quad (1)$$

U and v are vertices in the graph, L is the maximum Euclidean distance between any two nodes (see [9]), α and β are parameters in range (0, 1]. According to [6] larger values of α increase the densities of short edges and larger β values increase the density of longer edges. Waxman sets the values of α and β = 0.4 [1]. For the purpose of this work, values of α and β will be preset. Also, the maximum distance L will be calculated as the maximal possible distance between the nodes in the graph.

B. Evolution in graphs

EvoGeneSys [5] introduced in multilist structure and implemented evolutionary operators to work with it. They demonstrated the practical applicability of these methods on undirected unweighted graphs. The algorithm described there was not applied on weighted graphs with coordinates in 2D space. It was also not compared to a brute force approach.

This work uses as basis Waxman graph and updates the algorithms from [5] to work with spatial graphs. The usability of the evolutionary algorithm will be proved by comparing it to a naive approach, which will be introduced in the next part.

III. Naive Approach for RGG

The naive approach for graph generation, elaborated in this work, returns all possible graphs that meet specific requirements such as number of nodes, edges and width and height of a two dimensional space that contains the graph. All returned graphs are stored in a list. The algorithm can be split in 4 parts, illustrated in the following sections, where for each part is given a formula, giving all possible combinations:

A. Nodes generation

This part ensures, that a node is generated for each possible point in a rectangular 2D shape with integer values of coordinates. The total number of nodes for such a shape is:

$$vertexCount = (x_{max}-x_{min})(y_{max}-y_{min}) \quad (2)$$

B. Vertex selection

The vertex selection is used to combine different nodes in a graph. After generating all nodes (see previous paragraph), those vertices should be grouped in different graphs. For this reason, a combination is used- the total number of possibilities for nodes in graph is:

$$C(vertexCount, nodesInGraph) \quad (3)$$

C. Edges generation

The purpose of edges generation step is to generate all possible edges from set of nodes. According to its definition, each edge connects a pair of nodes [4]. So, in this case the total number of possible edges for each combination of nodes will be:

$$allEdges = C(nodesInGraph, 2) \quad (4)$$

For execution of the naive approach, a number of edges in graph is passed. To ensure the graph can be connected, this number must be at least nodesInGraph – 1 for an undirected graph. Then, the total number of edges combinations for a set of nodes will be:

$$C(allEdges, numberOfEdges) \quad (5)$$

D. Graphs generation

This step combines node sets and edge sets, in order to generate graphs. The following equation gives all possible graphs given shape coordinates, number of nodes and number of edges:

$$graphsCount = C(C((x_{max}-x_{min})(y_{max}-y_{min}), nodesInGraph), numberOfEdges)*C((x_{max}-x_{min})(y_{max}-y_{min}), nodesInGraph) \quad (6)$$

The set of graphs is generated following the pseudocode:

```
naiveGenerator( numberOfNodes, numberOfEdges,
rectangleCoordinates){
    Graph[] generatedGraphs;
    //generate all possible nodes in rectangle
    Nodes[] = generateNodes (rectangleCoordinates);
    //generate all combinations of nodes
    nodesForGraph[] = generateNodesCombinations(numberOfNodes,
    nodes);
    for(currentNodes : nodesInGraph){
        //generate all possible edges between nodes
        edges[] = generateEdges(currentNodes);
        //generate combinations of edges in graph
        edgesCombinations[]= generateEdgesCombinations(edges,
        numberOfEdges);
        for(currentEdges: edgesCombinations){
            Graph currentGraph = new Graph (currentNodes,
            currentEdges);
            generatedGraphs.add(currentGraph);
        }
    }
    return generatedGraphs;
}
```

Fig. 1 Naive approach- pseudocode

After all having all possible graphs, they can be evaluated given criteria and be reordered using it. For example, they can be filtered to a subset of connected graphs. However, the main generation approach is independent from this evaluation.

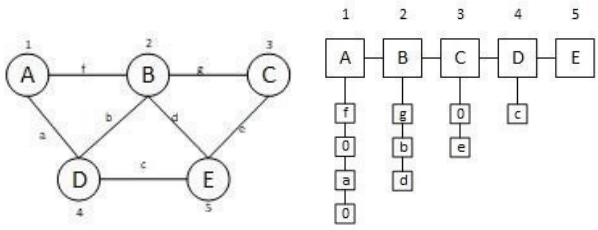
The naive approach has the complexity of ***O(C(C(shape,nodes), edges)* C(shape, nodes)***, where shape describes the number of all possible points with integer coordinates in the given 2D space. This complexity in O notation shows, that the speed of the algorithm will be significantly reduced by changing the total number of nodes in graph, all possible points or even the edges count. This assertion will be verified in the practical part of the work.

IV. Evolutionary approach for RGG

Another approach for RGG is to use the evolutionary operators in order to create new graphs. For this purpose, each graph should be represented in an appropriate structure, allowing an easy mutation and crossover. Such a structure was developed the in [5] multilist. For the purpose of this work, the multilist has to be updated to support evolutionary operations with spatial graphs respectively Waxman graph.

A. Multilist

Multilist (or ML) is a structure, used for representing graphs is as set of (N) lists, where the *main list* contains the nodes and the other lists (or *sublists*) contain the edges of a graph (see [5]). Each list of edges (or sublist) is connected to a node and contains the edges from this node. The size of each sublist depends on the position of the node in the main list and contains edges to the upcoming nodes from the main list following their order. As an example, the edge list from of the first node will contain N-1 edges. If an edge does not exist, it is represented as null. The edge list of the second node contains the connections to all upcoming nodes- or N-2 edges. The last node from the main list doesn't contain a sublist. Each edge is stored only in one list. The described multilist has a triangular structure and an example is given in the following figure:



(a) (b)

Fig. 2 (a) graph (b) multilist representation

The given graph in Fig 1. (a) is converted to a multilist in (b). Particulary, if we swap the indexes of vertex A and vertex B, a different multilist will be generated.

B. Crossover

Crossover using multilist structure according to [5] is done in two steps: each multilist is split in two sublists, sublists are swapped between and merged. As result 2 new individuals (offspring) is generated.

1) Split

To split a multilist, a split point in interval ($0 < S < N-1$) must be selected, where **N** is the number of vertices and **S** is the split point. The next figure gives an example for split operation of the graph in fig. 1 with split point 2:

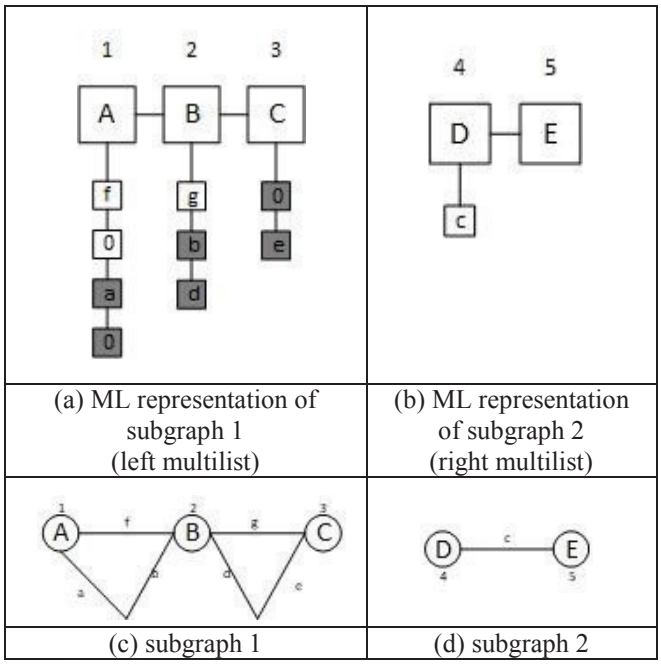


Fig. 3 Split operation

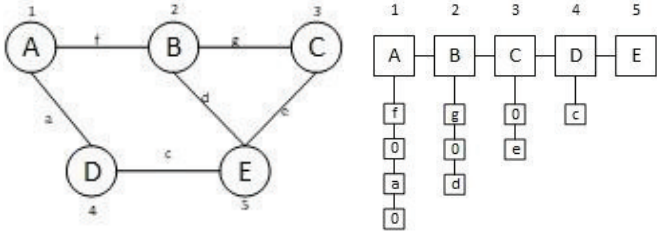
After splitting the multilist in two, some of the edges of the left multilist (colored in fig 2) will get invalid. They are used during merge operation to generate edges between the two sublists.

2) Merge

The purpose of merge operation is to generate a new individual by merging 2 multilists, describing 2 subgraphs. The merge is done naively by merging the 2 main lists. The left sublist contains edges to the right, so each edge has to be updated to lead to the new vertex and the distance should be recalculated. Important for merge operation is that the total number of vertices before splitting and after merging the graph does not change. This means that to the left multilist must be merged a right multilist with same number of nodes that was previously cut from the graph to produce the left multilist.

C. Mutation

Mutation is the second evolutionary operator. It appears in a random change in the individual (see [5]). Given a mutation rate, a mutation should appear not in all individuals. In this work, a mutation is done by randomly adding or removing and edge. Using the multilist, a random node and random edge from its sublist is selected. If the edge is null, a new edge is added. Otherwise the edge is removed.



(a) graph (b) multilist

Fig. 4 Mutation

The previous figure gives an example of mutation in the example graph from fig. 1. Due to mutation, the edge **b** between vertices **B** and **D** is removed. This also affects the multilist in (b).

D. Selection and fitness function

Selection operation is used to select a subgroup of individuals from the offspring that meet some conditions better than others. In the current case, those conditions are graph connectivity and maximal average edge length. The number of selected individuals is passed as parameter. The selected individuals are used as parents for the next epoch of the evolutionary algorithm (see Fig. 5).

```
evolGenerator(initNumGraphs, nodes, coords, alpha, beta, epochs,
offspringNum, crossoverPoint){
    Multilist[offspringNum] resultMultilist;
    Graph[] initPopulation = generateInitPopulation(nodes, coords,
    populationCount, alpha, beta);
    Multilist[] initMultilist = convertToMultilist(initPopulation);
    for( 0 .. epcohs){
        epochResult = doCrossover(resultMultilist, crossoverPoint);
        resultMultilist =doSelection(epochResult, offspringNum);
    }
    Graph[] resultGraphs = convertToGraph(resultMultilist);
    Return  resultGraphs;
}
```

Fig. 5 Evolutionary approach- pseudocode

V. Experiments and Results

In the practical part of this work both naive and evolutionary approach for generation of spatial random graphs were implemented in and a comparative analysis of them was done. In order to remove any language-dependent differences in performance both algorithms were implemented in Java. Multilists were represented using hash maps where the start vertex is the key and the value is a list of all edges from this node. This structure was selected because it uses key-value representation. In order to represent a graph, node, edge and graph classes were defined. The project was compiled and all experiments were executed on the same machine with 8 GB RAM and Intel i7-2640M CPU.

A. Experiment conditions and measures

In order to compare both algorithms, different experiment conditions are used. Dimensions of the rectangle are changed in order to test both approaches. Timer is implemented to measure the execution time of each algorithm. Maximal average edge distance is the other measured metric. Number of edges parameter is used only for the naive approach. In order to decrease its execution time, for number of edges is chosen to be equal to the number of vertices -1. This is used of the condition that the generated graphs are connected. In theory, the evolutionary approach can generate a connected graph with number of edges greater the number of nodes -1, but in most of the cases the generated graph contains exactly that number edges. This can be controlled by changing alpha and beta coefficients, but for the purpose of this work values are fixed. As described in [1], the values of α and β are set to be 0.4, the mutation rate is set to 0.15 and the offspring count is 20. Two values for count of epochs were chosen- 10 and 20. They're used to allow measure the time and result changes caused by the change of epochs. The following table shows the experiment conditions:

Experiment number	**# of nodes**	**# of edges**	**space**
1	5	4	3x3
2	5	4	4x4
3	5	4	4x5
4	5	4	10x10
5	5	4	20x20

Table 1 Experiment conditions- space and graphs

Experiments 1, 2 and 3 are executed for naive and evolutionary approaches. The last 2 experiments were done only for evolutionary approach, because the execution of them with a naive approach did not finish after 10 minutes. The results and conclusions from them are given in the following paragraph.

B. Results

The following diagram shows the execution time, measured for each of the experiments. Each experiment was executed five times. After that, the average time was calculated.

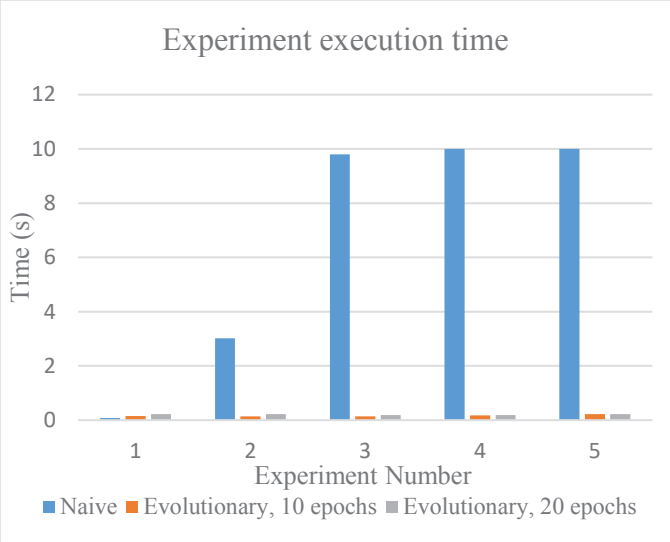


Fig. 6 Execution time

The measured time shows the rapid difference in the execution time of the naive approach in different cases. This behavior is expected and follows the description given in point III. The evolutionary approach works faster in 4 of 5 test cases. The increasing of space dimensions slows down the execution of the evolutionary method too, it's still much more appropriate that the naive approach. For test 4 and 5, the naive execution did not finish in 10 minutes and was not measured. The factor which increases the execution time of the evolutionary approach is the number of epochs. The execution of 20 epochs is almost always slower than the execution of 10 epochs. As a conclusion, the evolutionary approach is relevant for smaller and bigger spaces. The naive is not useable for dimensions over 4x5.

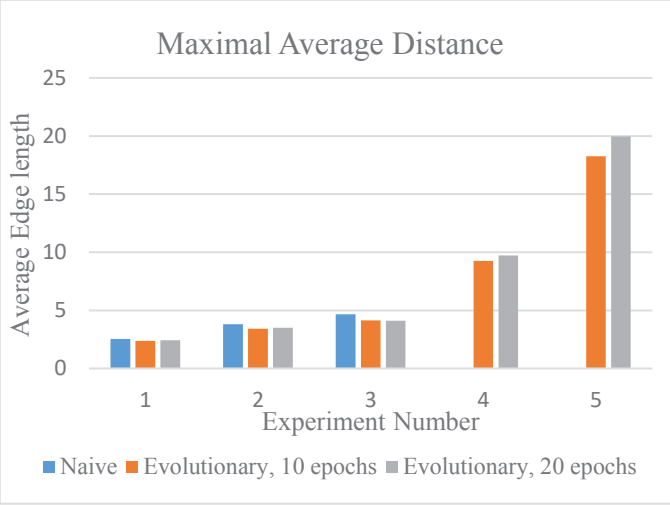


Fig. 7 Maximal average distance

Fig. 7 shows the metrics measured for maximal average distance on an edge in graph. The naive measures are control ones- they show the maximal possible distance. Compared to them, the results from the evolution are smaller. However, the difference in the results isn't very big. More, the increased number of epochs increased the result in more of the cases. This behavior is expected- the purpose of the evolutionary approach is not to give the best solution, but to give a result, which is optimal for some requirements. Theoretically the evolutionary approach can produce same result as the naive approach. The missing results for naive approach for experiments 4 and 5 are caused by the long execution time which was previously explained.

Analyzing both metrics, the evolutionary approach for spatial random graphs generation is applicable for graphs in small and bigger spaces. It produces relevant results for shorter execution time compared to naive approach which finds the best graph answering the distance condition, but it's not applicable for bigger dimensions or bigger graphs. Thus, only the evolutionary approach is practically applicable.

VI. Conclusion

The aim of this work was to create an evolutionary approach for generation of spatial random graphs and prove its usability in different conditions. For this, the Waxman graph was used. It was combined with an evolutionary algorithm using multilists following the approach described in [5]. During the work, a naive approach, generating all possible graphs given size and space was created. Both approaches were implemented and compared. The results showed the usability of the introduced evolutionary algorithms for generation of larger geometric random graphs. Analyzing the values of alpha, beta and mutation coefficients can improve the results of the evolution, which can be done in another work, related to this.

Acknowledgment

I would like to express my gratitude to Dr. Vanya Markova and Dr. Vetseslav Shopov for the support and advices during the research and the development of this work.

References

[1] B.M. Waxman, "Routing of multipoint connections", *IEEE Journal on Selected Areas in Communications*, 1988, pp 1617-1622

[2] See Shields, Rob, "Cultural Topology: The Seven Bridges of Königsburg 1736", *Theory Culture and Society*, 2012, pp 29.

[3] Roughan, Matthew, Tuke, Jonathan and Parsonage, Eric, "Estimating the Parameters of the Waxman Random Graph", *eprint arXiv:1506.07974*, 2015

[4] Robin J. Wilson, *Introduction to graph theory, 4th edition*, Longman, 1996

[5] L. Cordella, C. De Stefano, F. Fontanella, A. Marcelli, "EvoGeneSys, a New Evolutionary Approach to Graph Generation", *Applied Soft Computing 13 (4)*, 2013

[6] Naldi, Maurizio, "Connectivity of Waxman topology models", *Journal of Computer communications 29.1* ,2005, pp 24-31.

[7] Tettamanzi A.G.B. "Drawing Graphs with Evolutionary Algorithms", *Adaptive Computing in Design and Manufacture*. Springer, 1998, London

[8] Ventseslav Shopov, Vanya Markova, "COMPARISON OF RANDOM GRAPH GENERATORS", *Proceedings of the International Conference on Information Technologies (InfoTech-2016),* 2016

[9] Ventseslav Shopov, Vanya Markova, "EVOLUTIONARY APPROACH FOR SOLVING DYNAMIC GRAPH PROBLEMS", *Proceedings of the International Conference on Information Technologies (InfoTech-2016),* 2016