The Journal of Open Source Software

Edge Addition Planarity Suite and Generalized Graph Library

John M. Boyer [1¶] **and Wanda B. K. Boyer**[1]

1 Independent Researcher, Canada ¶ Corresponding author

DOI: 10.21105/joss.11163

Software

- Review
- Repository
- Archive

Editor: Daniel S. Katz

Reviewers:

- @DeIn0r
- @MODSetter

Submitted: 05 August 2026
Published: 13 September 2026

Summary

A *graph* is a mathematical structure comprising a set V of vertices and a set E of edges, with each edge e corresponding to a pair of vertices called the *endpoints* of e. A graph is *planar* if its vertices can be placed in distinct locations on a plane surface and its edges can be drawn on the plane without the edges intersecting, except at their common vertex endpoints. For a planar graph, a planarity algorithm typically outputs a combinatorial data structure that validation code can use to certify the planarity of the input graph. A planar graph drawing is typically produced by a separate algorithm. On the other hand, if an input graph is not planar, then a planarity algorithm typically outputs a minimal subgraph of the input graph that obstructs planarity. Validation code can use a minimal planarity-obstructing subgraph to certify the non-planarity of an input graph. Moreover, a minimal subgraph obstructing planarity can be used to help decide how to amend an input graph to *planarize* it.

Graphs are used to model a very wide array of real-world problems in which there are objects and relationships between the objects. In artificial intelligence, graphs are used to help with reasoning tasks, such as about the relationship pathways between persons of interest in law enforcement or between genes, tissues, diseases, and medications in bioinformatics research. In physics, graphs and planarity are used to help compute material phase transitions, particle interactions, and electromagnetic duality. Similarly, in chemistry, graphs may be used to represent atoms and their valence bonds in molecules. Planarity testing can help determine feasible molecular arrangements because the molecular graphs for many compounds of interest must be planar. In electronics, a graph may be used to represent the electrical components and wiring in a circuit, such as transistors and etchings on a silicon wafer. The graph of a circuit must be planar or planarized to eliminate short circuits.

The open source software described in this paper provides a highly performant generalized graph library that also implements state-of-the-art algorithms for planarity-related problems. The generalized graph library and planarity-related algorithms are available to C/C++ and Python graph application developers.

Statement of need

The `Edge Addition Planarity Suite` is an open source software package that was originally developed to implement a new planarity algorithm in Boyer & Myrvold (2004). The Edge Addition Planarity Algorithm is currently regarded as one of two state-of-the-art planarity algorithms due to its conceptual simplicity as well as its speed of execution (Contributors to Wikipedia, 2026). The software package is available from the `edge-addition-planarity-suite` GitHub repository under a 3-clause BSD license.

To enable development of the Edge Addition Planarity Algorithm, as well as the other planarity-related algorithms in Boyer (2006) and Boyer (2012), it was necessary to develop an extensible,

generalized graph library. This generalized graph library provides a high-performance solution for the graph algorithm representation and processing needs of a very wide array of graph applications.

In addition to enabling graph application development in C/C++, the popularity of Python as a scientific application development language has prompted the need for making this graph library and the algorithms it implements available to Python developers. The source code may be accessed by the `planarity` Github repository, and the `planarity` Python package may be installed via the Python Package Index (`PyPI`) by simply running `pip install planarity`.

State of the field

Several large scientific software packages offer a graph planarity algorithm, which is a complex graph algorithm suitable for benchmarking graph libraries. Mathematica and Wolfram Alpha include planarity algorithms, but these packages are not meant for high-performance graph algorithms, so the Library for Efficient Data Types and Algorithms (LEDA) is significantly faster than they are because it is an industrial library designed for computational efficiency. Yet, in Boyer et al. (2004), all planarity algorithms implemented in LEDA were shown to be several times slower than the Edge Addition Planarity Algorithm implementation. LEDA also has the disadvantage of being C++ only and has a proprietary license. The only graph library shown (in Boyer et al. (2004)) to have similar performance to the Edge Addition Planarity Suite is the Public Implementation of a Graph Algorithm Library and Editor (PIGALE). However, PIGALE is also C++ only, has a GPLv2 license, and lacks advanced algorithms for homeomorphic subgraph search. For example, torus embedding algorithms and torus obstruction isolation algorithms are able to operate differently based on whether a search for a subgraph homeomorphic to $K_{3,3}$ finds a result (Myrvold & Kocay, 2011; Myrvold & Woodcock, 2018). Similarly, several graph theoretic results are applicable to graphs that are known to be $K_{3,3}$-free, including results related to graph isomorphism, reachability, and finding matching cuts (Datta et al., 2009; Feghali et al., 2025; Thierauf & Wagner, 2014).

Software design

The generalized graph library in the Edge Addition Planarity Suite has a carefully curated API and an object-oriented architecture. The base **Graph** class structure includes

- base graph, vertex, and edge structure definitions and getter/setter methods;
- graph, vertex and edge allocation, deallocation, and reset methods;
- methods for iterating through all vertices, edges, and edge adjacency lists;
- graph readers and writers for several formats; and
- a dynamic subclassing/extension mechanisms and virtual function table.

The base Graph is extendable with utility methods for exploring a graph, labelling vertices and edges according to a depth-first search (DFS), and for managing connectivity and biconnectivity information. The vertex and edge structures are extended to enable storage of the information generated by the utility methods.

A Graph structure extended with the DFS-related utilities is further extendable to a **Planarity Graph**, which also further extends the vertex and edge structures. The main additional method provided by a Planarity Graph is `gp_Embed()`, which takes a Planarity Graph as input and outputs either a combinatorial planar embedding or a minimal planarity-obstructing subgraph.

Given a Planarity Graph, a number of subclass extensions are available for planarity-related problems. One extension solves outerplanarity. Another implements the planar graph drawing algorithm in Boyer (2006). Three other extensions provide Graph subclasses that solve the homeomorphic subgraph search algorithms appearing in Boyer (2012). All of these extensions further extend the vertex and edge structures and overload the `gp_Embed()` function.

In addition to being available to C and C++ developers, the APIs of the Edge Addition Planarity Suite and Generalized Graph Library are also made available via the `planarity` Python package with the source code made available in the `planarity` Github repository. Cython is used to preserve high performance while also broadening availability to the Python developer community.

A final aspect of this open source software project is our emphasis on software reliability. We perform validation code on `gp_Embed()` and its five overloads on billions of randomly generated graphs up to 10 million vertices and 30 million edges. The GitHub actions that run on every pull request include sanitizer tests on all graphs on 8 vertices. Finally, the `edge-addition-planarity-suite-testing` GitHub repository provides our multithreaded Python code for validating `gp_Embed()` and its five overloads on the more than one billion graphs having 11 or fewer vertices, as generated by the `geng` tool in Nauty and Traces (McKay & Piperno, 2025).

Research impact statement

The journal publication of the Edge Addition Planarity Algorithm (Boyer & Myrvold, 2004) currently has over 360 scholarly citations according to Google Scholar. The algorithm has been implemented in several large open source projects, including Boost, Magma, nauty and Traces (McKay & Piperno, 2025), and the Open Graph Drawing Framework (web archive). An executable program built from the source code was used to build a knot theory software package (Jablan & Sazdanović, 2007, pp. 7, 38). The source code from the Edge Addition Planarity Suite repository has been directly included in several large open source projects, including SageMath, Digraphs, and over 80 Linux platforms releases including for Debian, Devuan, Kali, openSUSE, Raspbian, Ubuntu, Alpine, ALT, Fedora, FreeBSD, Gentoo, Manjaro, MSYS2, and Parabola, according to repology/edge-addition-planarity-suite (web archive) and repology/planarity (web archive).

The Python package named `planarity` was originally developed in 2013 by Aric Hagberg as a way to get the planarity testing and drawing algorithms from Boyer & Myrvold (2004) and Boyer (2006) to work with graphs from NetworkX (Hagberg et al., 2008). Recently, Hagberg transferred the `planarity` Github repository to the GitHub Graph Algorithms Organization and the `planarity` Python package to the PyPI Graph Algorithms Organization, which are both maintained by the authors. On August 1, 2026, the Top PyPI Packages website indicated that the `planarity` Python package had over 700,000 downloads for the preceding month and was ranked in the top 1% of PyPI package downloads (ranked 5262 out of 864,036 projects). According to pepy.tech (web archive) the `planarity` Python package has been downloaded over 2 million times in total. The `planarity` Python package has now been included as a standard Python package in several versions of Linux including Debian, Devuan, Kali, Raspbian, and Ubuntu, according to repology/python:planarity (web archive).

AI usage disclosure

No generative AI tools were used in the writing of this manuscript, nor in the preparation of supporting materials. The authors are not aware of any substantive use of generative AI tooling yet in the development of the open source software projects reported in this paper. However, in light of the proliferation of availability and use of generative AI tooling in software development, we assert that it is our pull request (PR) review process, performed with no generative AI tooling, that protects the projects and their consumers from low quality or harmful contributions. As the project owners, we amend PRs with further commits or request contributor changes as needed, and we use our own holistic project expertise to ensure that the code contribution meets our quality requirements. Specifically, a code contribution must cover all parts of the project that it should, its code must be consistent with the rest of the code in the module(s) containing it, it must be a good solution conforming to every aspect

of the technical specifications in the issue(s) it solves, the tests also given in the technical specifications must be correctly coded, the coded solution must pass the coded tests, and all code in the contribution must pass our sanitizer and compiler-warning-sensitive automated tests. Only once a project owner has determined that these requirements are met by the pull request's code contribution is the pull request approved to be merged.

References

Boyer, J. M. (2006). A new method for efficiently generating planar graph visibility representations. In P. Eades & P. Healy (Eds.), *Proceedings of the* 13^{th} *international symposium on graph drawing 2005* (Vol. 3843, pp. 508–511). Springer-Verlag. https://doi.org/10.1007/11618058_47

Boyer, J. M. (2012). Subgraph homeomorphism via the edge addition planarity algorithm. *Journal of Graph Algorithms and Applications*, *16*(2), 381–410. https://doi.org/10.7155/jgaa.00268

Boyer, J. M., Cortese, P. F., Patrignani, M., & Di Battista, G. (2004). Stop minding your P's and Q's: Implementing a fast and simple DFS-based planarity testing and embedding algorithm. In G. Liotta (Ed.), *Proceedings of the* 11^{th} *international symposium on graph drawing 2003* (Vol. 2912, pp. 25–36). Springer-Verlag. https://doi.org/10.1007/978-3-540-24595-7_3

Boyer, J. M., & Myrvold, W. J. (2004). On the cutting edge: Simplified $O(n)$ planarity by edge addition. *Journal of Graph Algorithms and Applications*, *8*(3), 241–273. https://doi.org/10.7155/jgaa.00091

Contributors to Wikipedia. (2026). Planarity testing. In *Wikipedia, the free encyclopedia*. Wikipedia. https://en.wikipedia.org/w/index.php?title=Planarity_testing&oldid=1336627782

Datta, S., Nimbhorkar, P., Thierauf, T., & Wagner, F. (2009). Graph isomorphism for $K_{3,3}$-free and K_5-free graphs is in log-space. *Proceedings of the 29th Annual Conference on Foundation of Software Technology and Theoretical Computer Science (FSTTCS)*, 145–156. https://doi.org/10.4230/LIPIcs.FSTTCS.2009.2314

Feghali, C., Lucke, F., Paulusma, D., & Ries, B. (2025). Matching cuts in graphs of high girth and H-free graphs. *Algorithmica*, *87*, 1199–1221. https://doi.org/10.1007/s00453-025-01318-8

Hagberg, A. A., Schult, D. A., & Swart, P. J. (2008). Exploring network structure, dynamics, and function using NetworkX. *Python in Science Conference*. https://doi.org/10.25080/TCWV9851

Jablan, S., & Sazdanović, R. (2007). *LinKnot: Knot theory by computer* (Vol. 21). World Scientific. https://doi.org/10.1142/6623

McKay, B. D., & Piperno, A. (2025). *Nauty and traces user's guide (version 2.9.0)*. https://pallini.di.uniroma1.it/nug29.pdf

Myrvold, W., & Kocay, W. (2011). Errors in graph embedding algorithms. *Journal of Computer and System Sciences*, *77*(2), 430–438. https://doi.org/10.1016/j.jcss.2010.06.002

Myrvold, W., & Woodcock, J. (2018). A large set of torus obstructions and how they were discovered. *The Electronic Journal of Combinatorics*, *25*(1), 1–17. https://doi.org/10.37236/3797

Thierauf, T., & Wagner, F. (2014). Reachability in $K_{3,3}$-free and K_5-free graphs is in unambiguous logspace. *Chicago Journal of Theoretical Computer Science*, *2014*, 1–29. https://doi.org/10.4086/cjtcs.2014.002