Efficient machine learning based graph layout calculation for investigation platform

Alen Granda
alen.granda@student.um.si
Faculty of Electrical
Engineering and Computer Science,
University of Maribor
Koroška cesta 46
SI-2000 Maribor, Slovenia

Niko Lukač
niko.lukac@um.si
Faculty of Electrical
Engineering and Computer Science,
University of Maribor
Koroška cesta 46
SI-2000 Maribor, Slovenia

Aleksander Pur
aleksander.pur@policija.si
Ministry of the Interior
Štefanova ulica 2
SI-1000 Ljubljana, Slovenia

Štefan Kohek
stefan.kohek@um.si
Faculty of Electrical
Engineering and Computer Science,
University of Maribor
Koroška cesta 46
SI-2000 Maribor, Slovenia

Abstract

Modern web technologies enable interactive visualization of graphs in a web browser. However, larger graphs, which are common in investigation data, bring numerous technical limitations in terms of transfer bandwidth and application responsiveness. In this paper we propose a machine learning based method to efficiently transfer graphs' data between the client and the server. Graphs are stored in the investigation platform database on the server and are transferred to web browser on the client for interactive visualization and manipulation. For this reason, we utilize a concept called graph embedding. The main aim is to offer responsive web application due to less complex layout calculation algorithm on the client, less bandwidth requirements and incremental visualization of nodes during graph transfer. We performed distinct experiments, which demonstrate faster graph visualization on the client. We perceived up to 130 times faster layout calculation on the client compared to standard force graph algorithms with maximum 1000 nodes, while preserving adequate visualization accuracy.

Keywords

machine learning, graph embedding, graph visualization, investigation platform, web application

ACM Reference Format:

Alen Granda, Niko Lukač, Aleksander Pur, and Štefan Kohek. 2022. Efficient machine learning based graph layout calculation for investigation platform. In *Proceedings of Student Computing Research Symposium (SCORES'22).* ACM, New York, NY, USA, 4 pages. https://doi.org/XXXXXXX.XXXXXXX

SCORES'22, October 6, 2022, Ljubljana, Slovenia

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

Data representation using graphs is nowadays widely adopted standard at any form of applications [4]. Graphs provide natural and intuitive method of presenting enormous amounts of research data (e.g. bank transactions between entities, relations on social media), which daily increase in size. In this paper, the main focus is on modern web applications enriched with graphs, which are the basis of research platforms. Example of such application is an advanced investigation platform [13], which facilitates investigations by visualizing nodes with their relations using graphs. We extend the platform by utilizing suggested approach to facilitate several drawbacks, which are discussed in succeeding paragraph.

For graph visualization, most common approach in web applications is D3.js library [2], which uses Dwyer's proposition of incremental graph layout calculation. It computes incremental layout with complexity of $O(|V|\log|V| + |E|)$ per iteration where V is a set of vertices and E a set of edges [3]. This algorithm is computationally demanding, which is noticeable especially at the side of low-power clients (e.g. mobile phones). Therefore, web application may suffer from unresponsiveness. Another difficulty is also lower bandwidth in comparison to local desktop applications. To construct a layout, algorithm expects complete graph data to be available, which prevents incremental visualization of graphs and therefore responsiveness of the web application. By application of our proposition, aforementioned problems can be alleviated due to simultaneous data transfer and graph construction.

This problem has been extensively studied in the literature in terms of time and space complexity. Gove [6] recently presented random vertex sampling algorithm running at $O(|V|)$ time and $O(|V|^{\frac{3}{4}})$ auxilliary space. With combination of parallelization, speedup ratio of 26.7% may be achieved [17]. In addition, by using specialized method for graph representation termed Log (graph) [1], high compression ratios with minimal cost decompression can be realized.

In this paper we propose machine learning-based graph layout calculation for web applications. Given nodes and edges, the

server application trains neural network based on expected node coordinates calculated in advance. For the visualization in the web application on the client, only the weights of the neural network, graph nodes and graph edges are transferred from the server. The client is provided with the same neural network as the server. After having received the weights, the client is capable of incrementally constructing the layout by simultaneously receiving and predicting received node's coordinates, resulting in improved responsiveness compared to the standard layout calculation algorithms, which require complete graphs to be available in advance.

The experiments were performed on an advanced investigation platform data [13]. Results were compared to baseline visualization algorithms and assessed using graphs of accuracy and time complexity. We indicated lower bandwidth requirements with acceptable accuracy. In addition, faster graph construction on the client-side was demonstrated.

Below, the main contributions of the paper are outlined.

- Efficient procedure of conveying graph data from the server to the client is provided.
- Graph visualization workflow is designed, which breaks dependency between the nodes for the layout calculation and therefore enables incremental visualization of graphs.
- A methodology is delivered, which increases responsiveness of web applications on the client when visualizing enormous amounts of nodes and connections because of reduced visualizaton algorithm complexity.
- An approach of node coordinate prediction based on the node embedding method called random walk is introduced.

2 Layout construction

Proposed algorithm takes advantage of machine learning (ML) to predict node coordinates in the graph. However, graphs are known to hardly incorporate with ML algorithms due to their diversity [10]. As a consequence, they have to be transformed to an input whose neural network is familiar with. For this reason, a technique called graph embedding [7] is utilized. Graph embedding is a transformation of graphs, which preserves as much graph property information as possible. The main motivation of proposed algorithm is to suggest a concept of establishing a machine learning model for layout calculation of nodes on the server and transferring it to the client for execution while preserving original graph appearance.

Figure 1 depicts the proposed methodology. The powerful server is responsible for graph feature learning, while the client manages graph visualization. Below, each segment is described in detail.

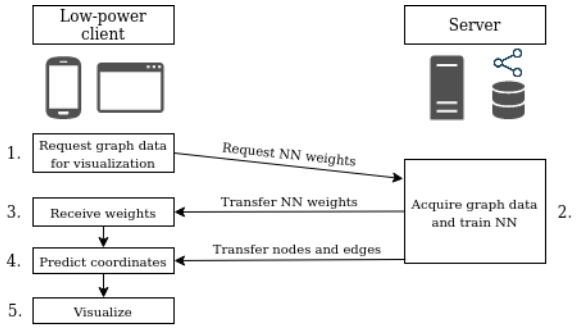

Figure 1: Proposed methodology workflow scheme.

2.1 Building machine learning model

ML model construction happens on the server-side of web application. The server is responsible to predict node coordinates and to teach the neural network. Figure 2 indicates suggested configuration of neural network based on the multilayer perceptron architecture (MLP) [14] with 2 dense hidden layers, each including q neurons. The output layer consists of 2 neurons, each representing coordinate in the final graph visualization. Nodes in the hidden layer are provided with the hyperbolic tangent activation function, whereas the output layer possesses simple linear activation function.

As an input data, MLP accepts n-dimensional array of node embedding, where n denotes the size of an embedding. The main aim of a node embedding is to transform a node into a vector space while preserving its properties [16]. Consequently, it retains node relatedness, resulting in clustering coupled nodes, while distancing unrelated nodes. Let c_i denote the i-th node in the graph. To construct its embedding, random walk-based node embedding *node2vec* [8] has been chosen. Random walk is a procedure of joining similar nodes in the complex graph to an embedding. As an input it accepts the starting node and produces a vector of specified length. Each element in the vector represents a characteristic of the visited node. With the help of extra parameters, which guide the walk, node2vec switches between breadth-first search (BFS) and depth-first search (DFS) to simulate the random walk. Correspondingly, it traverses local clusters as well as distant ones.

Before training the MLP, expected coordinates are computed for every node using an implementation of Fruchterman-Reingold force-directed algorithm [5] from NetworkX Python library [9]. As a learning algorithm backpropagation [11] in combination with Adam optimizer [12] is consumed. After having optimized the weights, the server sends them back to the client accompanied by the graph nodes and the edges. The client is subsequently responsible for graph layout calculation based on the received machine learning model and graph data.

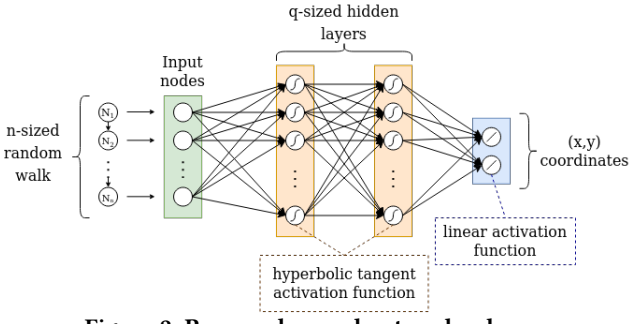

Figure 2: Proposed neural network scheme.

2.2 Graph visualization

The client is provided with the neural network from the server. As a consequence, it generates almost identical graph layout after acquiring the MLP weights. For this reason, the only task left for the client is to generate the random walks for an individual received node and predict its coordinates. After this stage, the client is prepared to visualize the graph.

3 Results and discussion

The aim of the experiments was to demonstrate improvements in speed compared to the standard visualization algorithms (e.g. Fruchterman-Reingold force-directed algorithm [5]) while maintaining the accuracy of graph visualization. Furthermore, the impact of neural network parameters was inspected by altering the number of neurons q in the hidden layers and by modifying the number of random walks r per individual node. Having small number of random walks might result in many different graph layouts as the random walks are arbitrarily generated.

Experiments were examined on hardware with following specifications: CPU: Intel Core i7-8700, RAM: 16 GB, GPU: GeForce GTX 1070. In addition, Python programming language was selected with TensorFlow [15] library of version 2.4.0 for neural network construction and NetworkX [9] library of version 2.5 to execute force-directed algorithm.

Input data was acquired from an advanced investigation platform [13]. Graphs are directed, containing from 10 to 1000 nodes. Figure 3 depicts an example of a graph with 12 related nodes. Before learning the model, the training set was prepared using output coordinates of Fruchterman-Reingold force-directed algorithm. As a consequence, ground truth for results is the latter algorithm, which was executed using NetworkX library. The root mean squared error (RMSE) metric were applied to analyze the correctness of the predicted node coordinates. RMSE indicates how distant are node coordinates between the suggested algorithm and node coordinates from the training set. Individual node coordinate was presented as floating-point numbers ranging between -10 and 10 units. Training period was limited to 100 epochs and learning rate was set to 0.001.

Three experiments were performed. The aim of the first was to assess an accuracy of proposed algorithm against the number of neurons q in the hidden layers of the MLP neural network. The number of random walks r for every node was arranged to 50, each having the length $n = 10$. Hence, the size of the input layer was fixed to 10. Such configuration has appeared as a good compromise between efficiency and time consumption when running experiments. Beginning with 10 neurons in each hidden layer, number of neurons was increased by 10 until 200. MLP learned the weights and RMSE was calculated in every iteration. Procedure was repeated 10 times for graphs with 10, 50, 100 and 1000 nodes. Figure 4 indicates average RMSE of all recurrences. Improvement of accuracy might be observed as the number of neurons increased, regardless of the initial graph size. However, starting with 100 neurons per hidden layer neural network exhibited no significant enhancement.

The focus of the second experiment was to evaluate precision of proposed algorithm against the number of random walks r per node. Strategy was related to the initial experiment. The number of random walks was manipulated with presupposition of having $q = 100$ neurons in each hidden layer of MLP and the length of each random walk n to be 50. Firstly, MLP was trained with 5 random walks per node and RMSE was calculated. Afterwards, 5 random walks were added and procedure was repeated. Iteration stopped when the number of random walks reached 100. Presented process was rerun 10 times for graphs having 10, 50, 100 and 1000 nodes. Figure 5 manifests average RMSE. It is noticeable how accuracy increased as more random walks were included for each node.

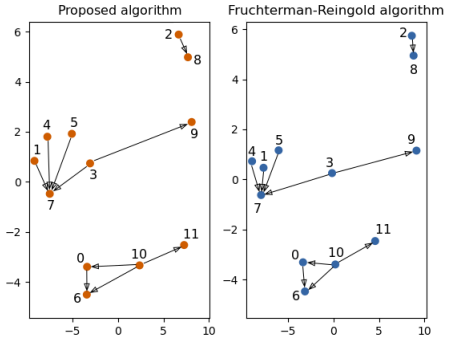

Figure 3: Example layout construction of proposed method against Fruchterman-Reingold algorithm.

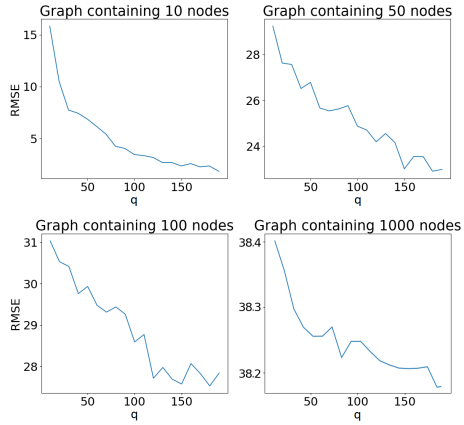

Figure 4: Graphs of accuracy against number of neurons in hidden layer of MLP.

Correspondingly to Figure 4, exactness started converging to some value as the number of random walks exceeded 60.

Finally, runtimes were examined. We experimented time consumption for node coordinate prediction with given neural network on the client for graphs of distinct size and discovered nearly linear increment with the number of nodes. Compared to an implementation of Fruchterman-Reingold algorithm in NetworkX library, Figure 6 indicates remarkable time-consumption reduction. In combination with the parallel layout construction while receiving the graph data, a considerable potential for minimizing time necessary to compute coordinates is perceived. Standard force-directed algorithms namely construct node coordinates with a time complexity

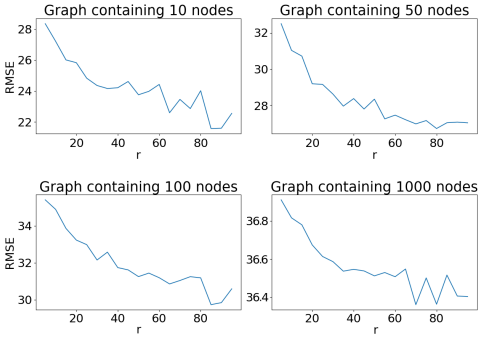

Figure 5: Graphs of accuracy against number of walks per node.

of $O(n^2)$ per individual iteration of the simulation, which might be time-consuming, especially for graphs extending 400 nodes as shown in Figure 6. Considering that the execution of algorithms occurs on devices with lower hardware specifications, the differences in time consumption should escalate.

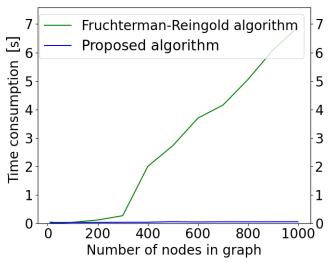

Figure 6: Time consumption for node coordinate calculation.

As mentioned, the purpose of the presented methodology was also to decrease bandwidth requirements. Let us presume our proposition of MLP architecture with $q = 100$ neurons in each hidden layer and an input layer of size $n = 10$. Hence, neural network consists of 11200 weights. On condition that each weight is presented as 32-bit floating point, the amount of data required to be transferred between the server and the client is 44800 bytes, which is considerably compact regarding today's technology. After the weights are sent, the client is able to gradually construct the layout while receiving the nodes and the edges.

A perspective to expose is the resistance of the proposed algorithm to the number of edges in the graph. Being the graph fully connected or not, the algorithm requires to generate r random walks of length n for every node. Random walk construction of a node without connections would result in a vector containing the node itself n-times.

4 Conclusion

A method for graph visualization in modern web applications using MLP neural network has been presented. Based on the experiments, time consumption of proposed procedure is promising compared to the Fruchterman-Reingold algorithm. In addition, visualization accuracy has been preserved. For this reason, the experiments have manifested that the purpose of proposed algorithm is fulfilled.

One of the concerns might be managing multiple big data requests from clients. However, as modern server infrastructures are getting more powerful each day, we believe our concept has practical value. Instead of executing complex calculations on the client and transferring huge amounts of data between the server and the client, we conceive our method as a replacement to aforementioned techniques. In addition, proposed methodology is able to construct layout incrementally at the time of receiving the nodes from the server, resulting in improved user experience. The only limitation is the requirement for transferring edges in an appropriate order for a random walk on the client. The problem can be alleviated by limiting the length of the walk. Moreover, the server could take advantage of caching to avoid neural network learning process duplication.

In this paper, the main focus was on node coordinate prediction. Nonetheless, our contribution might be reproduced to predict other graph parameters (e.g. graph coloring). In addition, presented methodology could be improved by selecting other neural network architectures. Having inspected the experiments, we have perceived how RMSE increased with graph size. Nevertheless, results have improved as the complexity of MLP increased. For this reason, we propose future research to focus on selecting a more complex neural network for given task. Aditionally, we suggest:

- Testing an impact of node2vec random walk parameters on accuracy,
- Sensitivity of the results to the length of node embeddings,
- Testing other graph embedding techniques,
- An evaluation of server performance when receiving multiple requests from clients and an evaluation of distributed computation.

Acknowledgments

The authors acknowledge joint financial support from the Slovenian Research Agency and Slovenian Ministry of the Interior (target research programme No. V2-2117).

References

[1] Maciej Besta, Dimitri Stanojevic, Tijana Zivic, Jagpreet Singh, Maurice Hoerold, and Torsten Hoefler. 2018. Log (graph) a near-optimal high-performance graph representation. In *Proceedings of the 27th international conference on parallel architectures and compilation techniques*. 1–13.

[2] Michael Bostock, Vadim Ogievetsky, and Jeffrey Heer. 2011. D^3 data-driven documents. *IEEE transactions on visualization and computer graphics* 17, 12 (2011), 2301–2309.

[3] Tim Dwyer. 2009. Scalable, versatile and simple constrained graph layout. In *Computer graphics forum*, Vol. 28. Wiley Online Library, 991–998.

[4] Nadime Francis, Alastair Green, Paolo Guagliardo, Leonid Libkin, Tobias Lindaaker, Victor Marsault, Stefan Plantikow, Mats Rydberg, Petra Selmer, and Andrés Taylor. 2018. Cypher: An evolving query language for property graphs. In *Proceedings of the 2018 International Conference on Management of Data*. 1433–1445.

[5] Thomas M. J. Fruchterman and Edward M. Reingold. 1991. Graph drawing by force-directed placement. *Software: Practice and experience* 21, 11 (1991), 1129–1164.

[6] Robert Gove. 2019. A Random Sampling O (n) Force-calculation Algorithm for Graph Layouts. In *Computer Graphics Forum*, Vol. 38. Wiley Online Library, 739–751.

[7] Palash Goyal and Emilio Ferrara. 2018. Graph embedding techniques, applications, and performance: A survey. *Knowledge-Based Systems* 151 (2018), 78–94.

[8] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. 855–864.

[9] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. 2008. Exploring Network Structure, Dynamics, and Function using NetworkX. In *Proceedings of the 7th Python in Science Conference*, Gaël Varoquaux, Travis Vaught, and Jarrod Millman (Eds.). Pasadena, CA USA, 11 – 15.

[10] William L. Hamilton, Rex Ying, and Jure Leskovec. 2017. Representation Learning on Graphs: Methods and Applications. *IEEE Data Eng. Bull.* 40, 3 (2017), 52–74. http://sites.computer.org/debull/A17sept/p52.pdf

[11] Henry J. Kelley. 1960. Gradient theory of optimal flight paths. *Ars Journal* 30, 10 (1960), 947–954.

[12] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization.. In *ICLR (Poster)*, Yoshua Bengio and Yann LeCun (Eds.). http://dblp.uni-trier.de/db/conf/iclr/iclr2015.html#KingmaB14

[13] Niko Lukač, Borut Žalik, Matej Brumen, David Jesenko, Štefan Kohek, Andrej Nerat, and Marko Bizjak. 2019. *Zasnova napredne preiskovalne platforme (NNP): zaključno poročilo.* UM FERI.

[14] Leonardo Noriega. 2005. Multilayer perceptron tutorial. *School of Computing. Staffordshire University* (2005).

[15] TensorFlow Developers. 2022. TensorFlow. https://doi.org/10.5281/ZENODO.4724125 Accessed: 2022-06-19.

[16] Mengjia Xu. 2021. Understanding graph embedding methods and their applications. *SIAM Rev.* 63, 4 (2021), 825–853.

[17] Abenov Zhansultan, Aubakirov Sanzhar, and Paulo Trigo. 2021. Parallel implementation of force algorithms for graph visualization. *Journal of Theoretical and Applied Information Technology* 99, 2 (2021), 503–515.