GCol: A High-Performance Python Library for Graph Colouring

Rhyd Lewis [1¶] **and Geraint Palmer** [1]

1 School of Mathematics, Cardiff University, Wales, United Kingdom ¶ Corresponding author

DOI: 10.21105/joss.07871

Software

- Review
- Repository
- Archive

Editor: Vissarion Fisikopoulos

Reviewers:

- @paulbrodersen
- @IvanIsCoding

Submitted: 19 February 2025
Published: 02 April 2025

Summary

Graph colouring is the computational problem of assigning colours to entities of a graph so that adjacent entities receive different colours. The aim is to use as few colours as possible. In general terms, a graph is a mathematical object comprising a set of *nodes* and a set of *edges* that join pairs of nodes. Graphs are also known as *networks*, nodes as *vertices*, and edges as *links*. Examples of graph colouring are shown in Figure 1, which help to demonstrate the following principles.

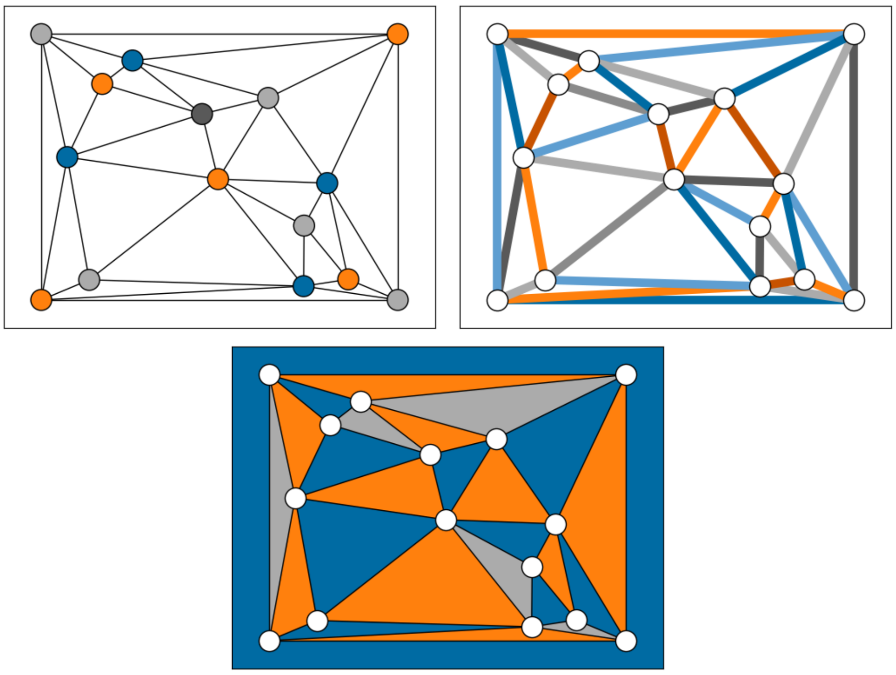

Figure 1: A node colouring, edge colouring, and face colouring (respectively), of a fifteen-node planar graph. Each of these examples uses the minimum number of colours and was generated using GCol's colouring and visualisation routines.

- A *node colouring* is an assignment of colours to the nodes of a graph so that adjacent nodes have different colours. The smallest number of colours needed for the nodes of a graph G is known as its chromatic number, denoted by $\chi(G)$. Identification of $\chi(G)$ is an NP-hard problem.
- An *edge colouring* is an assignment of colours to the edges of a graph so that all adjacent

edges have different colours. The smallest number of colours needed for the edges of a graph G is known at the chromatic index, denoted by $\chi'(G)$. According to Vizing's theorem, $\chi'(G)$ is either $\Delta(G)$ or $\Delta(G)+1$, where $\Delta(G)$ is the maximum degree in G. Identifying $\chi'(G)$ is also NP-hard (Toft & Wilson, 2011).

- A *face colouring* is an assignment of colours to the faces of a planar embedding so that all adjacent faces have different colours. Note that planar embeddings only exist for planar graphs. The smallest number of colours needed to colour the faces of a planar embedding is known as its face chromatic number. Due to the Four Colour Theorem, this number never exceeds four and, unlike $\chi(G)$ and $\chi'(G)$, can be determined in polynomial time (Robertson et al., 1997).

Graph colouring has applications in many practical areas including timetabling, sports league scheduling, designing seating plans, code optimisation, and solving Sudoku puzzles (Lewis, 2021a). It is also a topic of theoretical interest that often appears in university-level courses on graph theory, algorithms, and combinatorics (Cranston, 2024).

GCol is a new, open-source Python library for graph colouring that is built on top of the well-known NetworkX library (Hagberg et al., 2008). It provides easy-to-use, high-performance algorithms for the above three problems, as well as routines for equitable colouring, weighted colouring, pre-colouring, maximum independent set identification, and solution visualisation. The following code snippet shows how to use the library to create a dodecahedral graph G, colour its nodes using $\chi(G)=3$ colours, and then output the solution in textual and diagrammatic form (see Figure 2).

```
>>> import networkx as nx
>>> import matplotlib.pyplot as plt
>>> import gcol
>>> G = nx.dodecahedral_graph()
>>> c = gcol.node_coloring(G)
>>> print("Node coloring of G =", c)
Node coloring of G = {0: 0, 1: 1, 19: 1, 10: 1, 2: 0, 3: 2, 8: 0, 9: 2, 18: 0,
11: 2, 6: 1, 7: 2, 4: 0, 5: 2, 13: 0, 12: 1, 14: 1, 15: 0, 16: 2, 17: 1}
>>> nx.draw_networkx(G, node_color=gcol.get_node_colors(G, c))
>>> plt.show() # See Figure 2
```

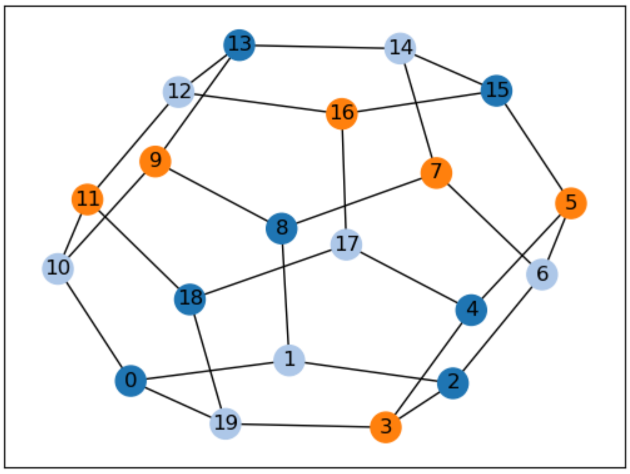

Figure 2: Output from the code snippet.

Statement of need

Open-source resources for graph colouring have existed for some time, primarily for node colouring. An early example is the PL/I code for node colouring included in the paper of Leighton (1979). Stand-alone C-based resources were also made available online in the mid-1990s due to Culberson (1994) and Trick (1994). A Java-based package implementing the methods of Culberson is now also available (Shah, 2020).

Graph colouring functionality is also included in some popular open-source C++ libraries. *The Boost Graph Library, Version 1.8.7* (2025) uses a simple constructive heuristic for node colouring, while the Lemon library (Dezso et al., 2011) includes a method for colouring the nodes of planar graphs using at most five colours. *The Goblin Graph Library, Version 2.8* (2025) features a similar method to Lemon and, in addition, includes a mixed integer linear programming approach for exactly solving the node and edge colouring problems. This algorithm has an exponential time complexity and, consequently, is unsuitable for larger problem instances.

A further open-source option that, in addition, includes visualisation tools is provided by SageMath (The Sage Developers, 2025). This has methods for both node and edge colouring, and can also enumerate *all* node colourings of a graph. Like Goblin, however, its algorithms are based on integer programming and operate in exponential time. NetworkX itself also includes some simple greedy heuristics for node colouring (Hagberg et al., 2008) as does the alternative Python library Graph-Tool (Peixoto, 2014). In addition, NetworkX features an exact polynomial-time algorithm for balancing the number of nodes per colour (equitable node colouring); however, this can only be applied when the number of available colours exceeds $\Delta(G)$—for fewer colours, where the problem is NP-hard, no functionality is available.

Further specialised methods for node colouring are also provided by the ColPack software (Gebremedhin et al., 2013) and in the algorithm suite of Lewis (2021b), both in C++. The algorithms of ColPack are described as "greedy heuristics in the sense that the algorithms progressively extend a partial colouring by processing one vertex at a time, in some order, in each step assigning a vertex the smallest allowable colour". On the other hand, the suite of Lewis (2021b) features several contrasting algorithms, including constructive heuristics, an exact algorithm based on backtracking, and bespoke metaheuristics. Finally, graph colouring functionality is also provided by the igraph library (Csardi & Nepusz, 2005). The open-source C- and R-based versions of this library use similar greedy heuristics to ColPack, whereas the (proprietary) Mathematica version also includes tools for edge and face colouring.

The above survey suggests that existing open-source options for graph colouring are limited. Current resources tend to either use simple constructive heuristics that lead to low-quality solutions, or exponential-time exact algorithms that cannot cope with larger graphs. There are also few open-source options for edge colouring, equitable colouring, and solution visualisation, and, to our knowledge, no options for face colouring, weighted colouring, or pre-colouring.

The GCol library features routines for all of the above. Optimisation is performed by a choice of node-colouring algorithms that include an exact, heuristically-guided, exponential-time backtracking algorithm, and several high-performance polynomial-time heuristics. The latter methods combine fast constructive methods with contemporary local search heuristics that extend the C++ implementations of Lewis (2021b), allowing high-quality solutions to be generated in reasonable run times, even for very large graphs. Edge colourings and face colourings are also determined by these algorithms by colouring, respectively, the nodes of the corresponding line graphs and dual graphs.

The various optimisation algorithms available in the GCol library are described in detail in its official documentation (*GCol*, 2025) and in the book of Lewis (2021a). These resources include detailed information on the asymptotic complexity of all methods used. Results concerning the runtimes and accuracy of GCol's algorithms, equitable colouring functionality, and other associated optimisation problems are also reported in the documentation.

References

Cranston, D. (2024). *Graph coloring methods*. Cranston, Richmond, Virginia. ISBN: 979-8-218-46240-0

Csardi, G., & Nepusz, T. (2005). The IGraph software package for complex network research. *InterJournal*, *Complex Systems*(1695), 1–9. https://igraph.org/

Culberson, J. (1994). *Culberson's node coloring implementations*. https://www3.cs.stonybrook.edu/~algorith/implement/culberson/implement.shtml.

Dezso, B., Juttner, A., & Kovacs, P. (2011). LEMON – an open source C++ graph template library. *Electronic Notes in Theoretical Computer Science*, *264*(5), 23–45. https://doi.org/10.1016/j.entcs.2011.06.003

GCol: A library for graph coloring. (2025). https://gcol.readthedocs.io/en/latest/.

Gebremedhin, A., Nguyen, D., Patwary, M., & Pothen, A. (2013). ColPack: Graph coloring software for derivative computation and beyond. *ACM Transactions on Mathematical Software*, *40*(1), 1–31. https://doi.org/10.1145/2513109.2513110

Hagberg, A., Schult, D., & Swart, P. (2008). Exploring network structure, dynamics, and function using NetworkX. In G. Varoquaux, T. Vaught, & J. Millman (Eds.), *Proceedings of the 7th Python in science conference (SciPy2008)* (pp. 11–15). https://doi.org/10.25080/TCWV9851

Leighton, F. (1979). A graph coloring algorithm for large scheduling problems. *Journal of Research of the National Bureau of Standards*, *84*(6), 489–506. https://doi.org/10.6028/jres.084.024

Lewis, R. (2021a). *A guide to graph colouring: Algorithms and applications*. Springer Cham. https://doi.org/10.1007/978-3-030-81054-2

Lewis, R. (2021b). *A guide to graph colouring: User guide*. https://rhydlewis.eu/gcol/.

Peixoto, T. (2014). The graph-tool Python library. *Figshare*. https://doi.org/10.6084/m9.figshare.1164194

Robertson, N., Sanders, D., Seymour, P., & Thomas, R. (1997). The four color theorem. *Journal of Combinatorial Theory*, *70*, 2–44.

Shah, S. (2020). JCOL: A Java package for solving the graph coloring problem. *Journal of Open Source Software*, *5*(48), 1843. https://doi.org/10.21105/joss.01843

The Boost graph library, version 1.8.7. (2025). https://www.boost.org/doc/libs/1_46_1/libs/graph/doc/index.html.

The Goblin graph library, version 2.8. (2025). https://goblin2.sourceforge.net/.

The Sage Developers. (2025). *SageMath, the Sage Mathematics Software System (Version 10.5)*.

Toft, B., & Wilson, R. (2011). A brief history of edge-colorings – with personal reminiscences. *Discrete Mathematics Letters*, *6*, 38–46. https://doi.org/10.47443/dml.2021.s105

Trick, M. (1994). *COLOR.C: Easy code for graph coloring*. https://mat.tepper.cmu.edu/COLOR/solvers/trick.c.