ORIGINAL ARTICLE

INTERNATIONAL JOURNAL OF ARTIFICIAL INTELLIGENCE

ISSN: 2692-5206, Impact Factor: 12,23

American Academic publishers, volume 05, issue 09,2025

Journal: https://www.academicpublishers.org/journals/index.php/ijai

VISUAL ANALYSIS AND TEACHING OF GRAPH STRUCTURES IN PYTHON

Utkirjon Mallayevich Saidov

Head, Department of Informatics, Samarkand State Pedagogical Institute; PhD.

Sevinch Normurod kizi Tursunbayeva

2nd-year student, Samarkand State Pedagogical Institute

E-mail: utkir.saidov.80@mail.ru

Abstract: The article examines effective approaches to learning and teaching graph theory through Python-based tools, with particular attention to NetworkX as a modern library for modeling, analyzing, and visualizing graphs. The paper motivates the pedagogical value of interactive visualizations for grasping abstract notions such as paths, cycles, degree, connectivity, and shortest routes. A sequence of classroom-ready examples is proposed to develop students' algorithmic thinking and problem-solving skills. The capabilities of NetworkX are discussed in comparison with commonly used software such as Gephi, Graphviz, and the Desmos Graphing Calculator.

Keywords: graph, graph theory, NetworkX, Python library, Gephi, Graphviz, Desmos Graphing Calculator.

Introduction. Graph theory plays a central role across computer science, mathematics, and engineering. However, its abstract concepts are often difficult for students to internalize without rich visualization and hands-on experimentation. Python—owing to its clarity, ecosystem, and low entry barrier—offers an ideal medium for inquiry-based learning. Among its libraries, NetworkX stands out for constructing both simple and complex graphs, running classical algorithms, and producing publication-quality visualizations.

In a pedagogical context, combining code and visuals helps students connect definitions to behavior. For instance, shortest-path problems (Dijkstra), minimum spanning trees, flows, centrality measures, and community detection can be demonstrated interactively. Furthermore, Python's integration with web frameworks (e.g., Streamlit, Flask) enables lightweight dashboards for classroom demonstrations and student projects.

Illustrative example (Dijkstra).

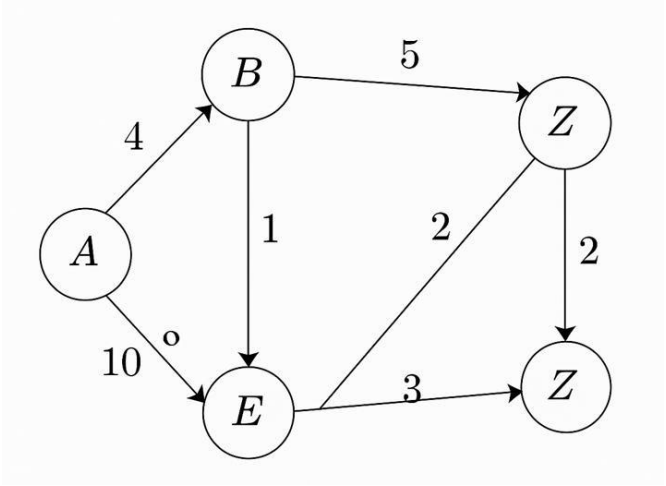
The following minimal snippet defines a weighted directed graph and computes the shortest path between two vertices using Dijkstra's algorithm in NetworkX:

```
import networkx as nx

G = nx.DiGraph()

G.add_weighted_edges_from([

   ('A', 'B', 4),
```



ORIGINAL ARTICLE

INTERNATIONAL JOURNAL OF ARTIFICIAL INTELLIGENCE

ISSN: 2692-5206, Impact Factor: 12,23

American Academic publishers, volume 05, issue 09,2025

Journal: https://www.academicpublishers.org/journals/index.php/ijai

```
    ('A', 'C', 2),
    ('B', 'C', 1),
    ('B', 'D', 5),
    ('C', 'D', 8),
    ('C', 'E', 10),
    ('D', 'E', 2),
    ('D', 'Z', 6),
    ('E', 'Z', 3)
])
```

The compactness and readability of the code facilitate formative assessment in class: students can hypothesize expected routes, run the algorithm, and interpret the output against the structure of the graph. The same approach extends naturally to other topics such as minimum spanning trees and flow algorithms.

Comparison with alternative tools.

While Gephi and Graphviz provide powerful visualization capabilities, they offer limited algorithmic flexibility for step-by-step demonstrations. By contrast, NetworkX enables both algorithmic experimentation and clear visuals within a single, reproducible notebook or script. Desmos is helpful for quick illustrations but is not designed for general graph algorithms. For large-scale or highly interactive visual analytics, dedicated tools may be preferable; however, for teaching and research prototypes NetworkX is well-suited.

Pedagogical workflow.

A practical lesson can proceed as follows: (1) introduce core concepts with small, interpretable graphs; (2) translate definitions into code with NetworkX; (3) visualize and annotate steps of an algorithm; (4) let students modify parameters and predict outcomes; (5) conclude with a brief reflection connecting the experiment to real-world applications (routing, scheduling, social networks). This cycle supports conceptual understanding and develops computational thinking.

Conclusion.

Modeling and visualizing graphs with Python—especially via NetworkX—provides an effective didactic pathway for teaching graph theory. Integrating code, visualization, and narrative explanations in a single environment enhances engagement, deepens understanding, and bridges theory with authentic problem solving.

REFERENCES

1. Bondy, J. A., & Murty, U. S. R. (2008). Graph Theory.
2. Diestel, R. (2016). Graph Theory (5th ed.).

ISSN: 2692-5206, Impact Factor: 12,23
American Academic publishers, volume 05, issue 09,2025
Journal: https://www.academicpublishers.org/journals/index.php/ijai

3. West, D. B. (2001). Introduction to Graph Theory (2nd ed.).

4. Easley, D., & Kleinberg, J. (2010). Networks, Crowds, and Markets.

5. Barabási, A.-L. (2016). Network Science.

6. Friedman, J., Hastie, T., & Tibshirani, R. (2009). The Elements of Statistical Learning (2nd ed.).

7. Gross, J. L., & Yellen, J. (2005). Graph Theory and Its Applications (2nd ed.).

8. Newman, M. E. J. (2010). Networks: An Introduction.