Supplementary Verification Files for "On extremal 2-connected graphs avoiding $(0 \mod 4)$-cycles"

Hojin Chu, Boram Park, and Homoon Ryu

July 14, 2025

1. Overview

This repository provides verification code for Proposition 2.3 of the paper "On extremal 2-connected graphs avoiding (0 mod 4)-cycles" by the same authors. The code systematically generates candidate graphs and filters them according to structural constraints specified in the proposition:

- Absence of 4-cycles
- Absence of certain $(0, n-1)$-paths of lengths $\equiv l_1, l_2 \pmod 4$
- Minimum degree constraints

The resulting graphs are then deduplicated up to isomorphism, and removes graphs containing 8-cycles. This tool is designed to support exploratory or extremal graph theory research.

Let $n \geq 3$, and $(H; x, y)$ be an n-vertex graph without a (0 mod 4)-cycle such that every edge in H is contained in an (x, y)-path of length $\equiv \ell_1 \pmod 4$ or $\equiv \ell_2 \pmod 4$. If there is no (x, y)-path of length $\equiv \ell \pmod 4$ for some $\ell \in L = \{\ell_1, \ell_2\}$, then H is bipartite, and hence $e(H) \leq \frac{3n-6}{2}$ by Lemma 2.5 (iii). Therefore, we assume that H contains both an (x, y)-path of length $\equiv \ell_1 \pmod 4$ and one of length $\equiv \ell_2 \pmod 4$. If such a graph exists, then it must be reversing-equivalent to $(F_n; x, y)$ shown in Figure 1. Since the inequality is easy to verify for each L in the absence of a long (x, y)-path, the code was designed under the assumption that such a path does exist.

To verify Proposition 2.3, it remains to show the following:

(A1) Suppose that $L = \{0, 3\}$. For $n = 6$, if $e(H) = 7$ then H is isomorphic to $(F_6; x, y)$.

(A2) Suppose that $L = \{0, 1\}$. For $n = 7$, if $e(H) = 9$ then H is reversing-equivalent to $(F_7; x, y)$.

(A3) Suppose that $L = \{1, 2\}$. For $n = 8$, if $e(H) = 11$ then H is reversing-equivalent to $(F_8; x, y)$.

(A4) Suppose that $L = \{2, 3\}$. For $n = 7$, $e(H) \leq 8$.

(A5) Suppose that $L = \{2, 3\}$. For $n = 9$, if $e(H) = 12$ then H is reversing-equivalent to $(F_9; x, y)$.

The file `main.py` processes the cases (A1)–(A5) by generating and filtering graphs with parameters corresponding to each case. The result files are named as `*_dedup_noC8.txt`.

- `exceed_*_dedup_noC8.txt` contains graphs H with $e(H) > t(n)$.
- `tight_*_dedup_noC8.txt` contains graphs with $e(H) = t(n)$

where $t(n)$ denotes the target number of edges. If such a file does not exist, then no such graph was found. Therefore, (A4) is true if and only if neither `tight_23_7_dedup_noC8.txt` nor `exceed_23_7_dedup_noC8.txt` exists. Furthermore, (A5) holds if and only if the file `exceed_23_9_dedup_noC8.txt` does not exist and every graph in `tight_23_9_dedup_noC8.txt` is reversing-equivalent to $(F_9; x, y)$. Since the number of graphs and the size of each graphs in `tight_23_9_dedup_noC8.txt` are small, it is feasible to manually verify that each graph is reversing-equivalent to $(F_9; x, y)$. The remaining cases, (A1), (A2), and (A3), can be verified in a similar manner to (A5).

2. File Descriptions

- `main.py`: Main execution file that runs all test cases and coordinates the workflow.
- `example_generator.py`: Core filtering logic, handles $e_{\max}$ filtering and writes graph files.
- `graph_generator.py`: Generates all graphs with a given set of fixed edges by enumerating over free edges.
- `graph_utils.py`: Implements graph condition checks: 4-cycle, path modulo 4, degree checks, etc.
- `dedup_tight_graphs.py`: Deduplicates graph files using NetworkX isomorphism checks.
- `filter_8cycles.py`: Further filters deduplicated graphs to remove those containing 8-cycles.

3. Execution Instructions

To run the project, execute:

```
python main.py
```

This will:

1. Generate graphs for various test cases
2. Save graphs in `tight_l1l2_n.txt` or `exceed_l1l2_n.txt`
3. Deduplicate and write `*_dedup.txt`
4. Filter 8-cycles to produce `*_dedup_noC8.txt`

4. Test Case Format

Each test case is of the form:

```
(l1, l2, i, j, k, fixed_edges_fn)
```

Where:

- ℓ_1, ℓ_2: included path lengths modulo 4
- i, j: range of n to consider
- k: edge threshold parameter, $e_{\max} = \lfloor (3n-k)/2 \rfloor$
- `fixed_edges_fn(n)`: function returning fixed edges for the given n

5. Output Files

- `tight_l1l2_n.txt`: Valid graphs with edge count = $e_{\max}$
- `tight_l1l2_n_dedup.txt`: Non-isomorphic graphs among the above
- `tight_l1l2_n_dedup_noC8.txt`: Further filtered to exclude 8-cycles
- Same for `exceed_l1l2_n.txt`, when graphs exceed $e_{\max}$

6. Dependencies

- Python 3.8+
- `networkx`
- `numpy`
- `os`

To install:

```
pip install networkx numpy os
```

7. Purpose

This tool was developed as part of a broader mathematical investigation into extremal structures in graphs, particularly those avoiding (k mod ℓ)-cycles and paths. The filtered graphs serve as gadgets in the study of structural theorems involving modular path lengths and cycle exclusions.