Genetic Algorithm for Drawing Undirected Graph's Layouts

Ekaterina Bochvaroska
Faculty of Mathematics, Natural Sciences and Information Technologies
University of Primorska
Glagoljška ulica 8,
SI-6000 Koper, Slovenia
89221049@student.upr.si

Aleksandar Tošić
Faculty of Mathematics, Natural Sciences and Information Technologies
University of Primorska
Glagoljška ulica 8,
SI-6000 Koper, Slovenia
Innorenew CoE
Izola, Slovenia
aleksandar.tosic@upr.si

Abstract—**In this paper, we present an empirical approach to graph layout improvement using a genetic algorithm, leveraging the power of parallel computing to enhance performance and scalability. We develop an iterative population-based evolutionary search for a random graph layout that combines selections, crossover, and mutation to obtain the best possible outlay of graphs concerning multiple aesthetic criteria. We evaluate our approach using a complete evaluation pipeline and compare the results with the ForceAtlas 2 algorithm. [1] Traditional graph topology algorithms typically suffer from degraded performance when dealing with large graphs but generally produce better results. Thus, using genetic algorithms might be preferable in large networks, where the need for performance is prioritized above the requirement for a detailed visual inspection of the resulting graph. The evolutionary algorithm is guided by a fitness function based on edge-crossing minimization, node distribution, and overall visual clarity. Finally, absent an objective method for measuring how appealing topologies are to the human eye, we provide a visual comparison between our results and commonly used graph layout algorithms.**

I. Introduction

When analyzing networks and visualizing data, it's crucial to create appealing graph layouts and discover attractive topologies. Tutte [2], a pioneer in the field of graph drawing, laid the groundwork for creating visually understandable representations of complex graphs. A well-designed graph provides both a precise and visually appealing representation of data.

Relying on his work Force-directed layout algorithms [1] aim to position the nodes dynamically and apply forces, hence reflecting the structure and relationships making it easier to find patterns and symmetries. However, these force-directed layouts often hit lower local minima quickly, as well as have a high running time. In contrast to this, genetic algorithms (GA) use more direct and adaptive methods to solve a wide range of optimization problems, making them widely popular. They are flexible and can be applied in many different scenarios without requiring specific assumptions about the problem. This work addresses the limitations of conventional methods by using a (GA) approach to improve graph representations. This approach not only aims for better optimization but also ensures faster processing, providing more balanced and visually appealing graphs that can handle larger datasets efficiently, demonstrating the potential of this algorithm in tackling such intricate problems.

Early work on the topic was introduced by "TimGA: A Genetic Algorithm for Drawing Undirected Graphs" [3]. In complement, TimGA proposes a genetic algorithm that is tailored for the layout of undirected graphs, which takes into consideration important aesthetic criteria such as minimizing edge crossings while enforcing an even distribution of node positions and reducing-edge length deviation [4]. Using a genetic algorithm, we seek to refine graph layouts until they are clear and visually appealing. We begin the process with a randomly generated graph, mimicking natural evolution in generating the initial population of graphs, which progressively evolved with selection, crossover, and mutation. A key component here is the quality of a graph layout, which can be evaluated by measuring different graph properties such as the number of edge crossings or how evenly the nodes are positioned across the displaying surface. As a result, the output layouts of the graphs are simplified, which again puts more emphasis on the importance of understanding the complex data. In addition, aiming to provide a global measure of the progress, we used Force Atlas 2 algorithm [1] to estimate the progress of the initial graph in comparison to the outcomes of (GA), serving as a visual reference to determine if any improvement has been achieved.

II. Implementation

In this section, we explain the genetic algorithm's functionality, its implementation for this project, and the methodology used to calculate the fitness function. Our approach is inspired by the principles of natural selection and genetics, we iteratively evolve a population of solutions to optimize graph layouts.

A. Overview of the Genetic Algorithm

The genetic algorithm follows these steps:

1) **Initialization**: We begin with a randomly generated graph based on user-defined sizes of the edge set ($|E|$) and vertex set ($|V|$), represented

by array lists. The initial population of graph individuals is created within the specified panel size, which we fixed to 1000x1000.

2) **Fitness Evaluation**: Evaluate each graph's fitness function score based on criteria such as minimizing edge crossings and evenly distributing nodes. Computationally this is the most expensive part of the algorithm.
3) **Selection**: Like nature employs Darwinism [5], we select the fittest layouts, without employing any changes *-elitism* [6] to serve as parents' layouts to the next generation to ensure the best solutions are preserved.
4) **Crossover**: By combining pairs of parent layouts to create new graphs, we introduce variations in the graph structure.
5) **Mutation**: To preserve genetic diversity, we introduce random changes to certain layouts, similar to how natural variation is essential in evolutionary processes.
6) **Termination**: Repeating the process until a stopping condition is met, such as reaching a maximum number of generations.

B. Fitness Function Calculation

The fitness function serves as a compass, guiding through the optimization journey. It evaluates how well an individual graph layout meets the desired criteria, specifically, it defines the quality of the potential solution. Evaluating through the whole population each graph receives a unique fitness score based on certain properties. Particularly as the algorithm evolves through multiple generations, the fitness function guides the selection of layouts for further improvement where it influences which solutions survive, reproduce, and contribute to the next iterations. Hence the fitness function consists of several components each contributing to finding the optimal solution:

1) Minimum Distance Neighbour Sum = Λ: This component evaluates the sum of the minimum distances between each node and its nearest neighbor, aiming to achieve an even distribution of nodes and reduce clustering within the graph layout. Its high values indicate that nodes are well-separated.

$$\Lambda = \sum_{i=1}^{N} \min_{j \in \text{neighbors}(i)} d(i,j) \tag{1}$$

where $d(i,j)$ is the Euclidean distance between nodes i and j, and N is the total number of nodes.

2) Node Spread = S: This term further distributes nodes by taking into account the number of nodes and the square of the minimum distance between nodes, promoting a wider spread to improve layout clarity. [7]

$$\text{S} = \frac{1}{4} \times N \times (\text{minD})^2 \tag{2}$$

where $minD$ is the Minimum Node Distance

3) Edge Length Deviation Θ : This term measures the deviation of edge lengths from an optimal edge length, which is the smallest edge length found overall in the graph structure to ensure a balanced and symmetric graph layout. [4]Further this value is increased by 5 units, which will again assist in spreading the edges to more even lengths across the graph. This avoids very long or very short edges and gives a much more uniform and aesthetically pleasing layout. By doing so, the lengths of the edges become more consistent, hence reducing clutter and increasing clarity in the graph.

$$\Theta = \frac{1}{|E|} \sum_{e \in E} (d(e) - \mu)^2 \tag{3}$$

where $d(e)$ is the length of edge e, μ is the Optimal Edge length and $|E|$ is the total number of edges.

4) Edge Crossings = X : Edge crossings refer to the number of intersections between edges in a graph layout. Minimizing edge crossings is essential for improving the visual clarity of the graphs' layouts. High numbers of edge crossings can lead to visual clutter, making it difficult to observe relationships and patterns within the data. [4]

$$\text{X} = \sum_{i=1}^{|E|} \sum_{j=i+1}^{|E|} \delta(e_i, e_j) \tag{4}$$

where $\delta(e_i, e_j)$ is 1 if edges e_i and e_j intersect, and 0 otherwise.

5) Combined Fitness Calculation: The total fitness score is a weighted combination of the above equations (1), (2), (3) and (4). The weights are chosen to balance the importance of each criterion, ensuring a well-organized graph layout. The fitness function formula is:

$$\begin{aligned} \text{FitnessScore} = &+ w_1 \times \Lambda \\ &+ w_2 \times \Theta \\ &+ w_3 \times \text{X} \\ &- w_4 \times \left(\frac{\Theta}{\Lambda + \epsilon} \right) \\ &- w_5 \times N \times (\text{minD})^2 \end{aligned}$$

In the code, the weights are:

$w_1 = 2.0$	(Weight for Minimum Distance Neighbour Sum)
$w_2 = 2.0$	(Weight for Edge Length Deviation)
$w_3 = 1.0$	(Weight for Edge Crossings)
$w_4 = 2.5$	(Weight for Edge Length Deviation Penalty)
$w_5 = 0.5$	(Weight for Node Spread)

where ϵ is a small constant added to avoid division by zero. Additionally, we incorporate a check for symmetrical edge crossings, as some symmetrical patterns can be beneficial. If such patterns are detected, the fitness is multiplied by a weight of 1.5.

C. Selection

Individual graphs are probabilistically selected from the population in the selection phase based on their fitness values. This means that better layouts are normally favored, ensuring a better creation of the next generation. Additionally, implementing elitism means that the graphs are firstly sorted based on their fitness scores and then only the top half of individuals are retained in the population. This approach hopefully ensures that the best layouts remain in the next generation leading to continuous improvement and progress.

D. Crossover

Crossover combines genetic material from two parent graphs to create new offspring. This process keeps the genetic pool diverse, which is important for finding better solutions and exploring different possibilities in the population. Usually, this operation requires two parent individuals to produce one offspring. Our implementation includes creating two offsprings to uniformly retain genes that could possibly lead to future improvement in the layout. [8] A crossover point is chosen randomly in the node list, and as a result, the parents' nodes are divided by the separation point. The child graphs are then created by replacing the first parent's node sublist with the second parent's node sublist, effectively exchanging the genetic material. It should be noted that in this process copies of the parents are normally utilized to avoid disrupting the original parents.

E. Mutation

There are two types of mutations that occur in every individual with a probability of 0.1% per graph. The mutation is implemented by randomly selecting a node from the graph, and performing one of two mutation operations on it:

1) *Rotate mutation* is a random rotation of connected nodes or random movement of selected incident edges, preserving their length and connectivity. This exploration of new configurations helps discover optimal layouts.
2) *Flip mutation* is divided into horizontal, vertical, and diagonal flips of a node's position.

The flipping mutation introduces more diversity avoiding local minimum. Figure 2 illustrates a horizontal flip in which node labeled 3 is flipped. One of the three options is chosen with equal probability. Rotational mutations would make it difficult to achieve a similar layout. By flipping the 3rd node we not only improve node distribution but reduce the number of edge crossings by one.

III. Results

Generally, the results from simulations show a significant improvement in layout with relatively fast convergence. Figure 1 shows a chronological time-lapse of the evolutionary process for a random graph with 10 nodes, 15 edges, and a population size of 500. We observe that in the first few hundred generations, most changes to the layout are improving on even spacing of nodes while the last few thousand generations significantly improve on symmetry. One of the components of this optimization apart from the one-point crossover and elitism was also the mutation method used, which introduces random perturbations to the graph layout.

Comparing these results with the state-of-the-art layout algorithm Force Atlas 2 [9] developed by Gephi [10], we offer visually comparable results. However, it is worth noting that Force Atlas 2 does not preserve edge lengths whereas our implementation does.

Furthermore, we measured computation time across various graph sizes—from 100 x 100 nodes and edges to scales reaching up to 6,400 x 6,400 over a generation of 100 graphs. The most computationally expensive operation is the fitness evaluation, for instance, calculating minimum distances between neighbors, edge length deviations, and edge crossings involves iterating through all the edges and nodes which becomes computationally heavier when dealing with larger graph sizes or larger populations. Through parallel programming, which takes advantage of multi-core processors, we distribute computations across multiple threads. [11] We implement this by using the ExecutorService class [1], which manages a pool of threads for concurrent execution, we allow multiple fitness evaluations to run simultaneously. A Semaphore is used to synchronize the completion of the tasks, ensuring all evaluations are completed before proceeding further. Our approach not only delivers efficient computational performance visible in Figure 4 but also ensures scalability for managing larger graphs.

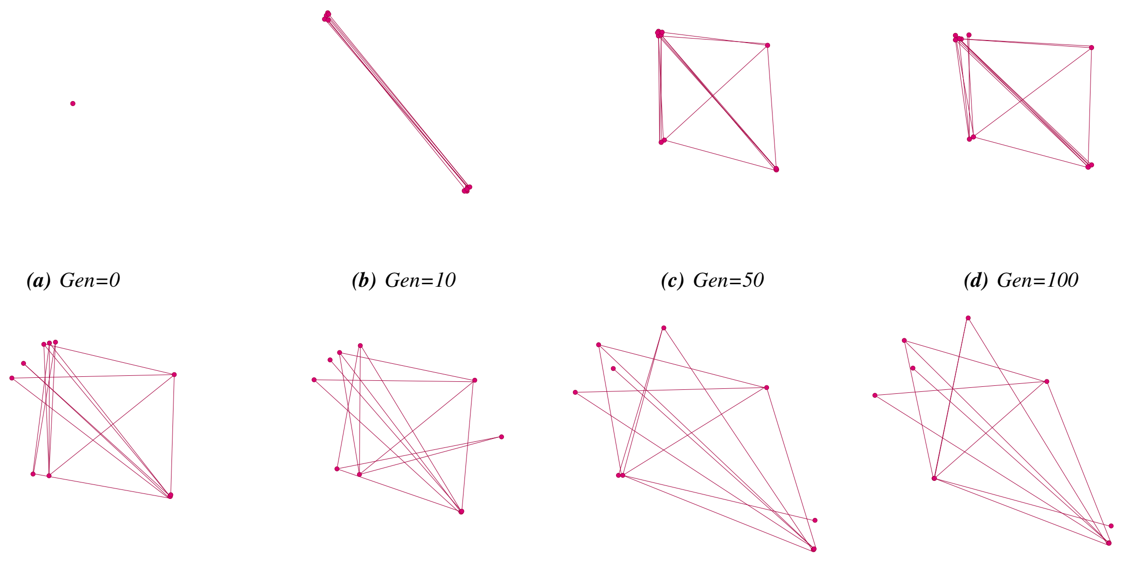
IV. Conclusion

Our research explores a relatively untapped area, as there are few published works on this topic, which are mostly quite outdated. Thus, we significantly improve the quality of the generated graph layout by integrating more precise and complex fitness calculations that take into account elements like minimum distances between neighbors, edge length, edge variations, symmetry, and edge crossings. Our strategic use of different mutations to preserve genetic diversity emphasizes how crucial it is to keep a variety of genes. Moreover, we show that even for large-scale graphs, genetic algorithms could be highly efficient by utilizing parallel computing architectures, especially in the fitness evaluation process. To conclude, our implementation of a more precise fitness function with the help of concurrent programming not only enhances performance but also lays a foundation for more complex graphs with efficiency and scalability, improving the graph layout quality, and opening a way for further research and development in this field.

References

[1] T. Masui, "Evolutionary learning of graph layout constraints from examples," in *Proceedings of the 7th annual ACM Sym-*

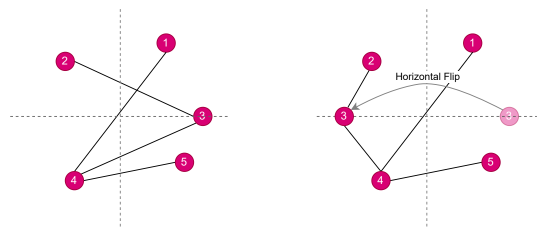
[1]https://github.com/EkaterinaB29/genetic-algorithm-graph-layouts



(*a*) *Gen=0* (*b*) *Gen=10* (*c*) *Gen=50* (*d*) *Gen=100*

(*e*) *Gen=500* (*f*) *Gen=1000* (*g*) *Gen=5000* (*h*) *Gen=10000*

Fig. 1: *Mutation by flipping a node (3) horizontally.*



(*a*) *Before mutation* (*b*) *After mutation*

Fig. 2: *Mutation by flipping a node (3) horizontally.*

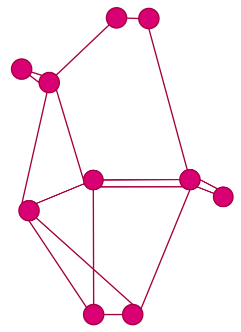


Fig. 3: *Force Atlas 2 layout from Gephi given the same input graph with 10 nodes, 15 edges.*

posium on User Interface Software and Technology, 1994, pp. 103–108.

[2] W. T. Tutte, "How to draw a graph," *Proceedings of the London Mathematical Society*, vol. 3, no. 1, pp. 743–767, 1963.

[3] T. Eloranta and E. Mäkinen, *TimGA: A Genetic Algorithm for Drawing Undirected Graphs*. University of Tampere,

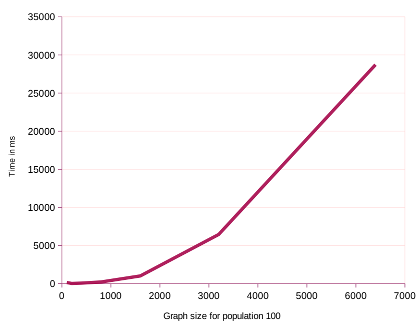


Fig. 4: *Requirement of running time in ms for different graph sizes*

Department of Computer Science, 1996.

[4] A. d. M. S. Barreto and H. J. Barbosa, "Graph layout using a genetic algorithm," in *Proceedings. Vol. 1. Sixth Brazilian Symposium on Neural Networks*. IEEE, 2000, pp. 179–184.

[5] S. J. Gould, "Darwinism and the expansion of evolutionary theory," *Science*, vol. 216, no. 4544, pp. 380–387, 1982.

[6] H. Du, Z. Wang, W. Zhan, and J. Guo, "Elitism and distance strategy for selection of evolutionary algorithms," *IEEE Access*, vol. 6, pp. 44 531–44 541, 2018.

[7] G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis, "Algorithms for drawing graphs: an annotated bibliography," *Computational Geometry*, vol. 4, no. 5, pp. 235–282, 1994.

[8] R. Poli and W. B. Langdon, "Schema theory for genetic programming with one-point crossover and point mutation," *Evolutionary Computation*, vol. 6, no. 3, pp. 231–252, 1998.

[9] M. Jacomy, T. Venturini, S. Heymann, and M. Bastian, "Forceatlas2, a continuous graph layout algorithm for handy network visualization designed for the gephi software," *PloS one*, vol. 9, no. 6, p. e98679, 2014.

[10] M. Bastian, S. Heymann, and M. Jacomy, "Gephi: an open source software for exploring and manipulating networks," in *Proceedings of the international AAAI conference on web and social media*, vol. 3, no. 1, 2009, pp. 361–362.

[11] A. Zamuda, J. Brest, B. Bošković, and V. Žumer, "Differential evolution for parameterized procedural woody plant models reconstruction," *Applied Soft Computing*, vol. 11, no. 8, pp. 4904–4912, 2011.