A Random Flat Graph Drawing Generation Algorithm for Testing Printed Circuit Board Tracing Software

Sergey Sgadov [1]

[1] *Zaporizhzhia National University, Address, Zhukovskogo st. 66, 69600, Ukraine*

Abstract

Printed board circuits tracing program design need in testing steps. In case of preplanarization stage it is need to use planar graph data for test set. This paper presents a simple algorithm for generating a family of random planar graphs based on a triangulated graph that can be used in algorithms on graphs. The algorithm is based on the principle of removing random edges and generate both an adjacency matrix and a flat graph drawing. Generated graphs are connected undirected graphs without loops and multiple edges, without bridges and articulation points, without vertices with a local degree less than or equal to two. Results of performance measurement experiments are given.

Keywords

Random flat graph, graph drawing, algorithm on graph, graph generation, planarization

1. Introduction

There are two trends in methods of printed board circuits (PCB) programs tracing algorithms design – they are a geometrical approach (for example, wave methods of Lee) and using of the topological of connections graph of constructs [1]–[3]. In the algorithms, implementing topological approach great attention is paid to operations on flat sugrafs [4]–[6]. For example at the work [5] the flat part of the graph is built with extraction algorithm of subset of isometric cycles with a capacity equal to the cyclomatic number of the graph, characterizing with minimum value of the MacLane's functional, followed by removing the minimum number of edges. This allows us to consider the implementation of intersecting connections on PCB as the construction of a new topological drawing with added vertices. Its need for implementation of such algorithms testing on a set of adequate input data of a big size. Such data set often descripts a planar graph. Therefore, it is extremely desirable to have embedding in straight lines on the plane of this graph for preliminary estimate of its properties and comparison with results of testing of tracing algorithms. It is obvious that at small number of vertices of N, the solution of the task is trivial, but at big N and automatic generation of nondirectional flat graphs is desirable to have an algorithm or the program for a solution of such task.

2. Related Works

Denise, Vaskonsellos and Welsh [7] proposed the first algorithm of random generation of flat graphs where the Markov chain has been defined on a great number of planar graphs G_n with n tops. This algorithm is very simple and, apparently, well works at practice. However, it iterates only to uniform distribution so any algorithm execution will surely lead to obtaining heterogeneous results. It is aggravated with the fact that the speed of convergence is unknown. Bodirski, Gr. Opl and Kang [8], [9] developed the second approach on G_n for homogeneous accidental generation. It is based on the recursive method proposed by Nijenhuis and Wilf [8] and formalized by Flajolet, Van Cutsem and Zimmermann [10]. This method involves a precalculation stage where big tables with big coefficients

ICTERI-2021, Vol I: Main Conference, PhD Symposium, Posters and Demonstrations, September 28 – October 2, 2021, Kherson, Ukraine
EMAIL: sergs2222@gmail.com (Sergey Sgadov)
ORCID: 0000-0002-7994-6530 (Sergey Sgadov)

that calculated with use of recursive ratios. Bodirsky and others apply a recursive method on flat graphs that allows well-known combinatory decomposition according to consecutive levels of connectivity. Time necessary for a precalculation stage is big. More precisely, for random generation of flat graphs with vertices (and, perhaps, also with the fixed number of edges), has requirements for time and memory at a precalculation stage up to $O(n^7 (\log n)^2 (\log \log n))$ and $O(n^5 \log n)$. After tables have been calculated, time for each generation is $O(n^3)$. In [8] the algorithm by computing complexity of $O(n^7)$ and the requirement for memory is $O(n^4)$ for the uniform flat graph is proposed. Unfortunately, any of the listed methods does not guarantee inseparability of the graph and also does not build its embedding on to Euclidean plane. Thus, if one want to visualize the graph, they have to use special embedding algorithm realization.

3. The Problem Statement

Despite the large number of publications on the topic of random structures, the author was unable to find among them a method that would simultaneously give a drawing and an adjacency matrix of a plane graph consisting of cycles and satisfying additional conditions. Therefore, one have a problem and it is necessary to generate random single-connected planar graph from N-vertices and simultaneously geometrical drawing of the graph embedded onto the plane (so-called, flat graph). Also it is applied additional requirements - the graph must be nonseparable, that is to represent coherent nondirectional acyclic graph and without multiple edges, without bridges and articulation points, without vertices with local valence of vertices smaller or equal to two. Therefore, it to be aimed to receive the graph in the form of the list of adjacency and the geometrical drawing of the graph at the same time, without using special algorithms on planar graphs embedding . This article considers of a solution of a problem of generation of random flat graphs embedding in the Delphi language.

4. Proposed Algorithm for Flat Graph Generation

Let us declare a vertex as **integer** number, then an edge will be describes as two incident vertices (V_1, V_2). The list **Edges** of edges can describes the graph. In addition, 2D structure G (adjacency list) describes the graph as a list of vertices that are incident to given is declared. This is a two-dimensional structure that represents a list of lists of variable length. The Delphi XE10.4 syntax allows you to implant properties and methods that provide an interface to the structure's fields in such a way that it can be accessed like a two-dimensional array. Also one needs to represent the vertex as a geometric point with coordinates (x, y). The list **Point** is a list of such geometrical representation of graph's vertices.

Here the algorithm of procedure that proposed for random nonseparable flat graph generation (Figure 1).

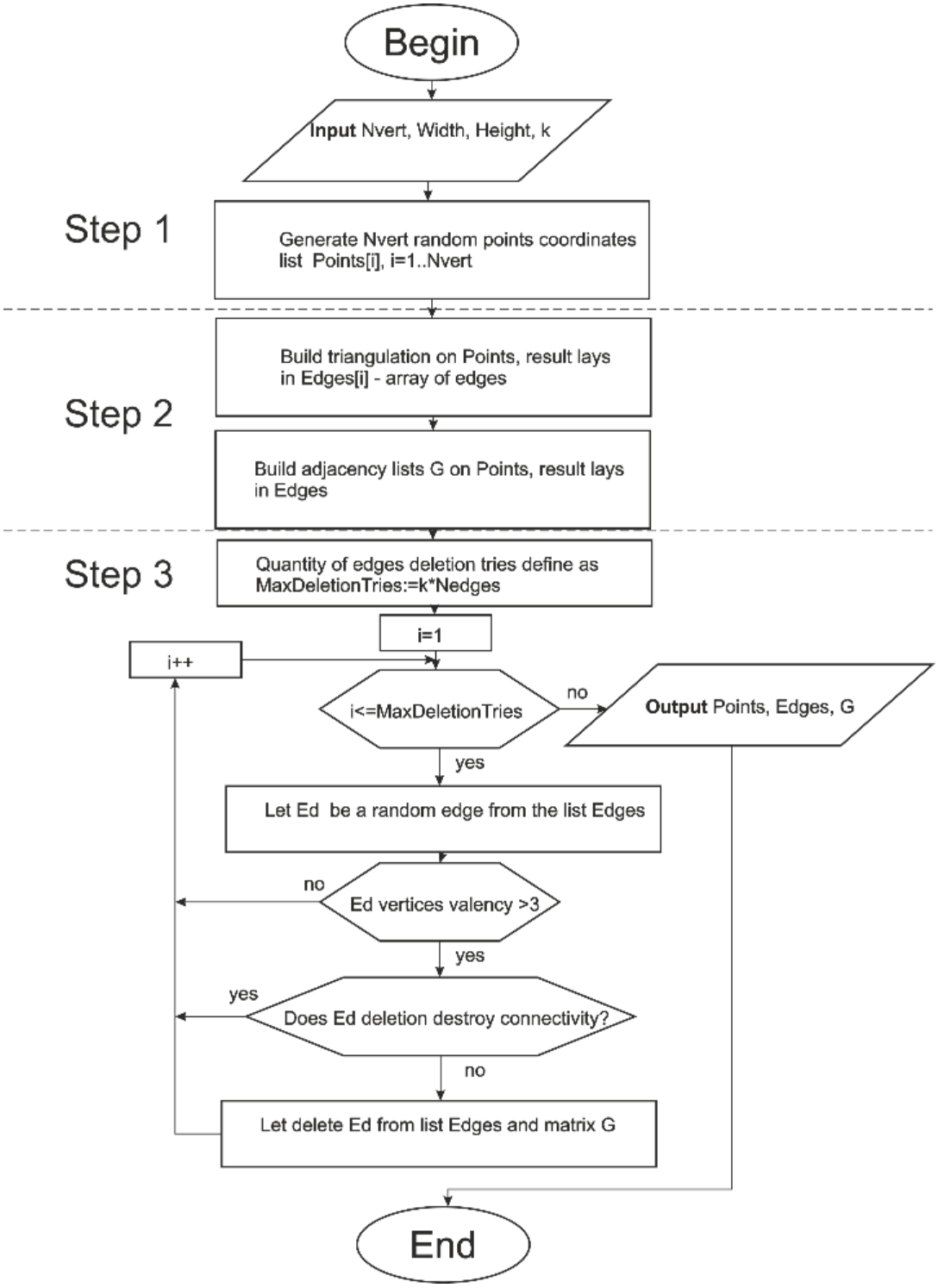

Figure 1: Algorithm of GeneratePlanarTriangGraph procedure for flat graph generation.

In the algorithm the following identifiers are used:

NVert is quantity of tops in the graph,

Width, **Height** plot area sizes,

k - the number of attempts of edges removing as a part from their total quantity.

Result of the algorithm work is the random planar graph presented as the list of adjacency G and Points - coordinate of vertices of graphs embedding onto a plane in straight lines.

Let us consider steps of this algorithm.

Step 1.

Let us set coordinates of borders of two-dimensional rectangular area. Then generate N random uniform distributed points. After this, let us sort the list of points on one of coordinates (Figure 2)

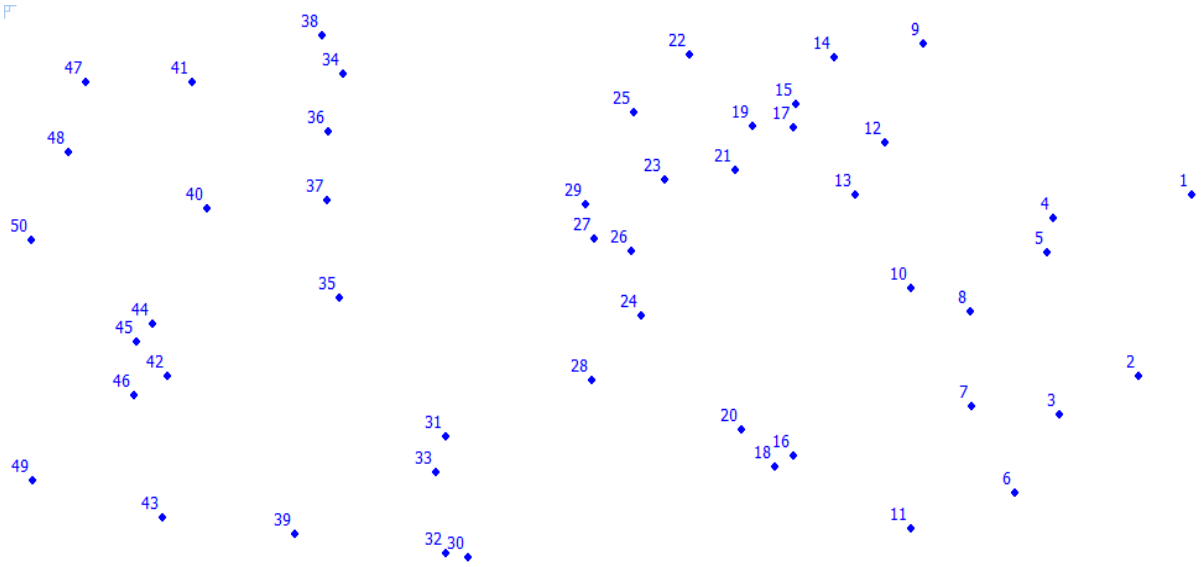

Figure 2: Uniform distributed points

Step 2.

If possible, connect as many N-points as possible with non-intersecting straight lines. For small N, this can be done graphically. but for N more than 20, it makes sense to use special algorithms, for example, Delaney triangulation algorithms. After triangulation is applied, the 2D region will be triangulated. At this stage, a graph is obtained consisting of the same size of triangular faces and it is possible to compose an adjacency matrix of such a graph, because we know how the points are connected (Figure 3).

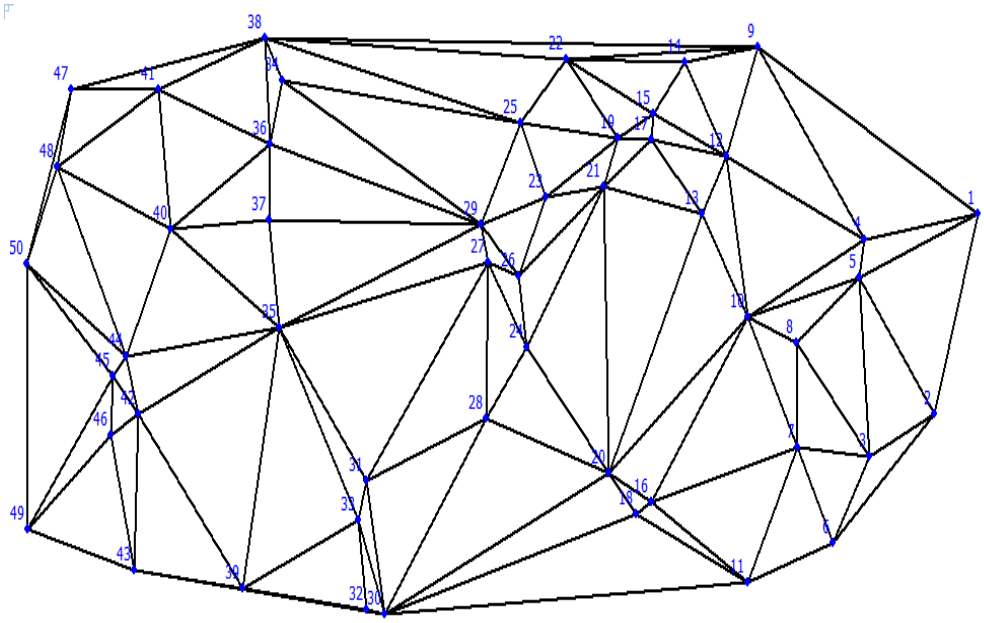

Figure 3: Delaney triangulation result.

Step 3.

Random removal of edges. (Figure 4) Despite the fact that at the previous step we have already obtained a certain random flat graph, it is clearly not enough to solve the problem. Therefore, at this stage, one should try to sequentially remove some predetermined number of random edges. However, if this is done without preliminary verification, then as a result we may find that the resulting graph has obtained properties we do not need, for example, it has not be simply connected, or dangling vertices have appeared. Therefore, before removing an edge, it is necessary to check:

1. is the edge a bridge;
2. is the edge an articulation point;
3. what is the valence of the vertices of the edge (if it is too small, then removing such an edge at best can turn the graph into a tree, and at worst - cause the appearance of dangling vertices).

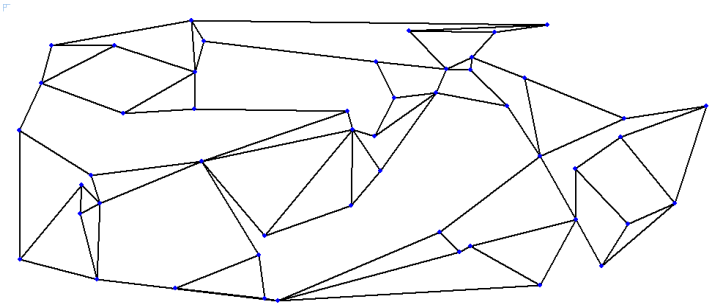

Figure 4: Random flat graph after random edges deletion.

4.1. Triangulation stage

For triangulation, we used the scanline algorithm, which assumes that all points have been sorted by one of the coordinates before, and then, in their order, are processed by the next way:

- Construct a triangle on the first 3 points. Further, for each next point, we will perform the steps based on the fact that there is a Delaunay triangulation for the already added points and, accordingly, a minimal convex hull for them.
- Add triangles formed by visible edges and the point itself (that is, add edges from the point in question to all ends of the visible edges).
- Check for the Delaunay condition all quadrangles generated by visible edges. If the condition is not met somewhere, then we rebuild the triangulation in the quadrangle (recall that there are only two of them) and recursively run the check for quadrangles generated by the edges of the current quadrilateral (because only in them, after changing, the Delaunay condition could be violated).

There is an assumption that the computational complexity of the algorithm is approximately $O(N * \log N)$. It's not worse than $O(N^2)$ in other triangulation methods [11],[12] and does not significantly affect the main algorithm with computational complexity not less than $O(N^2)$. Below there are experimental data that show a quasi-linear dependence of the execution time on N.

4.2. Checking before edges removing

It is necessary to check whether the removal of an edge will lead to a violation of the graph connectivity. This will not happen if it is possible to find a path to one of the vertices that does not contain the given edge. If, as a result of a depth-first search from a vertex of an edge, we can return only through another vertex, then such an edge cannot be deleted. To do this, virtually remove the edge and perform a depth-first search from all vertices that have not yet been in the spanning tree construction. If in this case there is a return to the vertices incident to the removed edge, then return TRUE - the edge can be removed. In case of violation of the graph connectivity, areas are formed that do not belong to the spanning tree, which will be detected. For these purposes, we will create a function (Figure 5)

function bridges_delete_edge(vertexfrom,
vertexto: TVertex;
var G :TAdjMatrix):Boolean,

where vertexfrom, vertexto are the vertices of the tested edge, Gis the adjacency matrix. The function should return true if an edge (vertexfrom, vertexto) can be removed without breaking the connectivity.

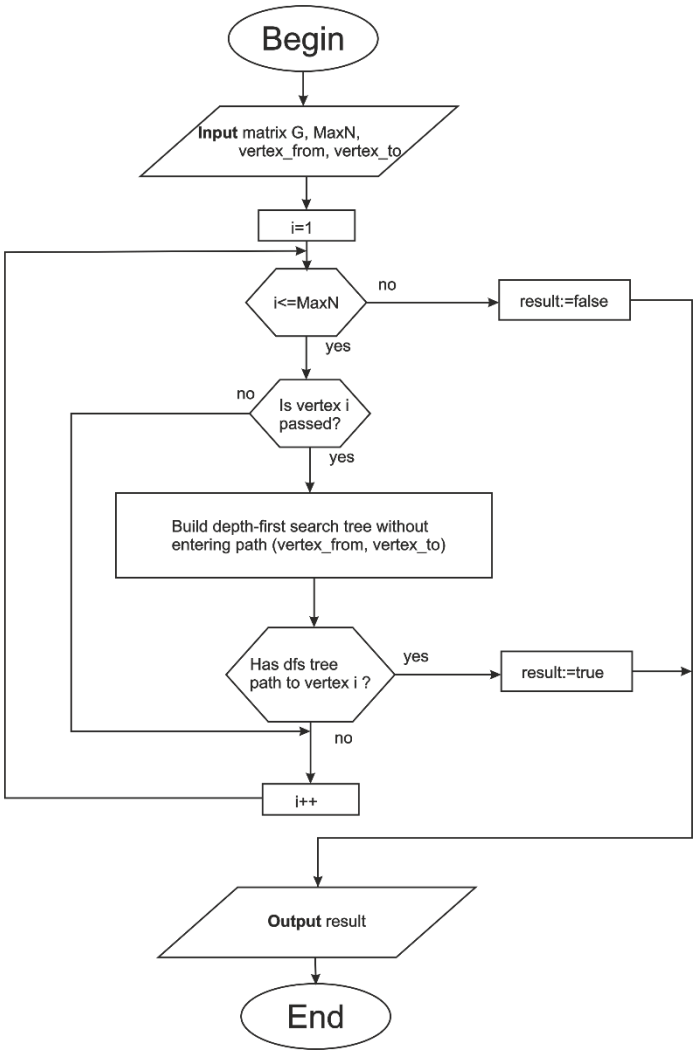

Figure 5: Algorithm of function bridges_delete_edge for edge removing possibility checking.

Similarly way we created a function

function articulation_point_delete_edge(vertexfrom,
vertexto: TVertex;
var g :TAdjMatrix):boolean;

which, using depth-first search, will check whether the removal of vertexfrom, vertexto vertices will lead to the formation of another connected area. Separately, it is necessary to check the local valences of the vertices of the edge in case the edge is included in the chain (then the valencies will be 2 or less). Such edges also should not be removed. In general, at the moment before deleting an edge, you can insert any check whether the graph will have the required properties or not.

5. Experimental Verification of the Proposed Algorithm and Behcmarking.

Algorithm was implemented with Delphi XE10.4 using VCL libraries because of using VCL based classes for graph drawing. Also one had to take account Delphi written extensions for Altium CAD.

Hardware configuration of the system:

- CPU Ryzen 5 1600 (6 physical core, 12 threads) 3700MHz
- Chipset AMD X370
- Memory 8GB DDR4 DRAM 2666MHz
- SSD – Samsung 860 EVO 500GB SATA III

The Win API QueryPerformanceCounter calls were injected into the code of the GeneratePlanarTriangGraph procedure, and the triangulation time and the vertex removal time were measured separately for a different number of vertices N and the deletion part (Table 1, Table 2)

$$k = \frac{[\text{the number of removal attempts }]}{N_{edges}}, \quad (1)$$

(where N_{edges} is the number of edges in triangulation). Due to the fact that the main algorithm has an implicit dependence on the list of edges, it was executed in one thread.

Table 1.

Execution time at k=0.7

N	Triangulation (s)	Edges elimination (s)
30	0.0003319	0.0002721
100	0.0014141	0.0031128
200	0.0033824	0.0132563
500	0.0124573	0.0816445
1000	0.0354214	0.3328531
2000	0.1072302	1.3482507
5000	0.5348077	9.0200086
10000	1.896688	38.0121159

Table 2.

Execution time at k=10

N	Triangulation (s)	Edges elimination(s)
30	0.0003	0.00319
50	0.00055	0.00861
100	0.00131	0.03401
1000	0.03496	3.79863
2000	0.10718	15.338
5000	0.53194	105.515
10000	1.90757	450.353

In addition, the dependence of the execution time (sec) on N (Figure 6) was built.

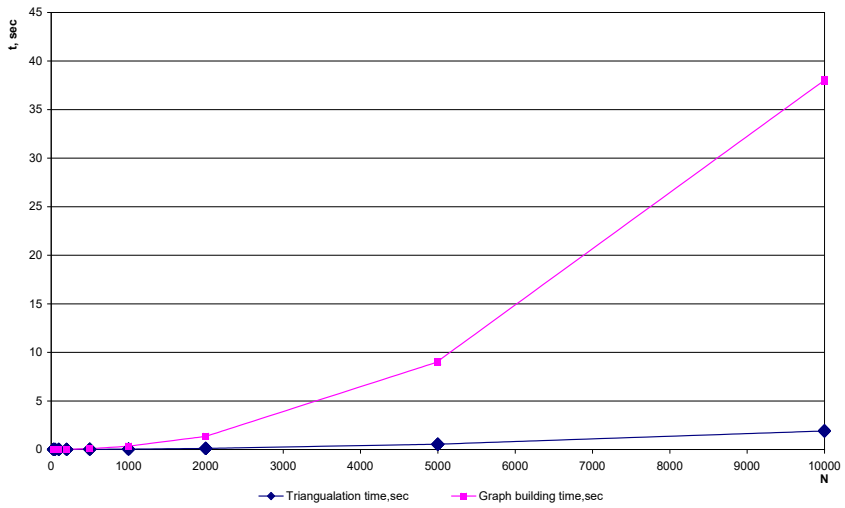

Figure 6: Dependence of the execution time (seconds) on vertex quantity N

The results of the program are shown at Figure 7-9. The adjacency matrix is not given here for obvious reasons.

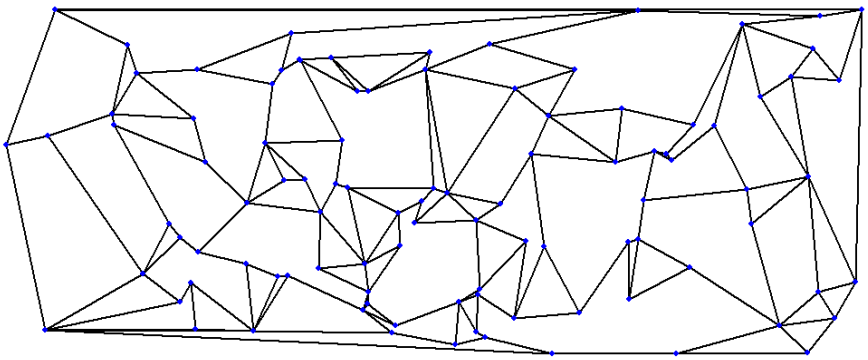

Figure 7: Vertex quantity N=100, deletion part k=0.7

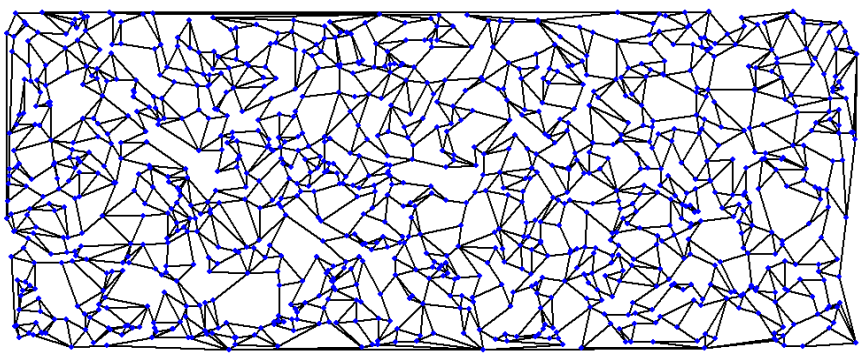

Figure 8: Vertex quantity N=1000, deletion part k=0.7

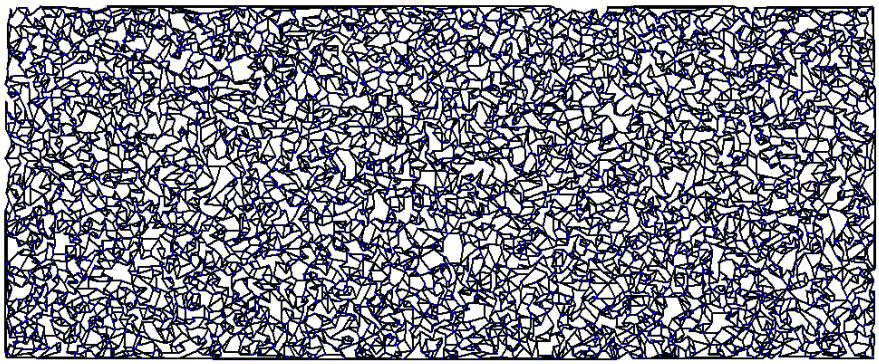

Figure 9: Vertex quantity N=10000, deletion part k=1

If one change the number of deletion attempts to ten times the number of edges in triangulation, then an increase in the size of the faces becomes noticeable (Figure 10-12).

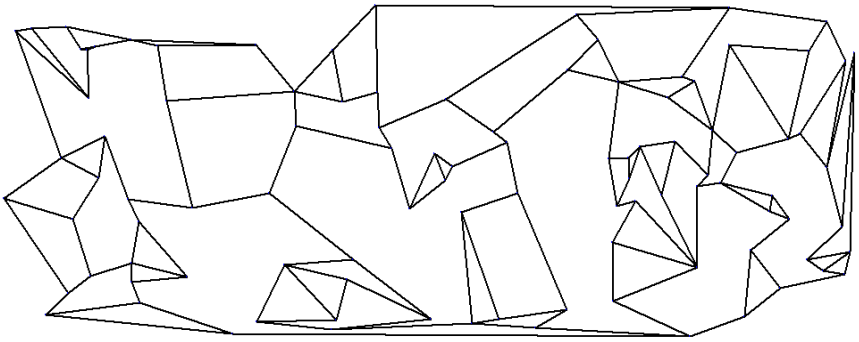

Figure 10: Vertex quantity N=100, deletion part k=10

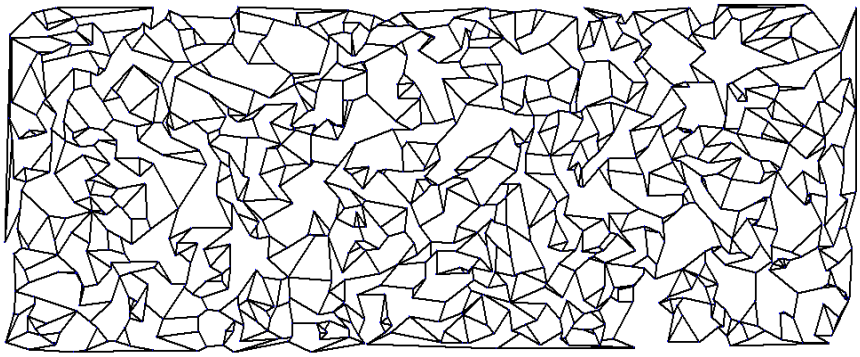

Figure 11: Vertex quantity N=1000, deletion part k=10

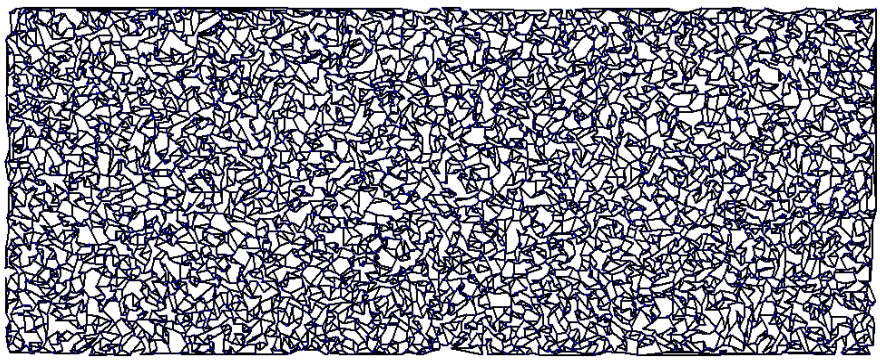

Figure 12: Vertex quantity N=10000, deletion part k=10

One can see (Table 2) that the increase in execution time relative to N remains quadratic, while the increase in productivity from the number of attempts to remove edges appears as linear. Unfortunately,

if the graph is checked for connectivity, it will not allow parallelizing the main algorithm. Therefore, the program was launched in single-threaded mode. Although the elimination of this check immediately reduces the computational complexity by a factor of N, due to the softening of the requirements for the result, it allows parallelization.

6. Conclusion

The scientific novelty of work is that the random flat graph drawing generation algorithm have been proposed that differs its computing complexity of $O(N^2)$ also allows to generate at the same time graph embedding drawing. As show experiments computing complexity of an algorithm approximately quadratic, generally because of checks on connectivity. If one not to be aimed a task to save the connectivity of the graph, then the computing complexity will be quazilinear or linear. Also in this case parallelization of an algorithm will be possible.

The practical value of work is that the software in the Delphi language is worked out that implements the offered algorithm and allows to receive random flat the graph and a matrix (list) of its adjacency. We recommend it for generating test data sets for preplanarisation algorithm implementation. It gives verifiability of programs the implementing algorithms on planar graphs and also to control visually quality and properties of input data for testing as in addition to a connectivity matrix the program gives embedding of the graph onto a plane.

.

7. References

[1] Den'dobren'ko, B. N. , and Malika, A. S. , Automation of Electronic Radio Equipment Design, Vysshaya shkola, Moscow, Russia, (1980).

[2] Bazilevich, R. P. and etc., The Custom VLSI Layout Minimization on the Digital Circuits Topological Design, Control systems and computers, Kyiv, Ukraine, vol 11, pp. 42-50 (2012) URL: http://dspace.nbuv.gov.ua/handle/123456789/83082.

[3] Kureichik, V. V. , Matematicheskoe obespechenie konstruktorskogo i tekhnologicheskogo proektirovaniya s primeneniem SAPR, Radio i Svyaz', Moscow, Russia, (1990).

[4] Kurapov, S. V. , Davidovsky, M. V. , Topological approach to connections conducting in flat formfactor. Komponenty i Tekhnologii, 11(**172**) pp. 127–130, (2015) URL: https://www.elibrary.ru/contents.asp?id=34181322&selid=24863778.

[5] . Kurapov, S. V, Davidovsky, M.V., Algorithms for constructing topological and geometric drawings of the SES graph // Components and Technologies. **6** , pp.128-132, (2017)

[6] Kurapov S.V., Davidovsky M.V., Checking the planarity and building a topological drawing of a plane graph (depth-first search), Prikl. MD, **2 (32)**, pp.100–114, (2016)

[7] Denise, A. , Vasconcellos, M. , and. Welsh, D.J. The random planar graph. Congressus Numerantium, **113** pp. 61–79 (1996).

[8] Bodirskya, M. , Gröpl, C. , Kang, M., Generating labeled planar graphs uniformly at random, Theoretical Computer Science **379(3),** pp. 377–386, (2007). doi:10.1016/J.TCS.2007.02.045.

[9] Nijenhuis, A. ,Wilf, H.S., Combinatorial algorithms, 2rd. ed. Academic Press Inc., New York, San Francisco, London (1978).

[10] Flajolet, Ph., Zimmerman, P., Cutsem ,B. V., A calculus for the random generationof labelled combinatorial structures.Theoretical Computer Science, volume 132(1-2), pp. 1–35, (1994) .

[11] Teplov, A., Maykov, K. , Comparative analysis and modification of algorithms for constructing delone triangulation In: Modern Technologies in Science and Education - STNO-2017 Volume 4 pp.103-106, (2017).

[12] Klyachin V., Triangulation Algorithm Based on Empty Convex Set Condition, Science Journal of Volgograd State University. Mathematics. Physics **3**, vol. 28, pp. 27-33, (2015)