GraphCalc: A Python Package for Computing Graph Invariants in Automated Conjecturing Systems

Randy Davila [1,2]

1 RelationalAI, United States **2** Department of Computational Applied Mathematics & Operations Research,Rice University, United States

Summary

`GraphCalc` is a Python library for computing an extensive collection of graph-theoretic invariants, designed to support research in combinatorics, network science, and automated reasoning. It implements more than 100 exact functions, covering classical measures (e.g., independence number, chromatic number, spectral radius) and many lesser-known invariants central to extremal graph theory and domination theory.

Originally developed as the invariant engine for the automated conjecturing system *TxGraffiti* (Davila, 2025a), `GraphCalc` has grown into a general-purpose research tool for constructing large, structured datasets of graph invariants. These datasets—often organized into tabular *knowledge tables*—enable symbolic pattern mining, hypothesis generation, and automated conjecture discovery. For example:

```
>>> import graphcalc as gc
>>> graphs = [gc.cube_graph(), gc.octahedron_graph()]
>>> functions = ["order", "size", "spectral_radius", "independence_number"]
>>> gc.compute_knowledge_table(functions, graphs)
   order  size  spectral_radius  independence_number
0      8    12              3.0                    4
1      6    12              4.0                    2
```

While general-purpose libraries like `NetworkX` (Hagberg et al., 2008), `igraph` (Csárdi & Nepusz, 2006), and `SageMath` (The Sage Developers, 2020) provide broad graph functionality, they rarely support the wide range of nonstandard invariants used in combinatorics. `GraphCalc` fills this gap by offering exact implementations of many parameters unavailable elsewhere. All functions are implemented exactly using integer programming, enumeration, or symbolic methods. For NP-hard invariants (e.g., independence number, chromatic number, domination variants), `GraphCalc` relies on mixed-integer programming models via `PuLP` (Mitchell et al., 2011) and solvers such as `COIN-OR CBC`, ensuring exactness for small- to medium-sized graphs where symbolic relationships are most visible.

By enabling high-resolution invariant datasets, `GraphCalc` complements automated conjecturing systems like *TxGraffiti* and the *Optimist* (Davila, 2025b). These systems analyze numerical patterns in `GraphCalc`'s output to generate new conjectures, many of which have already been proven as theorems. `GraphCalc` serves as both a comprehensive toolkit for graph theorists and a foundational component for symbolic discovery in modern mathematics.

Features

`GraphCalc` offers a robust suite of tools for computing, analyzing, and visualizing graph-theoretic invariants. It combines an intuitive Python interface with solver-enhanced backends and supports both `NetworkX` graph objects and internal `SimpleGraph` and related types—making

it versatile for everyday use, educational settings, and advanced mathematical experimentation. Key features include:

- **Extensive invariant coverage:** Compute a broad range of exact graph invariants, including classical quantities such as chromatic, clique, vertex cover, and independence numbers, as well as structural and degree-based invariants like residue, Slater number, and annihilation number.

- **Domination and forcing variants:** Includes over a dozen domination-type parameters (e.g., total, Roman, double Roman, restrained, outer-connected) and propagation-based parameters such as zero forcing, positive semidefinite zero forcing, and k-power domination. All are computed exactly using integer programming or exhaustive search.

- **Spectral and structural analysis:** Supports spectral computations, including adjacency and Laplacian eigenvalues, spectral radius, and algebraic connectivity, along with Boolean predicates for structural properties such as planarity, claw-freeness, triangle-freeness, and subcubicity.

- **Graph and polytope generators:** Provides built-in generators for classical graphs and convex 3D polytopes (e.g., tetrahedra, cubes, fullerenes)—useful for visualization, testing, and conjecture exploration.

- **Batch evaluation and knowledge tables:** Enables the evaluation of multiple invariants across entire graph collections using `compute_knowledge_table` (see previous section) or `all_properties`.

- **Visualization and user experience:** Offers built-in rendering for graphs and polytopes, fully type-annotated functions, extensive test coverage, and online documentation designed to support both research and instructional use.

Example Usage

The `GraphCalc` package supports both single-graph queries and batch evaluation over collections of graphs and polytopes (see the previous section). Below is a basic example using the *Petersen graph*:

```python
>>> import graphcalc as gc
>>> # Create the Petersen graph
>>> G = gc.petersen_graph()
>>> # Compute selected invariants
>>> gc.independence_number(G)
4
>>> gc.residue(G)
3
>>> gc.claw_free(G)
False
```

Relevance to Automated Discovery

Automated mathematical discovery has a rich history, dating back to symbolic logic programs like Wang's *Program II* in the 1950s (Wang, 1960), and advancing significantly with systems such as Fajtlowicz's *Graffiti* (Fajtlowicz, 1988) and DeLaViña's *Graffiti.pc* (DeLaViña, 2005) in the 1980s and 1990s. These pioneering systems demonstrated that computers could do more than verify known mathematics—they could help *generate* it, particularly by formulating conjectures grounded in patterns among graph invariants. Notably, *Graffiti* included its own embedded module for computing such invariants, a design decision that enabled the system to generate over 60 published conjectures, many appearing in top mathematical journals.

`GraphCalc` continues this lineage. Originally developed as the internal invariant engine for the *TxGraffiti* system, it served for years as a private computational backend before being released as an open-source Python package. This decision was motivated by the growing interest in AI-assisted mathematical reasoning and the desire to make a high-quality, extensible invariant engine available for others to experiment with.

Today, `GraphCalc` powers the latest version of *TxGraffiti* and its agentic counterpart the *Optimist* (Brimkov et al., 2024; Davila, 2025b), which analyze large families of graphs and polytopes to discover symbolic conjectures. While not optimized for massive-scale network analysis, `GraphCalc` excels in the domain where most mathematical conjectures are formed: *small to medium-sized graphs* that are easily visualized and reasoned about. This design philosophy echoes the foundational principles of Fajtlowicz's original system, which emphasized working with "small but interesting" graphs as fertile ground for discovery. By transforming these structures into structured numerical profiles, `GraphCalc` enables automated systems to detect symbolic patterns and formulate conjectures that are both novel and mathematically meaningful.

Acknowledgements

References

Brimkov, B., Davila, R., Schuerger, H., & Young, M. (2024). On a conjecture of TxGraffiti: Relating zero forcing and vertex covers in graphs. *Discret. Appl. Math.*, *359*, 290–302. https://doi.org/10.1016/j.dam.2024.08.006

Csárdi, G., & Nepusz, T. (2006). The igraph software package for complex network research. *InterJournal Complex Systems*, *1695*. https://doi.org/10.5281/zenodo.3630268

Davila, R. (2025a). *Automated conjecturing in mathematics with TxGraffiti*. https://doi.org/10.48550/arXiv.2409.19379

Davila, R. (2025b). *The optimist: Towards fully automated graph theory research*. https://doi.org/10.48550/arXiv.2411.09158

DeLaViña, E. (2005). *Graffiti.pc*: A Variant of Graffiti. In *Graphs and discovery* (Vol. 69, pp. 71–88). American Mathematical Society. https://doi.org/10.1090/dimacs/069/05

Fajtlowicz, S. (1988). On conjectures of Graffiti. *Discrete Mathematics*, *72*(1), 113–118. https://doi.org/10.1016/0012-365X(88)90199-9

Hagberg, A. A., Schult, D. A., & Swart, P. J. (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

Mitchell, S., Dunning, I., & O'Sullivan, M. (2011). PuLP: A linear programming toolkit for python. *Optimization Online*. https://optimization-online.org/2011/09/3178/

The Sage Developers. (2020). *SageMath, the Sage Mathematics Software System (Version 9.1)*. https://doi.org/10.5281/zenodo.4066866

Wang, H. (1960). Toward mechanical mathematics. *IBM Journal of Research and Development*, *4*(1), 2–22. https://doi.org/10.1147/rd.41.0002