ISSN 2707-1898 (print)

Український журнал інформаційних технологій
Ukrainian Journal of Information Technology
http://science.lpnu.ua/uk/ujit
https://doi.org/10.23939/ujit2025.01.131
Article received 14.04.2025 p.
Article accepted 01.05.2025 p.
UDC 004.9

Correspondence author
Z. Y. Shpak
zoreslava.y.shpak@lpnu.ua

I. M. Danych, Z. Y. Shpak
Lviv Polytechnic National University, Lviv, Ukraine

METHODS AND TOOLS FOR GRAPH VISUALIZATION IN DYNAMIC SYSTEMS: ANALYSIS AND EXPERIMENTAL STUDY

Graph visualization is a key tool for interpreting complex data in systems with dynamic relationships, such as transportation, computer, or social networks, where the underlying structure is continuously evolving. In such contexts, there is a growing need for adaptive visualization methods that can ensure clear, intuitive, and timely representation of structural changes. This paper analyzes force-directed graph layout methods, which simulate attractive and repulsive physical forces to determine vertex positions in a two-dimensional space, aiming to minimize the system's energy function. A custom software application was developed using the Java programming language, integrated with the Spring framework, the GraphStream library, and JavaFX for visualization. The system provides functionality for implementing, configuring, and comparing the Eades, Fruchterman – Reingold, and Kamada – Kawai algorithms. An experimental study was conducted using various types of graphs from open datasets, particularly Rome-Lib and the Scotch Graph Collection, to evaluate the behavior of each algorithm under different conditions. Results show that the Fruchterman – Reingold algorithm demonstrates smooth and gradual layout transitions, high adaptability to structural changes, and good scalability, making it suitable for real-time dynamic graph visualization, such as traffic monitoring systems. The Kamada – Kawai algorithm provides stable and symmetric but has higher computational complexity and less suitability for interactive scenarios due to abrupt movements of individual vertices. The Eades algorithm performs effectively on sparse or tree-like graphs but tends to produce overly long edges and excessive edge crossings in denser graphs. The developed application supports automatic detection of graph structure changes and restarts the layout algorithm accordingly, enabling near real-time reflection of updates in the visual representation. A promising direction for future research is the integration of neural networks to automate layout evaluation, graph-type classification, and algorithm selection based on specific graph characteristics and task requirements. Such an adaptive approach is expected to enhance the efficiency, clarity, and responsiveness of graph visualizations in dynamic systems, contributing to improved monitoring, analysis, and decision-making based on graph-based models.

Keywords: graph visualization, force-directed methods, online algorithms, dynamic graphs.

Introduction

Data processing and data transformation into useful information is one of the key tasks of modern science and technologies. In the digital age, the value of large corporations largely depends on the volume and quality of accumulated data, which can be utilized for decision-making, trend forecasting, and profit maximization [1], [2].

Graphs are a powerful tool for modeling and analyzing systems with complex interconnections, finding applications in various domains: optimization of transportation networks and analysis of logistics systems [3], [4], design and enhancement of computer network architectures [5], investigation of social relationships and structures [6], as well as analysis and optimization of energy networks [7], among others. In such systems, where the data structure continuously evolves, the ability of graphs to represent these relationships in real time becomes especially important. For instance, in the field of cybersecurity of energy networks, graph models are used to analyze vulnerabilities and protect monitoring systems from attacks that distort real-time results of state estimation [8].

Thus, efficient graph visualization in dynamic systems, where information changes are continuous, is a highly relevant challenge with a broad range of applications. Under such conditions, there is a growing demand for adaptive visualization that allows immediate representation of updated data structures and facilitates their analysis. At the same time, preference of the optimal method for dynamic graph visualization remains an open question, as each algorithm has its own advantages and limitations. To enable testing and comparison of graph visualization algorithms, a software tool was developed to experimentally evaluate the performance of different approaches using practical examples.

The object of this research is the process of graph visualization in dynamic systems.

The research subject includes algorithms and tools for graph visualization in systems with dynamic changes.

The aim of the research is to analyze and investigate graph visualization algorithms, realize software implementation of the best algorithms, compare their efficiency, and assess their suitability for use in dynamic systems.

To achieve this aim, the following primary research objectives have been defined:

- to review existing graph visualization algorithms, analyze them, and select methods for further implementation;
- to implement programmatically the selected algorithms for experimental testing;
- to provide an experimental comparison of the effectiveness of graph visualization algorithms;
- to formulate conclusions regarding the effectiveness of the analyzed methods and to identify future research directions.

Materials and methods of the study. The study used methods of mathematical analysis, in particular, optimization approaches necessary for solving problems of minimizing the energy of graph structures in power visualization algorithms.

The corresponding software application was implemented using the Java programming language. Code dependency management was handled using the Spring Framework, which simplifies component management and increases the system's flexibility. In graph processing implementation were used tools of the GraphStream library.

For algorithm testing, pre-existing sets of test graphs from open sources were used, specifically the Rome-Lib and Scotch Graph Collection datasets [9].

Analysis of recent research and publications. Force-directed algorithms have proven effective for graph visualization, ensuring both stability and adaptability of the layout, as well as smooth real time updates in response to structural changes of the graph. By simulating physical forces of attraction and repulsion, these algorithms generate visually comprehensible layouts with minimal edge crossings, which simplifies the tracking of graph evolution. Studies confirm their ability to maintain dynamic stability and their suitability for online visualization tasks [10], [11].

One of the earliest and most well-known force-directed algorithms is the Eades algorithm, which models a graph as a mechanical system where vertices are represented as steel rings and edges as springs [12], [13]. Initially, the graph's vertices are placed at random positions, after which the system evolves according to physical forces that seek to bring it to a state of minimal energy, where the vertices occupy positions that yield a visually comprehensible layout with approximately equal edge lengths. The algorithm is based on two key principles that define the forces between vertices.

For adjacent vertices connected by edges (springs), the attractive force F_s is defined as a logarithmic function of the distance between the vertices:

$$F_s = C_1 \cdot \log(\frac{d}{C_2}), \quad (1)$$

where d is the distance between the vertices, and C_1 and C_2 are constants. The authors of the algorithm note that using linear springs according to Hooke's law can result in excessive attractive force for distant vertices, leading to layout instability, whereas the logarithmic dependency ensures a smoother force transition [13]. The attractive force equals zero when $d = C_2$, which corresponds to an equilibrium state. The recommended values for the constants are: $C_1 = 2.0$ and $C_2 = 1.0$. The dependency of the force F_s on the distance d for these constants is illustrated in Fig. 1.

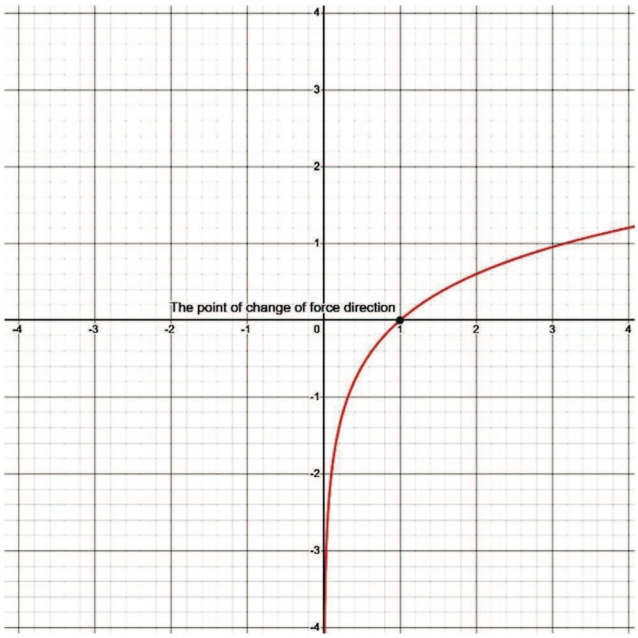


Fig. 1. Plot of the attractive force versus vertex distance for Eades' algorithm

The plot shows that when $d = C_2$ the force is zero, and as C_1 increases, the attractive force grows, allowing control over the "stiffness" of the springs.

For non-adjacent vertices, a repulsive force F_r, is applied, which follows the inverse-square law [13]:

$$F_r = \frac{C_3}{d}, \quad (2)$$

where d is the distance between the vertices, and C_3 is a constant, with a recommended value of $C_3 = 1.0$. This force ensures that non-adjacent vertices are distributed at a sufficient distance from each other, preventing overcrowding. The dependency of the repulsive force F_r on the distance d is shown in Fig. 2. The plot demonstrates that the repulsive force rapidly decreases with increasing distance, and increasing C_3 amplifies the repulsion, enabling finer control over vertex spacing in the layout.

The Eades algorithm follows an iterative process:

1. The vertices of the graph are placed at random initial positions.
2. A total of M iterations are performed, during each of which:
   - the resultant force for each vertex is calculated, taking into account the attractive forces Fs, from adjacent vertices and the repulsive forces Fr from non-adjacent vertices;

- each vertex is moved proportionally to the resultant force, scaled by the displacement coefficient C_4:

$$displacement = C_4 \cdot (F_s + F_r).$$

3. After completing the iterations, the resulting layout is visualized.

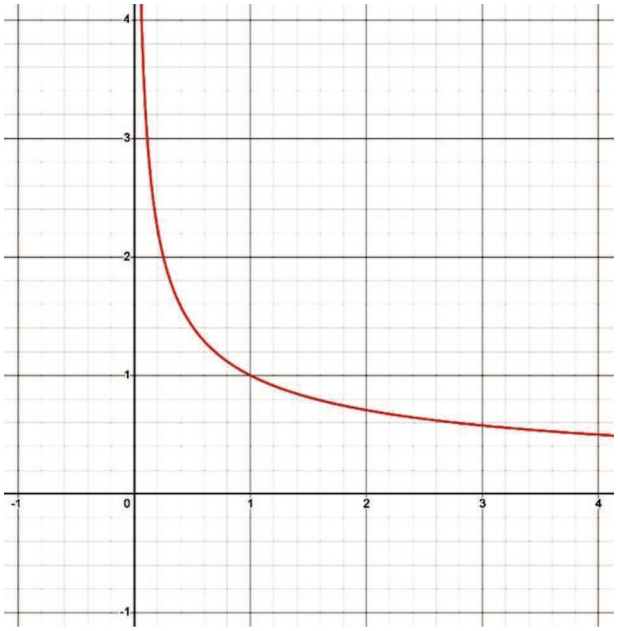


Fig. 2. Plot of repulsive force versus vertex distance for Eades' algorithm

The Eades algorithm performs effectively on many types of graphs, particularly trees and sparse graphs, producing layouts with approximately equal edge lengths and noticeable symmetry. However, it has limitations when applied to dense graphs, graphs with dense subgraphs, or graphs with a small number of bridges (edges whose removal disconnects the graph). In such cases, the algorithm may generate distorted layouts with excessively long edges or numerous edge crossings, which complicates the perception of the graph structure.

One of the improvements based on the ideas proposed by Eades is the Fruchterman-Reingold algorithm [12], [14]. Its primary goal is to create visually clear and expressive layouts that adhere to commonly accepted criteria, such as uniform edge lengths, even distribution of vertices in space, and the display of graph symmetry. The authors reinterpreted Eades' physical model, drawing an analogy to physical systems, such as atomic particles or celestial bodies, that interact through attractive and repulsive forces. As in the Eades algorithm, attractive forces act only between neighboring vertices, while repulsive forces act between all pairs of vertices. However, unlike Eades' logarithmic force functions, Fruchterman and Reingold proposed simpler and more computationally efficient functions for modeling forces [14].

The method introduces the concept of an optimal distance k, which depends on the size of the display area (frame) and the number of vertices in the graph, and is defined by the formula (3):

$$k = C\sqrt{\frac{area}{N}}, \tag{3}$$

where *area* – is the size of the display area, N is the number of vertices in the graph, and C – is an empirically determined constant. This optimal distance is used to define the attractive and repulsive force functions.

The attractive force f_a between neighboring vertices is given by:

$$f_a(d) = \frac{d^2}{k}. \tag{4}$$

The repulsive force f_r between all pairs of vertices is defined as:

$$f_r(d) = \frac{k^2}{d}, \tag{5}$$

where d is the distance between two vertices. The functions proposed by the authors (based on experimental observations) resemble Hooke's law, although they are not exact representations of it: the attractive force increases quadratically with distance, while the repulsive force decreases inversely with distance [14].

In the plot (Fig. 3), the repulsive force is shown in absolute value for clarity: the equilibrium point, where the two forces are balanced, occurs at $d = k$, which corresponds to the ideal distance between vertices.

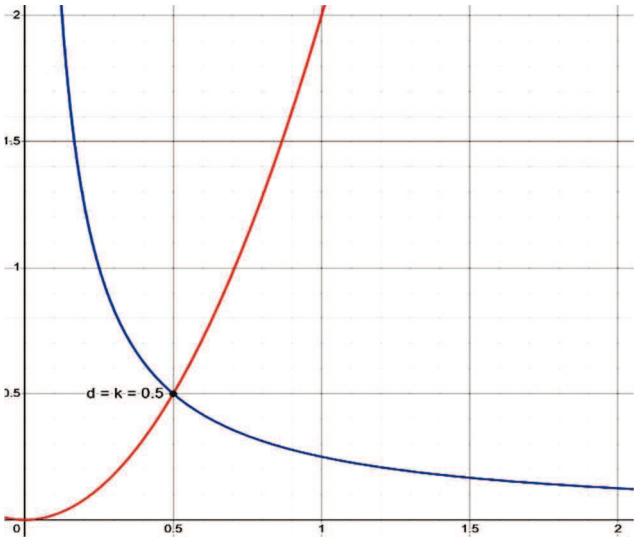


Fig. 3. Plot of attractive (red line) and repulsive (blue line) forces for the Fruchterman – Reingold algorithm with $k = 0.5$

In addition, the algorithm introduces the concept of "temperature", which serves to limit the maximum displacement of vertices during each iteration, thereby ensuring gradual stabilization of the layout [14]. Like the Eades algorithm, the Fruchterman – Reingold algorithm is iterative. Initially, the vertices of the graph are placed at random positions. At each iteration, attractive forces between neighboring vertices are calculated using formula (4), and repulsive forces between all pairs of vertices using formula (5). The vertices are then moved, with their

displacement constrained by the "temperature" parameter, the value of which is gradually decreased over time. The Fruchterman – Reingold method effectively reveals graph symmetry and produces visually appealing layouts, particularly for highly symmetric structures. However, it is prone to getting trapped in local minima and does not guarantee edge-crossing minimization, which can hinder the interpretation of complex graphs.

Another force-directed algorithm is the Kamada – Kawai algorithm, which emphasizes symmetry and even distribution of vertices based on graph-theoretic distance [15]. Unlike the Fruchterman – Reingold algorithm, which uses a global parameter k, Kamada – Kawai determines the theoretical distance between each pair of vertices as the length of the shortest path d_{ij}. The optimal "spring" length between vertices i and j is defined by formula (6):

$$l_{ij} = L \cdot d_{ij}, \tag{6}$$

where L is the base length representing the ideal Euclidean distance between two adjacent vertices and is calculated using formula (7):

$$L = \frac{L_0}{max_{i<j} d_{ij}}, \tag{7}$$

where L_0 is the length of a side of the display area.

The spring stiffness is defined as:

$$k_{ij} = \frac{K}{d_{ij}^2}, \tag{8}$$

where K is a constant. The total system energy E is defined as the sum of the elastic energies for all pairs of vertices, given by formula (9):

$$E = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{1}{2} k_{ij} \left(\lfloor p_i - p_j \rfloor - l_{ij} \right)^2 \tag{9}$$

where $\lfloor p_i - p_j \rfloor$ is the Euclidean distance between vertices. The authors propose using the Newton – Raphson method to find a local minimum of the energy function [15], where only one vertex is moved at each step of the algorithm, while the coordinates of the remaining vertices are "frozen."

The Kamada – Kawai algorithm operates iteratively: initially, the vertices are placed at the nodes of a regular n-gon, after which the shortest path matrix d_{ij}, optimal lengths l_{ij} and stiffness coefficients k_{ij} are computed. At each iteration, the vertex with the highest energy gradient is relocated using the Newton – Raphson method until a specified threshold is reached. The Kamada – Kawai algorithm effectively reveals graph symmetry, but it has high computational complexity and may converge to a local minimum, which can affect layout quality in the case of complex graphs.

Research results and their discussion

For the comparative study of the aforementioned methods, a custom software application for visualizing graph algorithms, developed by the authors, was used [16]. This application was significantly enhanced within the scope of our research: new visualization algorithms (in particular, the Eades and Kamada – Kawai algorithms) were integrated, and special functionality was added for flexible configuration of each algorithm's parameters. These enhancements allow the adjustment of key visualization characteristics, such as attraction force, repulsion force, and spring stiffness in the model.

The software solution was implemented in Java using several modern libraries. Specifically, the Spring framework was used to facilitate dependency management through the inversion of control mechanism; GraphStream served as the core library for working with graphs, providing built-in algorithms; and JavaFX was used for creating the graphical user interface. This technology stack offers a high degree of flexibility, simplifying the integration of new algorithms and the modification of existing functionality.

Graph visualization within the application begins with reading the input graph data and placing the vertices randomly. All integrated visualization algorithms operate in a cyclic manner, gradually optimizing the value of a specific energy function. This enables the creation of interactive animation: the layout of the graph is updated smoothly with each algorithm iteration. To ensure unified integration of various algorithms, a dedicated interface class was developed, which includes a set of common methods, particularly a method for executing a single iteration of the algorithm. This approach ensures compatibility of any algorithm with the visualization module regardless of its specific implementation.

The GraphStream library is based on an event-driven model, which made it possible to implement concurrent execution of computations and rendering. In particular, the algorithm runs in a separate thread and generates events indicating vertex position changes during each iteration, while another thread, responsible for visualization, intercepts these events and updates the graph layout accordingly. The algorithm iterations are executed at a controlled time interval. To ensure smooth and rapid layout generation, this interval was set to 10 ms, creating the impression of a continuous visualization process. It should be noted that this interval determines only the frequency of iteration execution; the actual time required to compute new vertex positions can be much shorter and depends on the complexity of the specific algorithm.

The iterative process continues until the system reaches a specified convergence criterion (minimum energy level). The stopping condition, which may vary depending on the selected algorithm, is encapsulated in a separate method within the aforementioned interface – this provides additional flexibility for integrating new algorithms. Once the layout stabilizes, the system continues monitoring the state of the graph. If any changes in the graph's configuration are detected (such as the addition or removal of elements), the module automatically restarts the iterative search for a new optimal layout in response. The overall functioning scheme of the visualization module is illustrated in Fig. 4.

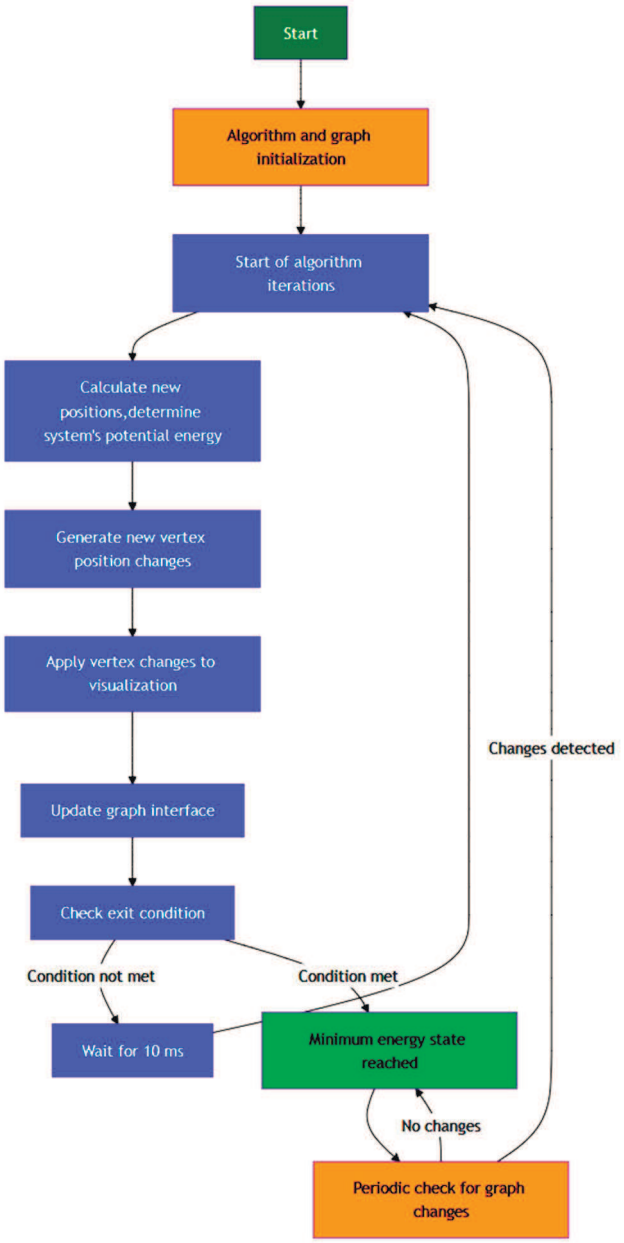


Fig. 4. Diagram illustrating the operation cycle of the graph visualization module with automatic algorithm restart upon detecting changes

As part of the experimental study, three vertex layout algorithms: Fruchterman –Reingold, Eades, and Kamada – Kawai were included into the integrated visualization module. The system enables flexible configuration of all key parameters of these algorithms, which influence the force and energy functions. This provides extensive capabilities for detailed examination of algorithm behavior under various conditions and for selecting optimal settings based on the specific characteristics and parameters of the graphs. Such approach enables algorithm adaptation to task-specific requirements and improves the effectiveness of their use across different scenarios.

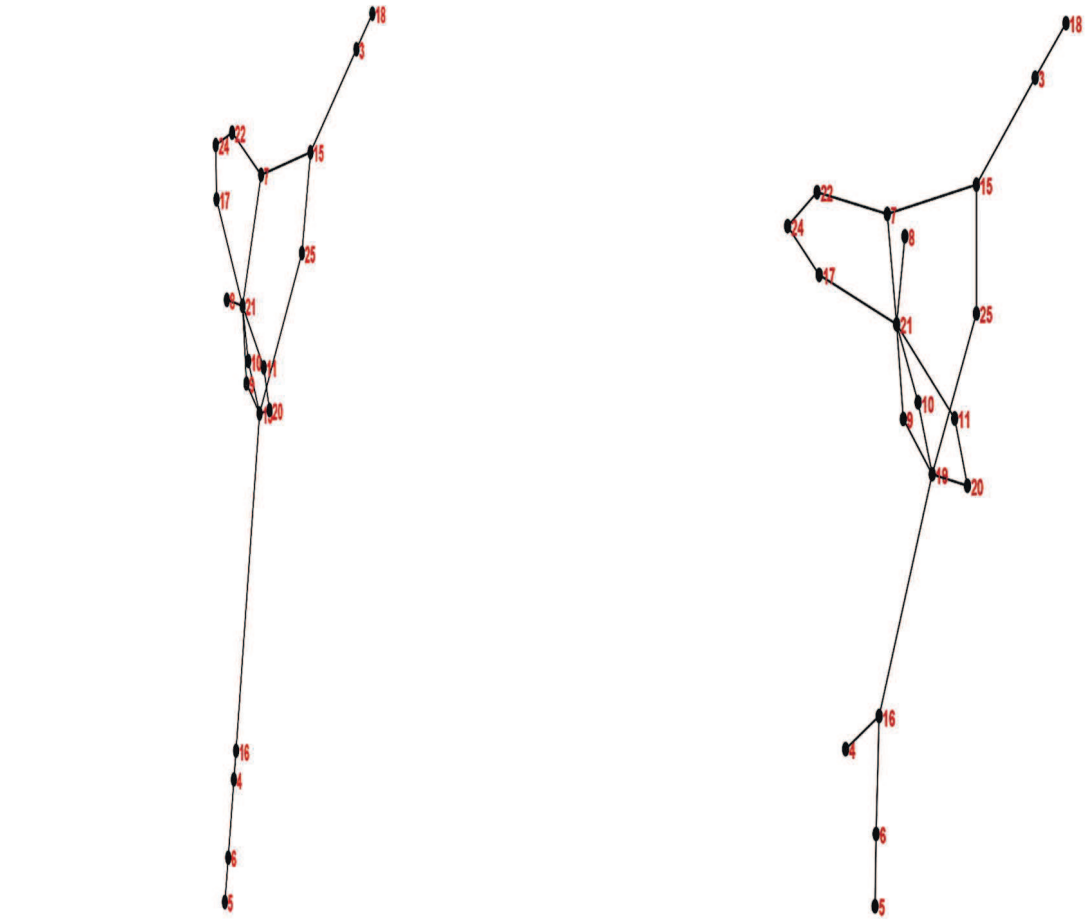
Each algorithm provides options for configuring specific constants and coefficients. In particular:

- *Eades Algorithm:* C_1 – parameter that defines the intensity of attractive forces; C_2 – the distance at which the attractive force becomes zero (equilibrium state); C_3 – parameter that determines the intensity of repulsive forces; C_4 – displacement coefficient that defines how far a vertex moves in proportion to the resultant force.
- *Fruchterman – Reingold algorithm:* K_1 – coefficient for the attractive force (modifies the base attraction formula): – $f_a(d) = \frac{d^2}{K_1}$; K_2 – coefficient for the repulsive force (modifies the base repulsion formula): – $f_r(d) = \frac{K_2^2}{d}$.
- *Kamada – Kawai algorithm:* K – constant that defines the spring stiffness in the model (affects the optimal "spring" length between vertices).

A series of experiments was conducted to evaluate the impact of the aforementioned parameters on the behavior of the algorithms. As an example, Fig. 5 presents the visualization results of the grafo1.26 graph from the Rome Library dataset using the Eades algorithm with different values of parameter C_2. To ensure the validity of the comparison, a random number generator with a fixed seed was used during the random placement of vertices. This eliminated the influence of randomness on the final layout and allowed the effect of changing C_2 to be isolated. The same approach was applied throughout all experiments.

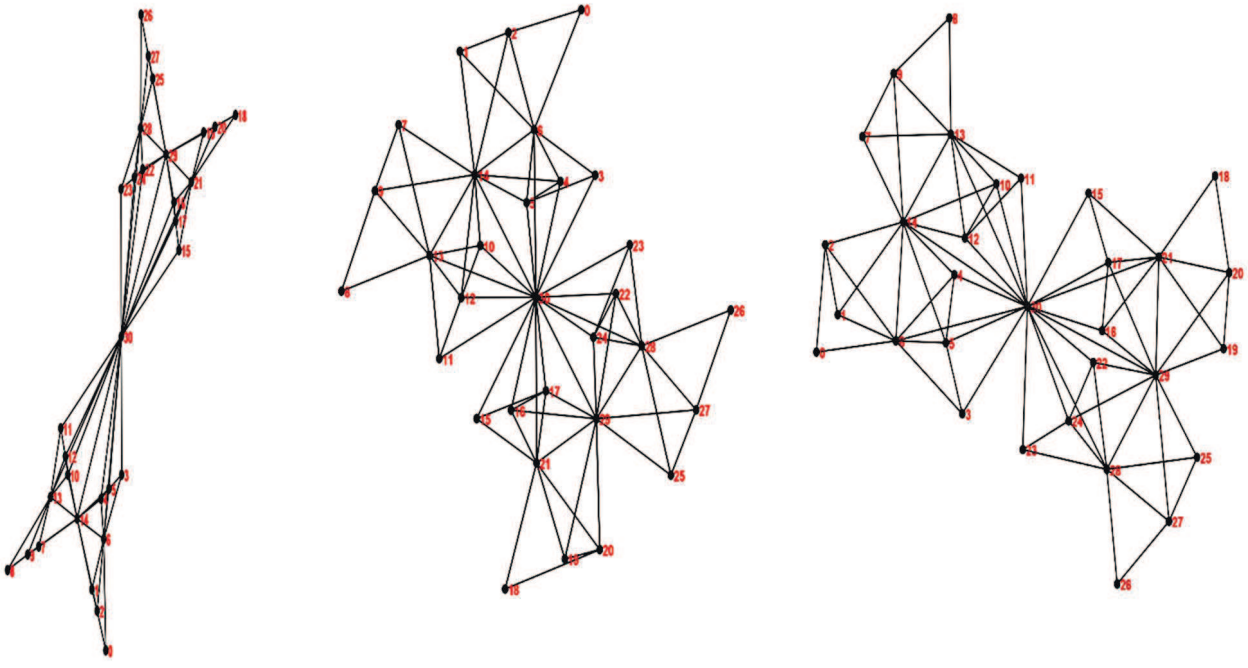
The result of the parametric experiments shows that the parameter C_2 has a significant impact on the layout generated by the Eades algorithm, effectively determining the optimal edge (spring) length in the graph. This can be useful for visualizing hierarchical structures, where longer edges help clearly delineate hierarchy levels and avoid node congestion at lower levels. In contrast, varying the parameters of the Fruchterman – Reingold and Kamada – Kawai algorithms (K_1, K_2 and K) did not demonstrate a noticeable effect on the final vertex layout. It is likely that these algorithms internally normalize or balance the effects of parameters, thereby reducing their overall impact on the outcome.

In addition, a comparative experimental analysis of all three algorithms was conducted to assess their performance and behavior across different types of graphs. Visualization results of two different graph structures (nd_31.78 from the Scotch Graph Collection and grafo1.26 from the Rome Library), generated using the three described algorithms, are presented in Fig. 6. The comparative study confirmed the previously noted drawbacks of the Eades algorithm: it tends to produce excessively long edges and numerous edge crossings, which complicate the perception of the graph's structure. Meanwhile, the layouts generated by the Kamada – Kawai and Fruchterman – Reingold algorithms were relatively similar in terms of vertex positioning. A more detailed analysis revealed that the Fruchterman – Reingold algorithm is more prone to becoming trapped in local minima of the energy function, whereas the Kamada – Kawai algorithm provides more stable results and is less susceptible to local minima.



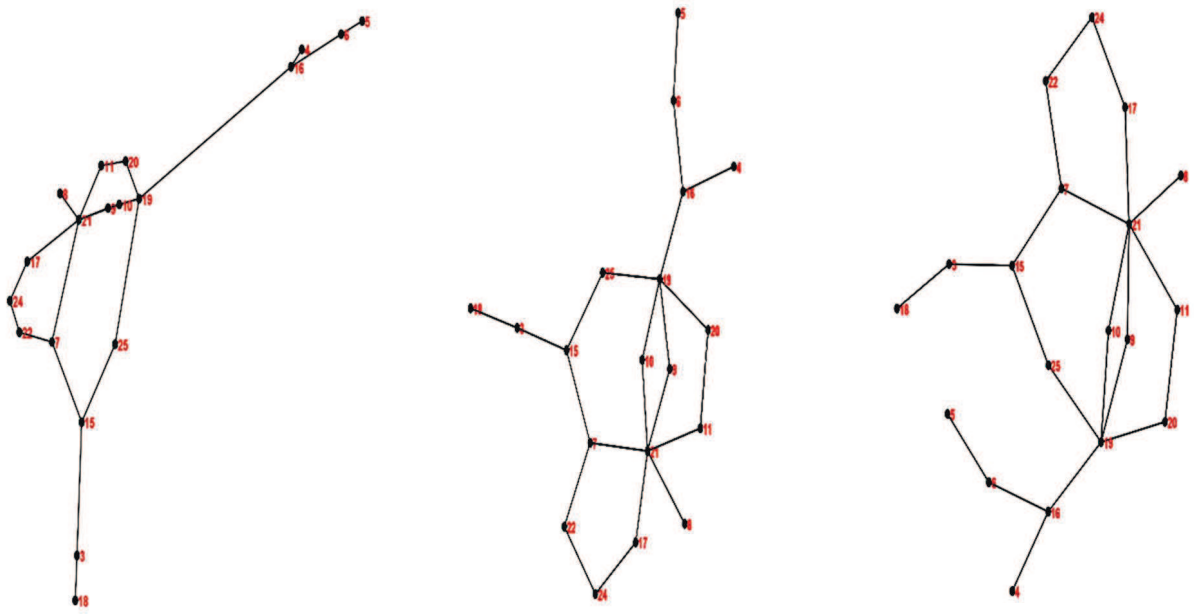
a *b*

Fig. 5. Visualization results of the grafo1.26 graph from the Rome Library dataset using Eades' algorithm with different C_2 values: a – $C_2 = 1.0$; b – $C_2 = 4.0$



a *b* *c*

Fig. 6. Visualization results of two graphs: nd_31.78, Scotch Graph Collection (images a, b, and c) and grafo1.26, Rome Library (images d, e, and f), using three algorithms: Eades (images a and d), Kamada – Kawai (images b and e) and Fruchterman – Reingold (images c and f)



d *e* *f*

Fig. 6. *(Continuation)*Visualization results of two graphs: nd_31.78, Scotch Graph Collection (images a, b, and c) and grafo1.26, Rome Library (images d, e, and f), using three algorithms: Eades (images a and d), Kamada – Kawai (images b and e) and Fruchterman – Reingold (images c and f)

Another important observation concerns the suitability of these algorithms for dynamic (interactive) visualization. The Kamada – Kawai algorithm moves only one vertex per iteration, attempting to locally minimize the system's energy. As a result, the position of this vertex may change quite abruptly from one iteration to the next, which negatively affects the smoothness of real-time layout updates. In contrast, the Fruchterman – Reingold algorithm updates the positions of all vertices gradually, ensuring a smoother and more consistent change in the graph's configuration. This leads to a visually "softer" layout evolution and quicker adaptation to changes in graph structure – an essential feature for dynamic systems. Moreover, experiments showed that the Fruchterman-Reingold algorithm scales better to larger graphs, especially under frequent structural changes, whereas the Kamada – Kawai algorithm proved more effective for relatively small but densely connected graphs. These characteristics make it possible to choose the most appropriate algorithm based on the size and density of the graph in dynamic visualization tasks.

The conducted research allowed us to identify and assess the advantages and limitations of each visualization algorithm under specific conditions. In particular, the Fruchterman – Reingold algorithm is recommended for visualizing dynamically changing networks (e. g., real-time monitoring of transportation networks), as it can smoothly adapt the layout to continuous route changes and provides a clear representation of the current state. The Kamada – Kawai algorithm, owing to the stability of its resulting layouts, is more effective for static analysis of complex networks (e. g., gene interaction models in biology), where it is important to accurately convey distances between connected entities. Despite the limitations mentioned earlier, the Eades algorithm can still be useful for visualizing sparse graphs (e. g., hierarchical tree-like structures), provided its parameters are configured to minimize the number of edge crossings.

Discussion of research results. The results obtained through the conducted experimental studies provide a detailed evaluation of the effectiveness and characteristics of fundamental methods and algorithms for two-dimensional dynamic graph visualization. The following outlines the scientific novelty of the study and the practical significance of the findings.

Scientific novelty of the obtained results. For the first time, a detailed comparative analysis was conducted on the behavior of three core graph visualization algorithms (Eades, Fruchterman – Reingold, and Kamada – Kawai) across different graph types, including scenarios involving dynamic changes. This made it possible to delineate the areas where each algorithm is most effectively applied and to assess how their parameters influence visualization quality.

A flexible graph visualization software module was developed using modern computing technologies, designed for the integration of various algorithms and supporting adaptive configuration of their parameters according to specific task requirements. Unlike previous solutions, which were limited to a fixed set of algorithms and standard configurations, the proposed application implements a universal approach to managing algorithms in dynamic environments.

Practical significance of the research results. The obtained results have significant practical potential for solving applied tasks related to the visualization of structures and processes in dynamic systems, as they can serve as a foundation for making informed decisions about the optimal visualization algorithm for each specific case.

The developed graph visualization software module can form the basis of interactive monitoring and analysis tools, where it is important to display the structure and changes of

complex systems in real time (e. g., transportation networks). Thus, the research outcomes open up new opportunities for applying graph visualization methods in practice, both for improving existing software systems and for developing new specialized systems in various domains.

Future research may focus on developing an adaptive graph visualization algorithm capable of automatically selecting the optimal method and its parameters based on the characteristics of input data and the task requirements. Implementing such an approach will require extensive experimental research, including comparative analysis of different algorithms on a wide range of graphs and the establishment of evaluation criteria for layout quality. A promising solution involves the use of neural networks, which are expected to enable assessment of the quality and interpretability of the resulting layouts, collection of comparative performance statistics, and automated selection of the most appropriate visualization algorithm for a given scenario. This adaptive approach opens substantial opportunities for automating the graph visualization process and improving its quality and efficiency in dynamic applications.

Conclusions

The conducted research enabled a critical analysis and experimental comparison of the main force-directed graph visualization algorithms: Eades, Fruchterman – Reingold, and Kamada – Kawai in the context of dynamic systems. The Java-based visualization module developed with flexible parameter configuration enabled the comprehensive investigation and comparative testing of these algorithms using extensive open datasets such as Rome-Lib and the Scotch Graph Collection.

The experiments demonstrated that the Fruchterman – Reingold algorithm produces optimal layouts for large graphs with frequent configuration changes due to its fast and smooth layout updates. The Kamada – Kawai algorithm stands out for its stability and ability to avoid local minima but is less suitable for interactive visualization because of potentially abrupt vertex movements. The Eades algorithm, in turn, can be effectively applied to visualizing sparse graphs and hierarchical structures but is not well-suited for dense graphs due to its tendency to produce excessively long edges and numerous crossings.

Based on these findings, recommendations were formulated for selecting algorithms depending on graph type and visualization requirements, particularly for monitoring tasks involving transportation or hierarchical systems. A promising direction for future research is the development of an adaptive algorithm using neural networks to select automatically the optimal methods and parameters. The results obtained hold both scientific novelty and practical value for real-time visual analysis of dynamic systems.

References

[1] Birch, K., Cochrane, D. T., & Ward, C. (2021). Data as asset? The measurement, governance, and valuation of digital personal data by Big Tech. *Big Data & Society*, *8*(1), 115. https://doi.org/10.1177/20539517211017308

[2] Gupta, S., Justy, T., Kamboj, S., Kumar, A., & Kristoffersen, E. (2021). Big data and firm marketing performance: Findings from knowledge-based view. *Technological Forecasting and Social Change*, *171*, Article 120986. https://doi.org/ 10.1016/ j.techfore.2021.120986

[3] Boeing, G. (2025). Modeling and analyzing urban networks and amenities with OSMnx (Working paper). University of Southern California. Retrieved from: https://geoffboeing.com/share/osmnx-paper.pdf

[4] Rathore, M. M., Shah, S. A., Awad, A., Shukla, D., Vimal, S., & Paul, A. (2021). A cyber-physical system and graph-based approach for transportation management in smart cities. *Sustainability*, *13*(14), Article 7606. https://doi.org/ 10.3390/ su13147606

[5] Anisa, R., Prihandini, R. M., Alvina, D. A. R. J., Makhfudloh, I. I., Agatha, A. B., & Wulandari, Y. N. (2024). Application of graph theory in computer network optimization. Retrieved from: https://www.researchgate.net/ publication/ 381490742_Application_of_Graph_Theory_in_ Computer_Network_Optimization

[6] Logan, A. P., LaCasse, P. M., & Lunday, B. J. (2023). Social network analysis of Twitter interactions: A directed multilayer network approach. *Social Network Analysis and Mining*, *13*(65). https://doi.org/10.1007/s13278-023-01063-2

[7] Aksoy, S. G., Purvine, E., Cotilla-Sanchez, E., & Halappanavar, M. (2018). A generative graph model for electrical infrastructure networks. *Journal of Complex Networks*, *7*(1), 128–162. https://doi.org/10.1093/comnet/cny016

[8] Bi, S., & Zhang, Y. J. A. (2016). Graph-based cyber security analysis of state estimation in smart power grid. *IEEE Communications Magazine*, arXiv preprint arXiv:1612.05878. https://arxiv.org/abs/1612.05878

[9] *Graph Layout Benchmark Datasets*. Retrieved March 26, 2025, from: https://visdunneright.github.io/gd_ benchmark_sets/

[10] Beck, F., Burch, M., Diehl, S., & Weiskopf, D. (2014). The state of the art in visualizing dynamic graphs. In R. Borgo, R. Maciejewski, & I. Viola (Eds.), *Eurographics Conference on Visualization (EuroVis) – STARs*. The Eurographics Association. https://doi.org/10.2312/eurovisstar.20141174

[11] Cheong, S.-H., Si, Y.-W., & Wong, R. K. (2021). Online force-directed algorithms for visualization of dynamic graphs. *Information Sciences, 556*, 223–255. https://doi.org/ 10.1016/ j.ins.2020.12.068

[12] Kobourov, S. G. (2013). Force-directed algorithms for schematic drawings and placement: A survey. In *Handbook of Graph Drawing and Visualization* (pp. 383–408). CRC Press. https://doi.org/10.1016/j.ins.2020.12.069

[13] Cheong, S.-H., & Si, Y.-W. (2020). Force-directed algorithms for schematic drawings and placement: A survey. *Information Visualization, 19*(1), 6591. https://doi.org/ 10.1177/ 1473871618821740

[14] Eades, P. (1984). A heuristic for graph drawing. *Congressus Numerantium, 42*, 149–160. Retrieved from https://www.cs.ubc.ca/~will/536E/papers/Eades1984.pdf

[15] Fruchterman, T. M. J., & Reingold, E. M. (1991). Graph drawing by force-directed placement. *Software: Practice and Experience, 21*(11), 1129–1164. https://doi.org/10.1002/spe.4380211102

[16] Kamada, T., & Kawai, S. (1989). An algorithm for drawing general undirected graphs. *Information Processing Letters, 31*(1), 715. https://doi.org/10.1016/0020-0190(89)90102-6

[17] Danych, I., & Shpak, Z. (2024). Practical and educational application for interaction with graphs. In *Proceedings of the 1st International Scientific and Practical Conference "Computational Intelligence and Smart Systems" (CISS-2024)* (pp. 37–39). Lviv: ATB Publishing.

І. М. Данич, З. Я. Шпак
Національний університет "Львівська політехніка", Львів, Україна

АНАЛІЗ ТА ЕКСПЕРИМЕНТАЛЬНІ ДОСЛІДЖЕННЯ МЕТОДІВ І ЗАСОБІВ ДЛЯ ВІЗУАЛІЗАЦІЇ ГРАФІВ У ДИНАМІЧНИХ СИСТЕМАХ

Візуалізація графів є важливим інструментом для інтерпретації складних даних у системах із динамічними зв'язками, таких як транспортні, комп'ютерні чи соціальні мережі, де структура даних постійно змінюється. У таких умовах особливо актуальне завдання створення адаптивних методів, здатних забезпечити чітке, зрозуміле й своєчасне відображення змін у структурі графа. У роботі проаналізовано силові методи візуалізації графів, які моделюють сили притягування та відштовхування для пошуку позицій вершин графа у двовимірному просторі, що мінімізує сумарну енергетичну функцію системи. Розроблено програмний застосунок із використанням мови програмування Java, фреймворку Spring, бібліотеки GraphStream та засобів JavaFX для інтеграції, реалізації, налаштування та порівняння алгоритмів Ідса, Фрюхтермана – Рейнгольда та Камада – Каваї. Виконано експериментальне дослідження, у межах якого протестовано поведінку кожного алгоритму на різних типах графів із відкритих наборів даних, зокрема Rome-Lib і Scotch Graph Collection. Виявлено, що алгоритм Фрюхтермана – Рейнгольда забезпечує плавні й поступові зміни макета, високу адаптивність до змін і належну масштабованість, тому він ефективний для динамічної візуалізації, зокрема у системах моніторингу транспорту. Алгоритм Камада – Каваї забезпечує стабільність макета, однак має високу обчислювальну складність і менш наочний у разі інтерактивної візуалізації змін конфігурації графа через різкі переміщення окремих вершин. Алгоритм Ідса придатний для роботи з розрідженими графами або деревоподібними структурами, але поступається за якістю під час роботи зі щільними графами. Розроблена система підтримує автоматичне реагування на зміну структури графа та повторний запуск алгоритму, що дає змогу відображати актуальний стан макета в режимі реального часу. Запропоновано напрям подальших досліджень, пов'язаний із використанням нейронних мереж для автоматичної оцінки якості макетів, класифікації типу графа та вибору оптимального алгоритму й параметрів його роботи. Очікується, що це підвищить ефективність, точність і зручність візуалізації графів у динамічних системах, забезпечивши нові можливості для моніторингу, аналізу та прийняття рішень на основі графових моделей.

Ключові слова: відображення графів, математичні моделі, силові методи, алгоритми реального часу, динамічні графи.

Інформація про авторів:

Данич Іван Миколайович, аспірант, кафедра автоматизованих систем управління. **Email:** ivan.m.danych@lpnu.ua; https://orcid.org/0000-0003-2533-2039

Шпак Зореслава Ярославівна, канд. техн. наук, доцент, кафедра автоматизованих систем управління. **Email:** zoreslava.y.shpak@lpnu.ua; https://orcid.org/0000-0003-4375-2985

Цитування за ДСТУ: Данич І. М., Шпак З. Я. Аналіз та експериментальні дослідження методів і засобів для візуалізації графів у динамічних системах. *Український журнал інформаційних технологій.* 2025, т. 7, № 1. С. 131–139

Citation APA: Danych, I. M., & Shpak, Z. Y. (2025). Methods and tools for graph visualization in dynamic systems: An experimental study. *Ukrainian Journal of Information Technology,* 7(1), 131–139. https://doi.org/10.23939/ujit2025.01.131