Counting Cycles with AI

Anonymous Authors

Abstract

Despite recent progress, AI still struggles with advanced mathematics. We consider a difficult open problem: How to derive a Computationally Efficient Equivalent Form (CEEF) for the cycle count statistic? The CEEF problem has no known general solution, and requires delicate combinatorics and tedious calculations. Such a task is hard to accomplish by humans but is an ideal example where AI can be very helpful. We solve the problem by combining our novel approach with the powerful coding skills of AI. Our results use delicate graph theory and contain new formulas for general cases that were not previously known. We find that, while AI is unable to solve the problem on its own, it is able to solve it if we provide it with a clear strategy, step-by-step guidance, and carefully written prompts. For simplicity, we focus our study on DeepSeek-R1 but we also investigate other representative LLMs.

Recent AI4Math work largely evaluates models on solved benchmarks (Hendrycks et al., 2021; Zheng et al., 2022; Davies et al., 2021; Drori et al., 2022; Trinh & *et al*., 2024; Lample et al., 2022); only a few studies target open mathematical problems (Romera-Paredes et al., 2024; Alfarano et al., 2024). By a novel humAI approach, i.e. a combination of human intelligence and AI capabilities, we solve one such problem: deriving a **C**omputationally **E**fficient **E**quivalent **F**orm (CEEF) for cycle count statistics. Our results use delicate graph theory and contain new formulas for general cases that were not previously known.

The CEEF problem involves delicate and extremely tedious combinatorics, and it is hard for humans to solve. In detail, for a symmetric $A \in \mathbb{R}^{n\times n}$ and an integer $m \geq 3$, the order-m cycle count (CC) statistic is

$$C_m = \sum_{\substack{i_1,i_2,\ldots,i_m \\ \text{distinct}}} A_{i_1 i_2} A_{i_2 i_3} \cdots A_{i_m i_1}. \tag{1}$$

As C_m does not depend on the main diagonal of A, we assume $A_{ii} = 0$ for $1 \leq i \leq n$ for simplicity. In the special case where A is the adjacency matrix of an undirected network (we call this the *binary case* below), $C_m/(2m)$ is the number of m-cycles in the network (Jin et al., 2021). This is why we call C_m the cycle count statistic. Cycle count statistics are useful in network analysis (Gao & Lafferty, 2017; Jin et al., 2021; 2023; 2025c), covariance matrix testing (Li & Chen, 2012), and estimating matrix spectrum (Kong & Valiant, 2017). Especially, high-order cycle count statistics are of interest, for they may yield better testing power or estimation accuracy than lower-order ones.

How to compute C_m is known as a challenging problem. A *brute-force* approach is to compute C_m using m-layers of for-loops (see (1)), but the resultant complexity is $O(n^m)$. For a less expensive approach, we may use the *combinatorial* approach (Harary, 1969; Perepechko & Voropaev, 2009), where we express C_m as the linear combination of finitely many terms, each having a computation cost much smaller than $O(n^m)$. For example, when $m = 8$, C_8 is a linear combination of 44 terms (see Table 1). To compute these terms, one term (marked in blue) has complexity $O(n^4)$, and all other terms have complexity $O(n^3)$. Thus, this formula computes C_m in $O(n^4)$ time, far below the $O(n^m)$ brute-force cost.

In Table 1, 43 terms have a *succinct expressive algebraic (SEA)* form, and one remaining term (marked in blue) is a full-sum (FS) term (meaning no constraint on the 4 indices over which we take the sum). In this paper, we may

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

Table 1. The formula of C_8 produced by our Python code, obtained by our approach.

+36	$\mathbf{1}_n' \cdot ((A \circ A \circ A \circ A \circ A \circ A \circ A \circ A) \cdot \mathbf{1}_n)$
-96	$\mathbf{1}_n' \cdot (((A \circ A) \cdot \mathbf{1}_n) \circ ((A \circ A \circ A \circ A \circ A \circ A) \cdot \mathbf{1}_n))$
-36	$\mathbf{1}_n' \cdot (((A \circ A \circ A \circ A) \cdot \mathbf{1}_n) \circ ((A \circ A \circ A \circ A) \cdot \mathbf{1}_n))$
-112	$\mathbf{1}_n' \cdot ((A \circ A \circ A \circ A \circ ((A \circ A) \cdot (A \circ A))) \cdot \mathbf{1}_n)$
+32	$\mathbf{1}_n' \cdot ((A \circ A \circ A \circ A \circ A \circ ((A \cdot A) \cdot A)) \cdot \mathbf{1}_n)$
+72	$\mathbf{1}_n' \cdot ((((A \circ A) \cdot \mathbf{1}_n) \circ ((A \circ A) \cdot \mathbf{1}_n)) \circ ((A \circ A \circ A \circ A) \cdot \mathbf{1}_n))$
+16	$\mathbf{1}_n' \cdot ((((A \circ A) \cdot ((A \circ A) \cdot \mathbf{1}_n)) \circ ((A \circ A \circ A \circ A) \cdot \mathbf{1}_n))$
+80	$\mathbf{1}_n' \cdot ((A \circ A \circ A \circ A \circ ((A \cdot A) \circ (A \cdot A))) \cdot \mathbf{1}_n)$
+32	$\mathbf{1}_n' \cdot (((A \circ A) \cdot \mathbf{1}_n) \circ ((A \circ A \circ A \circ A) \cdot ((A \circ A) \cdot \mathbf{1}_n)))$
+192	$\mathbf{1}_n' \cdot ((A \circ A \circ A \circ ((A \circ A \circ (A \cdot A)) \cdot A)) \cdot \mathbf{1}_n)$
+32	$\mathbf{1}_n' \cdot ((A \circ A \circ A \circ (((A \cdot A) \circ (A \circ A \circ A))) \cdot \mathbf{1}_n)$
+4	$\mathbf{1}_n' \cdot ((A \circ A \circ A \circ ((A \cdot (A \circ A \circ A)) \cdot A)) \cdot \mathbf{1}_n)$
+64	$\mathbf{1}_n' \cdot (((A \circ A) \cdot \mathbf{1}_n) \circ ((A \circ A \circ ((A \circ A) \cdot (A \circ A))) \cdot \mathbf{1}_n))$
+5	$\mathbf{1}_n' \cdot ((A \circ A \circ (((A \circ A) \cdot ((A \circ A) \cdot (A \circ A)))) \cdot \mathbf{1}_n)$
+22	$\Sigma_{i_1 i_2 i_3 i_4} (A \circ A)_{i_1 i_2} A_{i_1 i_3} A_{i_1 i_4} A_{i_2 i_3} A_{i_2 i_4} (A \circ A)_{i_3 i_4}$
-16	$\mathbf{1}_n' \cdot (((A \circ A \circ A \circ A \circ A) \cdot \mathbf{1}_n) \circ ((A \circ ((A \cdot A) \cdot A)) \cdot \mathbf{1}_n))$
-64	$\mathbf{1}_n' \cdot (((A \circ (A \cdot A)) \cdot \mathbf{1}_n) \circ ((A \circ A \circ A \circ (A \cdot A)) \cdot \mathbf{1}_n))$
-8	$\mathbf{1}_n' \cdot ((A \circ A \circ A \circ (A \cdot d((A \circ (A \cdot A)) \cdot \mathbf{1}_n) \cdot A)) \cdot \mathbf{1}_n)$
-64	$\mathbf{1}_n' \cdot ((A \circ A \circ A \circ ((A \cdot A) \cdot A)) \cdot ((A \circ A) \cdot \mathbf{1}_n))$
-16	$\mathbf{1}_n' \cdot ((A \circ A \circ A \circ (((A \cdot A) \cdot d((A \circ A) \cdot \mathbf{1}_n) \cdot A)) \cdot \mathbf{1}_n)$
-12	$\mathbf{1}_n' \cdot ((((A \circ A) \cdot \mathbf{1}_n) \circ ((A \circ A) \cdot \mathbf{1}_n)) \circ ((A \circ A) \cdot \mathbf{1}_n)) \circ ((A \circ A) \cdot \mathbf{1}_n))$
-16	$\mathbf{1}_n' \cdot ((((A \circ A) \cdot ((A \circ A) \cdot \mathbf{1}_n)) \circ ((A \circ A) \cdot \mathbf{1}_n)) \circ ((A \circ A) \cdot \mathbf{1}_n))$
-96	$\mathbf{1}_n' \cdot (((A \circ A) \cdot \mathbf{1}_n) \circ ((A \circ A \circ (A \cdot A) \circ (A \cdot A)) \cdot \mathbf{1}_n))$
-4	$\mathbf{1}_n' \cdot (((A \circ A) \cdot ((A \circ A) \cdot ((A \circ A) \cdot \mathbf{1}_n))) \circ ((A \circ A) \cdot \mathbf{1}_n))$
-24	$\mathbf{1}_n' \cdot ((A \circ A \circ (A \cdot d(((A \circ A) \cdot \mathbf{1}_n) \cdot A) \circ (A \cdot A)) \cdot \mathbf{1}_n)$
-32	$\mathbf{1}_n' \cdot ((A \circ A \circ A \circ (((A \circ (A \cdot A)) \cdot A)) \cdot (A \circ A)) \cdot \mathbf{1}_n)$
-64	$\mathbf{1}_n' \cdot ((A \circ A \circ A \circ (((A \circ (A \cdot A)) \cdot (A \circ (A \cdot A))) \cdot \mathbf{1}_n)$
-16	$\mathbf{1}_n' \cdot ((A \circ A \circ A \circ (((A \cdot A) \circ (A \cdot A)) \cdot (A \circ A))) \cdot \mathbf{1}_n)$
+8	$\mathbf{1}_n' \cdot ((A \circ A \circ A \circ (((((A \cdot A) \cdot A) \cdot A) \cdot A)) \cdot \mathbf{1}_n)$
+16	$\mathbf{1}_n' \cdot ((((A \circ A) \cdot \mathbf{1}_n) \circ ((A \circ A) \cdot \mathbf{1}_n)) \circ (((A \circ A) \cdot (A \cdot A)) \cdot \mathbf{1}_n))$
+8	$\mathbf{1}_n' \cdot (((A \circ A) \cdot ((A \circ A) \cdot \mathbf{1}_n)) \circ ((A \circ (((A \cdot A) \cdot A)) \cdot \mathbf{1}_n))$
+16	$\mathbf{1}_n' \cdot ((((A \circ A) \cdot \mathbf{1}_n) \circ ((A \circ A) \cdot \mathbf{1}_n)) \circ (((A \circ (((A \cdot A) \cdot A) \cdot A) \cdot A)) \cdot \mathbf{1}_n))$
+24	$\mathbf{1}_n' \cdot ((A \circ A \circ (((A \cdot A) \cdot A) \cdot A) \circ (A \cdot A)) \cdot \mathbf{1}_n)$
+12	$\mathbf{1}_n' \cdot ((A \circ A \circ (((A \cdot A) \cdot A) \circ (A \cdot (A \cdot A))) \cdot \mathbf{1}_n)$
+8	$\mathbf{1}_n' \cdot (((A \circ A) \cdot \mathbf{1}_n) \circ ((A \circ (((A \cdot A) \cdot A) \cdot A) \cdot ((A \circ A) \cdot \mathbf{1}_n)))$
+16	$\mathbf{1}_n' \cdot (((A \circ (A \cdot A)) \cdot ((A \circ A) \cdot \mathbf{1}_n)) \circ ((A \circ (A \cdot A)) \cdot \mathbf{1}_n))$
+4	$\mathbf{1}_n' \cdot (((A \circ A) \cdot \mathbf{1}_n) \circ ((A \circ ((A \cdot A) \cdot d((A \circ A) \cdot \mathbf{1}_n) \cdot A)) \cdot \mathbf{1}_n))$
+4	$\mathbf{1}_n' \cdot (((A \circ A) \cdot ((A \circ (A \cdot A)) \cdot \mathbf{1}_n)) \circ ((A \circ (A \cdot A)) \cdot \mathbf{1}_n))$
+24	$\mathbf{1}_n' \cdot ((A \circ (((A \cdot A) \cdot A) \circ (A \cdot A)) \circ (A \cdot A)) \cdot \mathbf{1}_n)$
+2	$\mathbf{1}_n' \cdot ((A \circ (((A \cdot A) \circ (A \cdot A)) \circ (A \cdot A)) \cdot A)) \cdot \mathbf{1}_n)$
-8	$\mathbf{1}_n' \cdot (((A \circ A) \cdot \mathbf{1}_n) \circ ((A \circ (((A \cdot A) \cdot A) \cdot A) \cdot A)) \cdot \mathbf{1}_n))$
-8	$\mathbf{1}_n' \cdot (((A \circ (((A \cdot A) \cdot A) \cdot A)) \cdot \mathbf{1}_n) \circ ((A \circ (A \cdot A)) \cdot \mathbf{1}_n))$
-4	$\mathbf{1}_n' \cdot (((A \circ ((A \cdot A) \cdot A)) \cdot \mathbf{1}_n) \circ ((A \circ ((A \cdot A) \cdot A)) \cdot \mathbf{1}_n))$
+1	$\mathbf{1}_n' \cdot ((A \circ ((((((A \cdot A) \cdot A) \cdot A) \cdot A) \cdot A)) \cdot \mathbf{1}_n)$

encounter many terms like these 43 terms; we call them *SEA* terms. Moreover, we call a FS term an *incompressible full-sum (IFS) term* if we cannot rewrite it as a FS term with fewer layers of sum. The FS term marked in blue in Table 1 is seen to be an IFS term. To solve the CEEF problem, the goal is to derive a formula that writes C_m as a linear combination of finitely many terms, each of which is either a SEA term or an IFS term (ideally, we hope all terms are SEA terms; this is possible when $m \leq 7$, but impossible when $m \geq 8$). See Section A of the supplement for the formal definitions of SEA, FS, IFS, and CEEF.

When m is relatively small, we can derive such a formula *explicitly by hand*. For binary matrix A, (Harary, 1969) solved the case of $m = 4, 5$, (Chang & Fu, 2003) solved the case of $m = 6$, (Movarraei & Boxwala, 2016) solved the case of $m = 7$, and (in a remarkable paper) (Perepechko & Voropaev, 2009) solved the case of $3 \leq m \leq 13$. For general matrix A, (Jin et al., 2021) solved the case of $m = 3, 4$. Unfortunately, for larger m, deriving the formula by hand is no longer feasible. For instance, when $m = 12$, the formula for C_m is a linear combination of 1900 terms. To derive the formula in such cases, we need delicate and extremely tedious calculations, and it is easy to make mistakes (e.g., as pointed out by (Perepechko & Voropaev, 2009), (Harary, 1969) made a mistake for the case of $m = 7$). For these reasons, such a task is simply too hard for humans, and the problem remains a long-lasting unsolved problem.

Recent developments of AI provide a great opportunity. Especially, for problems that need delicate and tedious combinatorics, AI may offer advantages for implementation and verification. However, if we directly ask AI to generate the desired formula for a given m, it usually outputs a result far from correct. We may use multi-shot learning (e.g., (Brown et al., 2020; Wei et al., 2022)), where we provide AI with the formulas for smaller m. But since the formula rapidly gets more complicated as m increases, such an approach does not work either. To address the challenge, we propose a novel humAI approach, where we combine the strengths of humans and AI. While many AI tools are available, we focus primarily on DeepSeek-R1 (DS), which has been shown to be competitive in solving mathematical problems (e.g., (Evstafev, 2024)). For completeness, we also investigate several other LLMs using the same prompts; see Table 2.

Method. Our approach has two steps. Fix a set of m distinct indices $S = \{i_1, i_2, \ldots, i_m\}$ and let $\mathcal{G}$ be the (undirected simple) graph with edges between $i_1\&i_2, i_2\&i_3, \ldots, i_m\&i_1$, but nowhere else. In the first step, we identify all multi-graphs induced by $\mathcal{G}$ through a merging process on S. We then divide all multi-graphs into different classes according to isomorphism and pick one from each class. For each size k, let $\{\mathcal{G}_{m,k,t} : 1 \leq t \leq b_{m,k}\}$ be the resulting class representatives, let $\sigma_{m,k,t} = \{S_1, \ldots, S_k\}$ be a partition inducing $\mathcal{G}_{m,k,t}$, and define

$$\begin{aligned} d_{m,k,t} &= \text{size of class } t \text{ of size-}k \text{ multi-graphs}, \\ h_{m,k,t} &= \Pi_{i=1}^{k}(|S_i| - 1)!. \end{aligned} \tag{2}$$

If $w(a, b)$ is the edge weight between nodes a and b in $\mathcal{G}_{m,k,t}$, define

$$f_{m,k,t}(A, j_1, \ldots, j_k) = \Pi_{\{a<b:w(a,b)>0\}} A_{j_a j_b}^{w(a,b)}. \tag{3}$$

Theorem 2.1. *Fix $m \geq 3$. Let $d_{m,k,t}$ and $h_{m,k,t}$ be as in (2), and suppose $\mathcal{G}_{m,k,t}$ is the induced multi-graph from the partition $\sigma_{m,k,t}$. We have $\mu(\hat{0}, \sigma_{m,k,t}) = (-1)^{m-k} \cdot h_{m,k,t}$ and*

$$C_m = \sum_{k=2}^{m} \sum_{t=1}^{b_{m,k}} a_{m,k,t} \sum_{j_1,\ldots,j_k} f_{m,k,t}(A, j_1, \ldots, j_k),$$

where

$$a_{m,k,t} = d_{m,k,t} \cdot \mu(\hat{0}, \sigma_{m,k,t}) = (-1)^{m-k} \cdot d_{m,k,t} \cdot h_{m,k,t}.$$

In this step, DS made a remarkable contribution in (a) generating a script to correctly identify all multi-graphs, (b) presenting each multi-graph visually, (c) checking isomorphism, and (d) suggesting that the Möbius function can be useful to find the linear coefficients of the desired formula.

In the second step, we relate each FS term to a labeled multi-graph, where each layer in the sum corresponds to a node in the graph. We propose a recursive pruning algorithm where 'pruning a node' is equivalent to 'reducing a layer in the sum without changing the value of the sum'.

Theorem 2.2. *When the algorithm stops, we only have two possible cases in the final LMG: (a) there is only one node left, or (b) no pendant node of either type is left. Suppose we have the first case. Let v be the label for the only one node there. Then $\sum_{j_1,\ldots,j_k} f(A, j_1, \ldots, j_k) = \mathbf{1}_n' v$, which is a SEA term. Suppose we have the second case. In this case, we have at least 4 nodes remaining, each of them has at least 3 immediate neighbors, and $\sum_{j_1,\ldots,j_k} f(A, j_1, \ldots, j_k)$ reduces to an FS term with ℓ-layer of sum, $\ell \leq \lfloor m/2 \rfloor$.*

In this step, DS also made a remarkable contribution in (a) writing a script to implement our algorithm (which is hard to implement without advanced coding skills), and (b) successfully converting each FS term to a SEA term or an IFS term as expected. We also investigate our humAI approach with several other representative LLMs by feeding in the same prompts we use for DS, up to some minor changes; see Table 2.

We now have a clear strategy for deriving a formula as desired for each C_m, but executing this strategy is labor-intensive. For example, the formula for C_{12} has 1900 terms,

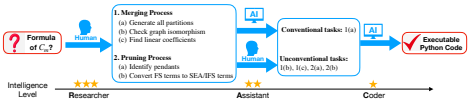


Figure 1. The pipeline of our humAI approach for counting cycles.

and deriving the explicit formula in such a case is challenging for humans. A straightforward use of AI does not work: when we ask LLMs to reason in natural language or write code to derive a formula for C_m, they succeed only for very small m (e.g., $m = 3, 4, 5$), where correct answers exist in the literature. When $m \geq 6$, the desired formula either is not well-known or does not exist in the literature, and LLMs fail to output a satisfactory result: they often guess a few terms with incorrect coefficients and lack a correct reasoning approach. This suggests that AI is unable to solve the CEEF problem independently.

Figure 1 summarizes our pipeline: humans divide the CEEF problem into steps and outline a strategy, while AI helps write code for step-specific algorithms and then produces executable Python code whose input is m and whose output is the SEA/IFS formula for C_m in LaTeX code. These roles require different levels of intelligence, which we call level-R (Researcher), level-A (Assistant), and level-C (Coder), respectively.

For simplicity, we focus on DeepSeek-R1 (DS-R1), a recent reasoning model with competitive performance in benchmark tasks (Guo et al., 2025), but we also investigate other LLMs. First, we prescribe the format for input and output. For input, we teach DS-R1 to express graphs in strings. For output, we ask DS-R1 to format the output as LaTeX code that prints each term as either a SEA term or an IFS term. Next, we divide our project into 5 major tasks, 3 for the merging process (1(a), 1(b), 1(c)), 2 for the pruning process (2(a), 2(b)). See Figure 1. Among these tasks, according to difficulty level, 1(a) is a *conventional task* for AI, and others are *unconventional tasks* for AI.

Task 1(a) is well-studied in the literature and AI is able to accomplish it without much struggle. For this task, we only provide a verbal description in the prompt and DS-R1 is able to find an appropriate algorithm on its own. This demonstrates that DS-R1 has *level-A (Assistant) intelligence*. The other 4 tasks are unconventional for AI, and for each of them, AI needs detailed guidance and carefully written prompts. For Task 1(b), DS-R1 does not know what *isomorphism* means in the current setting, so we need to inform it with the precise definition first. After that, DS-R1 knows how to check isomorphism in a brute-force fashion, but the algorithm is very slow. To address the problem, we inform DS-R1 that for checking isomorphism, we only need to consider degree-preserved permutations. This significantly speeds up the computing. For Task 1(c), despite that the Möbius function is well-known, we must teach DS-R1 how to relate it to our problem. As an example, we inform DS-R1 by prompts that we only need to consider the pairs of partitions $\mu(\pi, \sigma)$ with $\pi = \hat{0}$ (the finest partition). With step-by-step guidance, DS-R1 is able to produce the right results. For Tasks 2(a)-2(b), since SEA and IFS terms are new definitions, we need to inform DS-R1 with the precise definition first. Our prompt starts with the prescribed input format, followed by detailed instructions on both pruning types. See Sections B and E of the supplement for the full graph-theoretic formulation, algorithms, and proofs.

Results and validation. The final result of our method is a Python code to generate each C_m. The input is an integer $m \geq 3$, and the output is the desired formula. In this formula, each FS term is converted to either a SEA term or an IFS term. For validation, we use two approaches. In the first approach, we manually compare our derived formula with those derived by hand for $m = 3, 4, \ldots, 8$ and those by (Perepechko & Voropaev, 2009) for $m = 3, 4, \ldots, 13$ (where A is binary, so we need to first simplify our formula for this special case). In the second approach, for a relatively small n, we generate many $n \times n$ symmetric matrices A and compute C_m for each in a brute-force fashion. We then use the results to validate the formula. Based on long and tedious validation, we conclude that the AI-produced output formulas are correct. Below, we report the term counts and computational complexities for the combinatorial formulas derived using the humAI approach.

m	3	4	5	6	7	8	9	10	11	12
#Terms (a_m)	1	3	3	10	15	44	89	254	633	1900
#SEA terms	1	3	3	10	15	43	86	239	581	1666
#$O(n^4)$ IFS terms						1	3	13	46	190
#$O(n^5)$ IFS terms								2	6	37
#$O(n^6)$ IFS terms										7

In summary, first, DS-R1 is able to accomplish conventional tasks, with correct answers. This demonstrates that DS-R1 has *level-A (Assistant) intelligence*. Second, DS-R1 is able to accomplish unconventional tasks, provided with step-by-step guidance and carefully written prompts. This demonstrates that DS-R1 has *level-C (Coder) intelligence* for these unconventional tasks: it is able to accomplish the task, but needs detailed step-by-step guidance. Somewhat surprisingly, DS-R1 also has preliminary level-R Researcher intelligence. One example is that, when we tried to prove Theorem 2.1 for the first time, we didn't realize it was connected to the Möbius function. DS-R1 discovered this connection independently and we solved the problem following its hint. Another example is, DS-R1 is able to detect inconsistency in our prompt. In one of our prompts, we provided DS-R1 with both a formula for the linear coefficients and an example for the linear coefficients, but unfortunately, the coefficients in the examples are not consistent with those in the definition. DS-R1 detected such an inconsistency. Using

its hint, we revised the prompt, and DS-R1 then produced the correct answer. See Section C of the supplement for implementation details, formula-generation output, validation, runtime, and model-comparison setup.

Investigation with other LLMs. While our focus is on DS-R1, it is also of interest to examine representative LLMs under the same prompts. We test 11 models spanning seven providers. To make the comparison diagnostic, we decompose the prompt-response process into two stages: Stage 1 asks the model to enumerate partitions of the m-cycle, filter self-loops, group multi-graphs by isomorphism, and compute the aggregated Möbius coefficient; Stage 2 asks the model to apply pendant pruning and construct the SEA/IFS formula. Stage 1 is verified against ground-truth graph types and coefficients, and Stage 2 by the final formulas, using manual checks for small m and numerical verification against the dynamic-programming benchmark. The results are in Table 2.

Table 2. Evaluation of 11 LLMs. A ✓ indicates success throughout the reported validation range, and a ✗ indicates at least one failure.

Model	Stage 1: Merging				Stage 2: Pruning		Summary
	Syntax(1)	1a	1b	1c	Syntax(2)	2a/2b	
Claude Opus 4.8	✓	✓	✓	✓	✓	✓	Successful
Claude Sonnet 4.6	✓	✓	✓	✓	✓	✓	Successful
DeepSeek V4 Pro	✓	✓	✓	✓	✓	✓	Successful
GPT-5.5	✓	✓	✓	✓	✓	✓	Successful
Gemini 3.1 Pro	✓	✓	✓	✓	✓	✓	Successful
Gemini 3.5 Flash	✓	✓	✓	✓	✓	✓	Successful
GLM 5.1	✓	✓	✓	✓	✓	✓	Successful
Kimi K2.6	✓	✓	✓	✓	✓	✓	Successful
MiniMax-M3	✓	✓	✓	✓	✓	✓	Successful
MiniMax-M2.7	✓	✓	✓	✓	✗	✗	Stage 1 only
DeepSeek V3.2	✗	✗	✗	✗	✗	✗	Unsuccessful

Key findings are as follows. Ten of the eleven models successfully complete Stage 1; for instance, the successful Stage-1 models correctly report 41 surviving partitions and 10 canonical types at $m = 6$, and 715 surviving partitions and 44 canonical types at $m = 8$. Stage 2 further separates the models: nine models that pass Stage 1 also produce numerically correct formulas in the reported validation range. MiniMax-M2.7 passes the Stage-1 structural checks but its Stage-2 code cannot be validated because the generated program fails to import, while DeepSeek V3.2 fails already in Stage 1 and its Stage-2 output also fails the numerical checks. This pattern illustrates the diagnostic value of separating Merging from Pruning: a Stage-2 failure can be interpreted differently depending on whether the canonical Stage-1 input has first been verified. See Sections C.4, and C.1 of the supplement for the full comparison protocol, model identifiers, and benchmark prompts.

Applications. Using our derived formulas, we revisit three problems where high-order cycle information is useful. First, in weak-spike testing, the cycle-count method is substantially more effective than the empirical-eigenvalue method in the weak-signal regime: for $n = 1000$, the ideal CC error decreases from about 0.410 at $m = 1$ to about 0.130 at $m = 12$, while EE remains around 0.390; for $n = 500$, the best CC error is about 0.250 at $m = 12$, whereas EE is about 0.430. Second, in spike estimation, the Newton–Prony estimator built from cycle statistics improves as more cycle information is used: in the reported $n = 1000$ experiment, its normalized MSE decreases from about 0.532 at $m = 2$ to about 0.075 at $m = 12$, while the direct EE estimator has normalized MSE about 0.443. Third, in a higher-order pairwise IBM simulation under an undirected DCMM design, the standardized statistics $\Phi_2, \ldots, \Phi_6$ show increasing separation between the null and alternative distributions as the order grows, both at target SNR $= 2$ and at target SNR $= 1.5$. These experiments support the same qualitative conclusion: higher-order cycle statistics become practically useful once the CEEF formulas make them computationally accessible. See Section D of the supplement for the full experimental setups and figures.

In summary, our humAI approach solves the long-standing CEEF problem for cycle count statistics; see Section C of the supplement for the formulas for $m = 3, 4, \ldots, 12$. AI does not solve this problem independently, but with a clear strategy, step-by-step guidance, and carefully written prompts, it becomes an effective research assistant for difficult open mathematical problems.

Our contribution is two-fold. First, we provide evidence that AI can be useful in research-level mathematics when guided by human mathematical structure. Second, compared with existing works (e.g., (Perepechko & Voropaev, 2009; Harary & Manvel, 1971)), we use new graph-theoretic ideas to derive computationally efficient formulas for C_m, $m \geq 3$, which are useful in network analysis (Gao & Lafferty, 2017; Jin et al., 2021; 2023; Kong & Valiant, 2017), spiked-eigenvalue estimation (Kong & Valiant, 2017), matrix testing (Li & Chen, 2012), and related problems.

Supplementary Material

This supplementary material consists of the detailed proofs, validation details, model-comparison information, applications, and benchmark prompts corresponding to the main section above.

A. Preliminaries

In this section we introduce the notation used throughout the supplement. The cycle count statistic C_m is defined in (1); throughout the supplement we use the same zero-diagonal convention for A as in the main manuscript. If A has nonzero diagonal entries, the formulas are applied to $A - \mathrm{diag}(A)$. $\circ$ denotes the Hadamard product. For $n > 1$, I_n denotes the $n \times n$ identity matrix and $\mathbf{1}_n \in \mathbb{R}^n$ denotes the vector of all ones. For $v \in \mathbb{R}^n$, $d(v)$ or $\mathrm{diag}(v)$ denotes the diagonal matrix whose i-th diagonal entry is v_i; for $M \in \mathbb{R}^{n\times n}$, $d(M)$ or $\mathrm{diag}(M)$ denotes the diagonal matrix constructed from the diagonal entries of M.

Definition 1 (SEA term)**.** We call a term a *Succinct Expressive Algebraic (SEA)* term if it is an elementary function of A and $\mathbf{1}_n$ constructed using only three operations: (i) multiplication (applied to two matrices or a matrix and a vector), (ii) entry-wise product (applied to two matrices or two vectors), and (iii) diagonal extraction (applied to a matrix).

Definition 2 (FS term)**.** We call a term a K-layer *Full Sum (FS)* term if it has the form of

$$\sum_{i_1,i_2,\ldots,i_K} X^{(1)}_{j_1,k_1} X^{(2)}_{j_2,k_2} \ldots X^{(N)}_{j_N,k_N}, \qquad K \geq 1, N \geq 1,$$

where each $X^{(s)}$ is an $n \times n$ matrix that is an elementary function of A constructed using only matrix multiplication, entry-wise product, and diagonal extraction, and $j_1, k_1, \ldots, j_N, k_N$ take values in $\{i_1, i_2, \ldots, i_K\}$ with $j_\ell \neq k_\ell$, $1 \leq \ell \leq N$.

Definition 3 (IFS term)**.** We say a K-layer FS term is an *Incompressible Full Sum (IFS) term* if it cannot be written equivalently as an L-layer FS term for any $L < K$.

Any SEA term can be computed through matrix operations with cost dominated by matrix multiplication, while a K-layer FS term has direct summation cost $O(n^K)$. The pruning procedure below converts the FS terms produced by the merging step into SEA terms or IFS terms whenever possible.

Definition 4 (CEEF)**.** For each $m \geq 3$, a *Computationally Efficient Equivalent Form (CEEF)* of C_m refers to a formula that decomposes C_m into a linear combination of finitely many SEA terms and IFS terms, with explicit linear coefficients.

The remaining sections provide material omitted from the compressed workshop manuscript. Section B gives the graph-theoretic algorithm and its theoretical justification, Section C gives the AI implementation, validation, runtime information, and LLM comparison, Section D gives the full application experiments, and Section E gives the proofs.

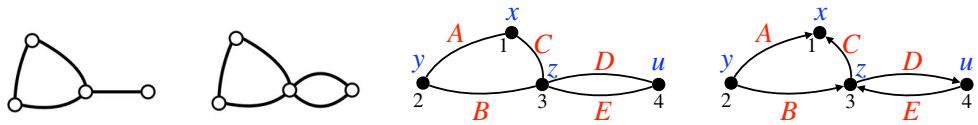
B. Theoretical results

This section contains the theoretical results of the paper, including main theorems and algorithms. Discussion on the AI-part contribution is deferred to Section C. Our goal here is to develop an algorithm for deriving the CEEF formula of C_m for any given m. We begin in Section B.1 by introducing a new concept in graph theory, FAM, which plays a key role in our results. The proposed algorithm consists of two steps, a merging step and a pruning step, described in Section B.2 and Section B.3, respectively.

B.1. The FAM and the associated full sum

In graph theory (Harary, 1969; Bondy & Murty, 1991), a simple graph is a collection of nodes and edges such that, between any pair of distinct nodes, there is either no edge or exactly one undirected edge, and self-loops (i.e., edges connecting a node to itself) are not allowed. A multi-graph relaxes these restrictions by allowing multiple parallel edges between two nodes, as well as self-loops. In Figure 2, the left two panels illustrate the simple graph and multi-graph, respectively.

Throughout this paper, simple graphs are assumed to be undirected, whereas multigraphs may be either undirected or directed. For a directed multigraph, each edge from node a to node b is represented by a triplet (a, b, s), where $1 \leq s \leq w(a, b)$ and $w(a, b)$ denotes the number of edges from a to b. For an undirected multigraph, edges between nodes a and b are likewise represented by triplets (a, b, s) with $1 \leq s \leq w(a, b)$. In this case, the ordering of a and b does not matter, and $w(a, b) = w(b, a)$ always holds. We write $E(\mathcal{G})$ for the set of all triplets (a, b, s) such that $w(a, b) \geq 1$.



$$\mathrm{FS}(G) = \sum_{1\le j_1,\dots,j_4\le n} x_{j_1} y_{j_2} z_{j_3} u_{j_4} \cdot A_{j_2 j_1} B_{j_2 j_3} C_{j_3 j_1} D_{j_3 j_4} E_{j_4 j_3}$$

Figure 2. A simple graph (left), a multi-graph (middle left), an undirected FAM (middle right), and a directed FAM (right). In the two FAMs, nodes are labeled by $x, y, z, u \in \mathbb{R}^n$, and edges are labeled by $A, B, C, D, E \in \mathbb{R}^{n\times n}$. The corresponding Full Sums (FSs) are displayed below these graphs. For an undirected FAM, the edge-label matrices are symmetric, so expressions such as $A_{j_2 j_1}$ may be replaced by $A_{j_1 j_2}$ without changing the value of the associated FS. In contrast, such substitutions are generally not valid for directed FAMs.

We propose the *Fully Annotated Multi-graph (FAM)*, as well as the Full Sum and Distinct Sum associated with the FAM. These definitions play a central role in our results.

Definition 5 (Fully Annotated Multi-graph (FAM))**.** A FAM $\mathcal{G}$ is a multi-graph whose nodes are labeled by vectors $v^{(a)} \in \mathbb{R}^n$ and whose edges (a, b, s) are labeled by matrices $M^{(a,b,s)} \in \mathbb{R}^{n\times n}$. For an undirected FAM, all edge-label matrices are required to be symmetric.

Definition 6 (FS and DS associated with a FAM)**.** Given an FAM $\mathcal{G}$, the Full-Sum (FS) associated with $\mathcal{G}$ is defined as

$$\mathrm{FS}(\mathcal{G}) := \sum_{1\le j_1,\dots,j_k\le n} \left(\prod_{1\le a\le k} v^{(a)}_{j_a} \prod_{(a,b,s)\in E(\mathcal{G})} M^{(a,b,s)}_{j_a j_b} \right),$$

where v_j denotes the jth entry of a vector $v \in \mathbb{R}^n$ and $M_{jj'}$ denotes the (j, j')th entry of a matrix $M \in \mathbb{R}^{n\times n}$. The Distinct-Sum (DS) associated with $\mathcal{G}$, $\mathrm{DS}(\mathcal{G})$, is defined similarly, except that $j_1, \dots, j_k$ are required to be distinct in the sum.

In Figure 2, the right two panels illustrate an undirected FAM and a directed FAM, respectively. Both contain four nodes and six edges. Their associated FS expressions are nearly identical; the key difference is that, for an undirected FAM, terms such as $A_{j_2 j_1}$ and $B_{j_2 j_3}$ may be replaced by $A_{j_1 j_2}$ and $B_{j_3 j_2}$, respectively, without changing the value of the FS, because the edge-label matrices are symmetric. In contrast, such substitutions are generally not valid for directed FAMs.

Note that $\mathrm{FS}(\mathcal{G})$ can be viewed as a special case of the FS expression in Definition 2, in which certain matrices $X^{(s)}$ are restricted to be diagonal and the associated indices satisfy $j_s = k_s$.

The above definitions connect to our problem through a particular FAM constructed as follows. Fix $m \ge 3$ and a set of m distinct indices $S = \{i_1, i_2, \dots, i_m\}$. Let $\mathcal{G} = \mathcal{G}(S)$ denote the simple graph whose edges are (i_1, i_2), (i_2, i_3), …, (i_{m-1}, i_m), and (i_m, i_1), with no other edges. Equivalently, $\mathcal{G}$ is the cycle graph on the vertex set S. We refer to $\mathcal{G}$ as the root graph associated with S.

Definition 7 (The root-FAM)**.** Fix m and S, and let $\mathcal{G}$ be the root graph as described above. We define the root FAM associated with S to be the FAM obtained by labeling each node of $\mathcal{G}$ with $\mathbf{1}_n$ and each edge with the matrix A (the original data matrix). By construction, every root FAM is undirected.

We observe that the FS and DS associated the root-FAM are

$$\mathrm{FS}(\mathcal{G}) = \sum_{j_1,\dots,j_m} A_{j_1 j_2} A_{j_2 j_3} \dots A_{j_m j_1} = \mathrm{trace}(A^m), \qquad \mathrm{DS}(\mathcal{G}) = \sum_{j_1,\dots,j_m (dist)} A_{j_1 j_2} A_{j_2 j_3} \dots A_{j_m j_1} = C_m. \tag{4}$$

This illustrates how these concepts are intrinsically connected to our problem. Note that although $\mathcal{G}$ depends on S, the quantities $\mathrm{FS}(\mathcal{G})$ and $\mathrm{DS}(\mathcal{G})$ do not depend on S.

B.2. The merging process

In the merging process, we aim to express C_m as a linear combination of finitely many FS terms. In detail, let $m \ge 3$ and $S = \{i_1, i_2, \dots, i_m\}$, where $i_1, i_2, \dots, i_m$ are distinct. Recall that $\mathcal{G} = \mathcal{G}(S)$ denotes the root-FAM associated with S.

Definition 8 (Partition)**.** Fix $k \leq m$. We call $\sigma = \{S_1, S_2, \ldots, S_k\}$ a k-partition of S if $S_1, S_2, \ldots, S_k$ are pairwise disjoint non-empty subsets of S satisfying $\cup_{j=1}^{k} S_j = S$. We call k the size of the partition (denoted by $|\sigma|$) and each S_j a block. The set of all partitions for S are denoted by $\Pi(S)$.

Given each $\sigma \in \Pi(S)$, we can create a new FAM from the root FAM, by node merging according to the partition σ.

Definition 9 (The induced FAM by a partition)**.** Fix $1 \leq k \leq m$ and a k-partition $\sigma = \{S_1, S_2, \ldots, S_k\}$. The induced FAM associated with σ is the undirected FAM with node set $\{j_1, j_2, \ldots, j_k\}$. Its edges are induced by those of the root FAM as follows: an edge (a, b) in the root FAM induces an edge (j_ℓ, j_t) whenever $a \in S_\ell$ and $b \in S_t$. Consequently, the induced FAM may contain parallel edges and self-loops. Each node is still labeled by $\mathbf{1}_n$, and each edge is still labeled by the data matrix A. We call $k = |\sigma|$ the size of the induced FAM. For $1 \leq \ell \leq k$, the quantity $|S_\ell|$ is called the multiplicity of node j_ℓ.

By this definition, for each $1 \leq k \leq m$ and each k-partition, we obtain a size-k induced FAM (when $k = m$, it is the root FAM itself). We call such a process the *merging process*.

Example 1. Suppose $m = 4$ and $S = \{i_1, i_2, i_3, i_4\}$. We consider a partition $\sigma = \{\{i_1\}, \{i_2, i_4\}, \{i_3\}\}$. The induced FAM has three nodes $\{j_1, j_2, j_3\}$, where j_2 is obtained from merging i_2 and i_4. After merging, there are two parallel edges between j_1 and j_2, and two parallel edges between j_2 and j_3; all node labels are $\mathbf{1}_n$ and all edge labels are A. The associated FS term is

$$\mathrm{FS}(\mathcal{G}_\sigma) = \sum_{j_1, j_2, j_3} (A_{j_1 j_2})^2 (A_{j_2 j_3})^2 = \sum_{j_1, j_2, j_3} (A \circ A)_{j_1 j_2} (A \circ A)_{j_2 j_3} = \mathbf{1}_n' (A \circ A)^2 \mathbf{1}_n.$$

There are a total of 9 partitions for $m = 4$. However, many of the induced FAMs have self-loops. Since A has a zero diagonal, the associated FS term of such a FAM is always zero. Therefore, we restrict our attention to those induced FAMs that have no self-loop. Figure 3 illustrates two such induced FAMs.

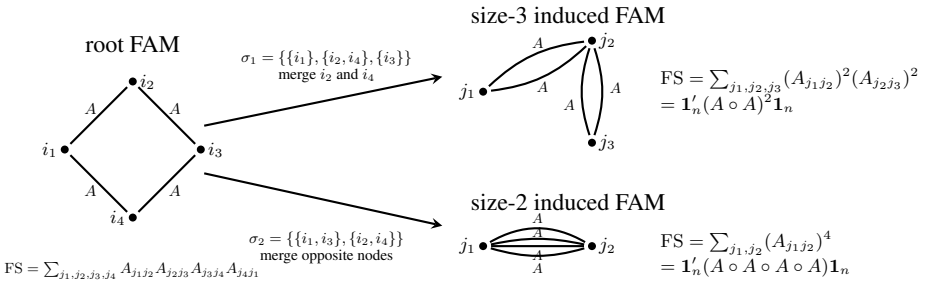


Figure 3. A partial illustration of the merging process for $m = 4$. The process starts from the root FAM (a 4-cycle) and considers all partitions. Each partition yields an induced FAM. Only two induced FAMs that have no self-loops are shown here.

Definition 10. We say two induced FAMs are isomorphic if one can be obtained from the other by a permutation of the nodes.

The merging process identifies all induced FAMs for the root FAM. Some of these induced FAMs are isomorphic. When two FAMs are isomorphic, it follows by Definition 6 that their corresponding FS terms must be equal. We thus divide all these induced FAMs into different classes, first by size and then by isomorphism. In detail, we propose Algorithm 1.

We present our main theorem about the merging step.

Theorem 1. *Fix $m \geq 3$. Let $\mathcal{G}_{m,k,t}$, $d_{m,k,t}$ and $h_{m,k,t}$ be as in Algorithm 1 for $2 \leq k \leq m$ and $1 \leq t \leq b_{m,k}$. Let $\sigma_{m,k,t}$ be the partition that induces the FAM $\mathcal{G}_{m,k,t}$. Define $\mu(\hat{0}, \sigma_{m,k,t}) = (-1)^{m-k} \cdot h_{m,k,t}$ and $a_{m,k,t} = d_{m,k,t} \cdot \mu(\hat{0}, \sigma_{m,k,t})$. Then,*

$$C_m = \sum_{k=2}^{m} \sum_{t=1}^{b_{m,k}} a_{m,k,t} \cdot \mathrm{FS}(\mathcal{G}_{m,k,t}).$$

This theorem provides a rigorous computational formula of C_m: It expresses C_m as a linear combination of finitely many FS terms, where all linear coefficients are explicitly computable. By Definition 6, $\mathrm{FS}(\mathcal{G}_{m,k,t})$ involves k layers of sum and can be computed with $O(n^k)$ cost. Since k ranges from 2 to m, this formula alone does not reduce the computational complexity compared to the brute-force approach. To complete the story, we will develop a pruning process in Section B.3, which converts each $\mathrm{FS}(\mathcal{G}_{m,k,t})$ equivalently to either an SEA term or an IFS term.

Algorithm 1 The merging process .

Input: $S = \{i_1, i_2, \ldots, i_m\}$.

a. List all partitions in $\Pi(S)$. For each partition, plot the induced FAM and record the size.

b. Exclude all induced FAMs with self-loops (because any induced size-1 FAM must have self-loops and are excluded, the size of an induced FAM ranges from 2 to m).

c. Divide all remaining FAMs into $(m-1)$ groups, where the k-th group contains all FAMs with size k, for $2 \le k \le m$ (when $k = m$, group k has only one member, which is the root-FAM $\mathcal{G}$).

d. Fix $2 \le k \le m$. Partition the k-th group into equivalence classes such that two FAMs belong to the same class if and only if they are isomorphic. List all classes in some fixed order (e.g. lexicographical order), and let

$$\begin{aligned} b_{m,k} &= \text{the total number of classes in the } k\text{-th group}, \\ d_{m,k,t} &= \text{cardinality of class } t \text{ (i.e., the number of FAMs contained in that class)}, \quad 1 \le t \le b_{m,k}. \end{aligned} \tag{5}$$

Now, all induced FAMs are partitioned into classes according first to their size and then to graph isomorphism. Each class is indexed by a pair (k, t), where $2 \le k \le m$ and $1 \le t \le b_{m,k}$.

e. For each class (k, t), select a candidate FAM denoted by $\mathcal{G}_{m,k,t}$. Define $h_{m,k,t}$ by

$$h_{m,k,t} = \Pi_{i=1}^{k}(g_i - 1)! \quad \text{if } k < m, \qquad \text{and} \qquad h_{m,k,t} = 1 \quad \text{if } k = m. \tag{6}$$

Output: $b_{m,k}$ for $2 \le k \le m$, and $(\mathcal{G}_{m,k,t}, d_{m,k,t}, h_{m,k,t})$ for $2 \le k \le m$ and $1 \le t \le b_{m,k}$.

Remark 1 (Connection to the Möbius function). In Theorem 1, $\mu(\cdot, \sigma_{m,k,t})$ coincides with the Möbius function on a partition lattice, and $\hat{0}$ refers to the finest partition. Initially, we were not aware that the linear coefficients in the decomposition of C_m are related to the Möbius function. However, when we guided the AI (specifically, DeepSeek-R1) to compute these coefficients, AI hinted us that the Möbius inversion technique might be relevant. Indeed, our proof of Theorem 1 involves deriving an explicit form of the Möbius function; see Section E.3.

Remark 2 (Self-loops). Algorithm 1 excludes induced FAMs that contain self-loops. Indeed, when a FAM has self-loops, each summand in the associated Full Sum (see Definition 6) involves diagonal entries of A, which are assumed to be zero in our setting (otherwise, we plug in $A - \mathrm{diag}(A)$). As a result, the Full Sum is zero and omitted in the computation circuit.

We note that C_m is the Distinct-Sum associated with the root FAM $\mathcal{G}$; see (4). In this sense, Theorem 1 expresses a DS-term as a linear combination of FS-terms. One may wonder whether the opposite is true. The following corollary shows that $\mathrm{tr}(A^m)$, the Full-Sum associated with the root FAM, can be expressed as a linear combination of DS-terms on the induced FAMs.

Corollary 2. *For any* $m \ge 3$,

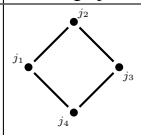
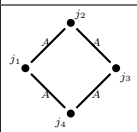
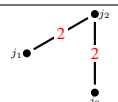
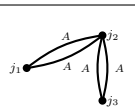
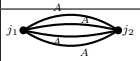
$$C_m = \mathrm{tr}(A^m) - \sum_{k=2}^{m-1} \sum_{t=1}^{b_{m,k}} d_{m,k,t} \cdot \mathrm{DS}(\mathcal{G}_{m,k,t}).$$

Although we do not us this corollary directly in computation, it provides another intuitive decomposition of C_m.

Example 2. When $m = 4$, $b_{m,k} = 1$ for $k = 2, 3, 4$; i.e., each group has one equivalence class. Table 3 shows the induced FAMs that represent the three classes. Suppose $\mathcal{G}_{m,k,t}$ has nodes $\{j_1, \ldots, j_k\}$. By Definition 6, there is a function $f_{m,k,t}(A, j_1, \ldots, j_k)$ such that $\mathrm{FS}(\mathcal{G}_{m,k,t}) = \sum_{j_1,\ldots,j_k} f_{m,k,t}(A, j_1, \ldots, j_k)$; see Table 3. For example, $f_{m,k,t}(A, j_1, \ldots, j_k) = (A_{j_1 j_2})^2 (A_{j_2 j_3})^2$ for the second FAM. It follows that $\mathrm{FS}(\mathcal{G}_{m,k,t}) = \sum_{j_1,j_2,j_3} (A_{j_1 j_2})^2 (A_{j_2 j_3})^2 = \mathbf{1}_n'(A \circ A)^2 \mathbf{1}_n$, so this term is equivalent to an SEA term. A similar observation is made for the other two FAMs, yielding the $m = 4$ CEEF formula displayed in Table 3.

Our results in this merging step are new, and the proofs are non-trivial with several new ideas and techniques. First, by introducing two new notions (FAM and the associated Full Sum for the FAM) in graph theory, we are able decompose C_m as a linear combination of the Full Sum of many induced FAMs. This allows us to develop sophisticated graph theory and use

Table 3. The key quantities for $m = 4$, with labeled simple graphs (LSG) included for illustration. Combining the table with Theorem 1, we get $C_4 = \mathrm{tr}(A^4) - 2\mathbf{1}_n'(A \circ A)^2\mathbf{1}_n + \mathbf{1}_n'(A \circ A \circ A \circ A)\mathbf{1}_n$.

(k,t)	Partition	Multigraph	FAM	$(d_{mkt}, h_{mkt}, a_{mkt})$	$f_{mkt}(A, j_1, \cdots, j_k)$	SEA form
$(4,1)$	$\{\{i_1\},\{i_2\},\{i_3\},\{i_4\}\}$ $j_k = i_k,\ 1 \le k \le 4$			$(1,1,1)$	$A_{j_1j_2}A_{j_2j_3}A_{j_3j_4}A_{j_4j_1}$	$\mathrm{tr}(A^4)$
$(3,1)$	$\{\{i_1\},\{i_2,i_4\},\{i_3\}\}$ $j_1 = i_1,\ j_2 = i_2,\ j_3 = i_3$			$(2,1,-2)$	$(A_{j_1j_2})^2(A_{j_2j_3})^2$	$\mathbf{1}_n'(A \circ A)^2\mathbf{1}_n$
$(2,1)$	$\{\{i_1,i_3\},\{i_2,i_4\}\}$ $j_1 = i_1,\ j_2 = i_2$			$(1,1,1)$	$(A_{j_1j_2})^4$	$\mathbf{1}_n'(A \circ A \circ A \circ A)\mathbf{1}_n$

the results to analyze C_m. In particular, the two notions play an important role in the pruning step below. Second, to figure out the linear coefficients in Theorem 1, we need to combine careful analysis with existing results on the Möbius function (at first, we do not know how to figure out these coefficients, but fortunately, AI hinted that the Möblus inversion technique may be relevant. We solve the problem by following the hint. This demonstrates how valuable AI can be for research).

B.3. The pruning process

In the merging process, C_m has been expressed as a linear combination of finitely many FS-terms, where all linear coefficients are explicitly computable. Our goal now is to convert each $\mathrm{FS}(\mathcal{G}_{m,k,t})$ equivalently to either an SEA term or an IFS term. To the best of our knowledge, this is the first work to formally introduce the notions of SEA terms and IFS terms. In particular, the concept of SEA terms based on three fundamental matrix-vector operations has not appeared in previous studies (e.g., (Harary, 1969; Chang & Fu, 2003; Movarraei & Boxwala, 2016; Perepechko & Voropaev, 2009; Jin et al., 2021)). Consequently, no existing methodology is available for this task; and the formulas previously derived for relatively small values of m rely largely on case-by-case analysis rather than on a principled procedure. In contrast, we develop a general and principled procedure applicable to arbitrary values of m.

Fix an FAM $\mathcal{G} = \mathcal{G}_{m,k,t}$. By Definition 6, there is a one-to-one correspondence between the number of nodes in this graph and the number of summation layers in the associated Full Sum. We propose a recursive pruning algorithm, in which "pruning a node" means reducing the number of summation layers by one, while preserving the value of the sum. To achieve this goal, we develop a scheme to prune the *Type-I and Type-II pendants* defined as follows.

Definition 11 (Pendants)**.** A node is called a *Type-I pendant* node if it has exactly 1 immediate neighbor (node b is called an immediate neighbor of a if there exists at least one edge between them). A node is called a *Type-II pendant* node if it has exactly 2 immediate neighbors. In either case, each immediate neighbor of a *pendant* node is referred to as a *hinge* node.

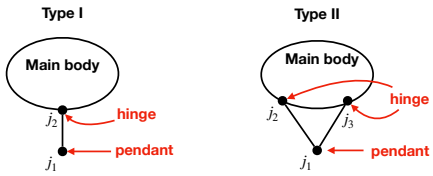


Figure 4. Type I pendant (left) and Type II pendant (right).

A FAM $\mathcal{G}$ obtained from the previous merging step is initially undirected. However, the pruning process requires $\mathcal{G}$ to be represented as a directed FAM. To this end, we arbitrarily assign an orientation to each edge in $\mathcal{G}$. Because the edge-label matrices of $\mathcal{G}$ are symmetric at this stage, the assigned orientations do not affect the corresponding full sum $\mathrm{FS}(\mathcal{G})$. As pruning proceeds, pendant nodes are successively removed and the FAM is updated accordingly. During this process, some edge-label matrices may become asymmetric, making it necessary to keep track of the edge orientations.

Definition 12 (Orientation alignment)**.** Given a directed FAM $\mathcal{G}$, an orientation alignment from node a to node b, denoted by $\mathrm{OA}_{a\to b}(\mathcal{G})$, re-orients every edge between a and b so that it points from a to b. Specifically, edges already directed from

a to b are left unchanged, while edges directed from b to a are reversed and their corresponding edge-label matrices are replaced by their transposes.

Combining Definition 12 with Definition 6, an orientation alignment does not alter the FS-term associated with a FAM.

We now describe the pendant pruning procedure. For both Type-I and Type-II pendants, the procedure consists of two steps: first performing an orientation alignment, and then pruning the pendant node and updating the node labels and edge labels of the resulting FAM. The essential requirement is that each of these operations preserves the corresponding FS-term.

Consider a Type-I pendant node j_1, with j_2 as the hinge node. Suppose there are s parallel edges between them. We conduct orientation alignment from the hinge node to the pendant node, i.e., $\mathrm{OA}_{j_2 \to j_1}(\mathcal{G})$. Let $M^{(1)}, \ldots, M^{(s)} \in \mathbb{R}^{n \times n}$ be the edge-labels after orientation alignment. Let $v \in \mathbb{R}^n$ and $u \in \mathbb{R}^n$ be the respective node labels for j_1 and j_2. When we prune the pendant node, we remove j_1 and all edges between j_1 and j_2, and update the node label of the hinge node j_2 by

$$u^{\text{new}} = u \circ \left[(M^{(1)} \circ \ldots \circ M^{(s)}) \cdot v \right], \tag{7}$$

where $\circ$ is the entry-wise product between two vectors or two matrices. Aside from the above modifications, all other nodes, edges, and their labels remain unchanged. An illustration of this updating rule is provided in the left panel of Figure 5.

Consider a Type-II pendant node j_1, with j_2 and j_3 as the hinge nodes. We perform $\mathrm{OA}_{j_1 \to j_2}(\mathcal{G})$ and $\mathrm{OA}_{j_1 \to j_3}(\mathcal{G})$, which means that all edges between the pendant and two hinges are re-oriented to point from the pendant to the hinges. Suppose there are s parallel edges between j_1 and j_2, and their labels are $Q^{(1)}, \ldots, Q^{(s)} \in \mathbb{R}^{n \times n}$ after orientation alignment, and there are t parallel edges between j_1 and j_3, with labels $R^{(1)}, \ldots, R^{(t)} \in \mathbb{R}^{n \times n}$ after orientation alignment. Let $y, x, z \in \mathbb{R}^n$ be the node labels for j_1, j_2, j_3, respectively. When we prune the pendant node, we remove j_1, all edges between j_1 and j_2, and all edges between j_1 and j_3; and we add a new directed edge from j_2 to j_3 with the following label

$$E = (Q^{(1)} \circ \ldots \circ Q^{(s)})^\top \cdot \mathrm{diag}(y) \cdot (R^{(1)} \circ \ldots \circ R^{(t)}). \tag{8}$$

Here, $\circ$ denotes entry-wise product, $\cdot$ denotes matrix multiplication, and $\mathrm{diag}(y)$ is a diagonal matrix whose diagonal entries are from y. All other nodes, edges, and their labels remain unchanged. An illustration is provided in the right panel of Figure 5.

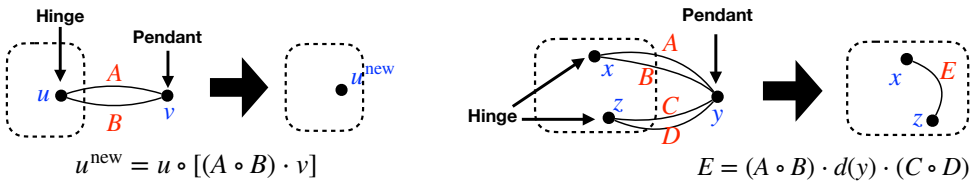


Figure 5. The pruning process and corresponding updating rule. Left: Type I. Right: Type II.

Lemma 3 (Invariance of the Full-Sum under pruning)**.** *Consider a FAM $\mathcal{G}$ with at least 1 pendant node which is either Type-I or Type-II. Suppose we pick any pendant node and prune it according to the two rules in (7)-(8), and let $\mathcal{G}^{pruned}$ be the FAM after pruning. Then, $\mathcal{G}^{pruned}$ has one fewer nodes than $\mathcal{G}$, while the associated Full-Sum remains unchanged:*

$$\mathrm{FS}(\mathcal{G}) = \mathrm{FS}(\mathcal{G}^{pruned}).$$

Lemma 3 justifies that pruning reduces the number of summation layers while keeping the associated Full-Sum unchanged. Furthermore, once the edge orientations have been aligned, the updating rules in (7)-(8) only use three operations: multiplication, entry-wise product, and diagonal extraction. These are exactly the operations allowed in SEA terms; see Definition 1.

We now propose the following algorithm.

We make a few observations. First, because $\mathcal{G}_{m,k,t}$ has k nodes, and each pruning step reduces the number of nodes by one, Algorithm 2 must terminate in at most $(k-1)$ steps. Second, by Lemma 3, $\mathrm{FS}(\mathcal{G}_{m,k,t}) = \mathrm{FS}(\mathcal{G}^{\text{final}}_{m,k,t})$. As a result, we can replace $\mathrm{FS}(\mathcal{G}_{m,k,t})$ by $\mathrm{FS}(\mathcal{G}^{\text{final}}_{m,k,t})$ in the decomposition of C_m presented in Theorem 1. Finally, we consider how to compute $\mathrm{FS}(\mathcal{G}^{\text{final}}_{m,k,t})$. When the algorithm terminates, if there is only one node left, $\mathrm{FS}(\mathcal{G}^{\text{final}}_{m,k,t}) = \mathbf{1}_n' v$, where $v \in \mathbb{R}^n$ is the label of this retaining node. By (7)-(8), v is obtained from A and $\mathbf{1}_n$ by using only three operations: multiplication,

Algorithm 2 The pruning process.

Input: an induced FAM $\mathcal{G}_{m,k,t}$ from Algorithm 1, with arbitrarily assigned edge orientations.

a. Check whether the FAM contains any pendant nodes. If not, terminate the algorithm. Otherwise, select an arbitrary pendant node and remove it together with all edges connecting it to its hinge node(s).

b. Update the node and edge labels using either (7) or (8), with the edge orientations updated as described above, depending on whether the pendant node is of Type I or Type II.

c. Repeat the above two steps, until either only one node remains or no pendant node exists.

Output: the resultant FAM $\mathcal{G}^{\text{final}}_{m,k,t}$ when the algorithm terminates.

entry-wise product, and diagonal extraction; therefore, $\mathbf{1}_n' v$ satisfies the requirement of an SEA term (Definition 1). If the algorithm terminates with $\ell > 1$ nodes, then $\text{FS}(\mathcal{G}^{\text{final}}_{m,k,t})$ is an ℓ-layer Full-Sum. The termination criteria requires that no pendant exists, which means that the number of summation layers cannot be further reduced by applying the pruning rules in (7)-(8).

The above observations are formalized in the following theorem:

Theorem 4. *Fix $m \geq 3$. Let $b_{m,k}$, $\mathcal{G}_{m,k,t}$ and $a_{m,k,t}$ be the same as in Theorem 1, and let $\mathcal{G}^{\text{final}}_{m,k,t}$ be the output by applying Algorithm 2 to $\mathcal{G}_{m,k,t}$. Then,*

$$C_m = \sum_{k=2}^{m} \sum_{t=1}^{b_{m,k}} a_{m,k,t} \cdot \text{FS}(\mathcal{G}^{\text{final}}_{m,k,t}).$$

Furthermore, there are only two possible cases for each $\mathcal{G}_{m,k,t}$:

- *Case 1: there is only one node in this FAM. Let $v \in \mathbb{R}^n$ denote the label of this node. Then, $\text{FS}(\mathcal{G}^{\text{final}}_{m,k,t}) = \mathbf{1}_n' v$, and it is an SEA term.*
- *Case 2: there is no pendant node. Then, there exist at least 4 nodes in this FAM, each with at least 3 immediate neighbors. In addition, $FS(\mathcal{G}^{\text{final}}_{m,k,t})$ is an ℓ-layer FS term with $4 \leq \ell \leq \lfloor m/2 \rfloor$.*

Comparing Theorem 4 with Definition 4, we see that the above decomposition of C_m satisfies the requirement of CEEF for every $m \geq 3$. In this sense, we have successfully solved the CEEF problem at a conceptual level. Indeed, for each specific m, one must still execute Algorithms 1-2 to derive the explicit expression of each $\text{FS}(\mathcal{G}^{\text{final}}_{m,k,t})$, as illustrated by the $m = 4$ derivation in Table 3 and by the $m = 8$ formula in Table 1. Although this process remains tedious, it no longer requires new theoretical insights; rather, it only requires a research assistant to correctly understand and follow our calgorithms step by step. For tasks of this nature, AI has important advantages over humans. In the next section, Section C, we describe how AI can be used to automatically implement the algorithm.

Example 3. We use the FAM in the third row of Table 3 to illustrate the pruning process. This FAM is visualized in the left panel of Figure 6. Its associated FS term is $\sum_{j_1,j_2,j_3} (A \circ A)_{j_1 j_2} (A \circ A)_{j_2 j_3}$. We first prune j_3 and then prune j_1. The output is a one-node graph with node label $v = (A \circ A)^2 \mathbf{1}_n$. The original FS term has been converted equivalently to $\mathbf{1}_n' v = \mathbf{1}_n' (A \circ A)^2 \mathbf{1}_n$, which is clearly an SEA term.

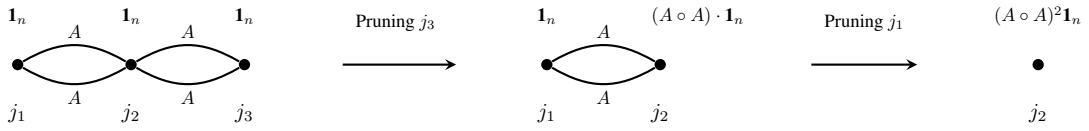


Figure 6. An illustration of the pruning process for a FAM with three nodes. In this example, it happens that edge-label matrices of the original, intermediate, and final FAMs are all symmetric. Thus, edge orientations do not matter and are omitted in the plots.

Our results are novel and nontrivial. We are the first to introduce the notions of SEA terms and IFS terms, which provide a rigorous framework for formulating the CEEF problem. Moreover, the pruning process proposed in this work has not

appeared in the existing literature. We also formulate the entire algorithm exclusively in the language of graph theory, making it naturally interpretable and teachable to AI systems.

C. Implementation with AI: prompts, results, validation, and comparison

In Section B, we have proposed an algorithm (Algorithms 1-2) for deriving the CEEF formula of C_m for each given m. However, the execution is labor-extensive. For example, the formula for C_{12} has 1900 terms, and to derive the explicit expression of each individual term is challenging for humans.

To overcome the challenge, we use AI, but (not surprisingly) a *straightforward use* of AI won't work. In particular, we have conducted experiments where we ask large language models (LLMs) to either reason in natural language or write a code to derive a formula C_m as desired. Such an approach only works for small values of m (e.g., $m = 3, 4, 5$), where correct answers exist in the literature. When $m \geq 6$, the desired formula either is not well-known or does not exist in the literature, and LLMs fail to output a satisfactory result: It often guesses a few terms with incorrect coefficients and lacks a correct reasoning approach. This suggests that AI is unable to solve the CEEF problem independently. We propose a hugAI approach which combines the strengths of human and AI.

C.1. The human-guided AI (hugAI) approach

In this approach, first, we divide the CEEF problem into many steps, and outline a clear strategy for each step; this is mostly done by us (humans), as given in Section B. Next, for each step, we either design an algorithm or formulate a clear algorithmic idea, and then ask AI to write a code for implementation. Finally, we ask AI to generate executable Python code that takes m as input and outputs the exact formula for C_m in LaTeX format. The three steps outlined here require different levels of intelligence, which we denote as *level-R (Researcher)*, *level-A (Assistant)*, and *level-C (Coder)*, respectively (see Figure 7). It is of interest to assess the level of intelligence exhibited by AI; we report the results below.

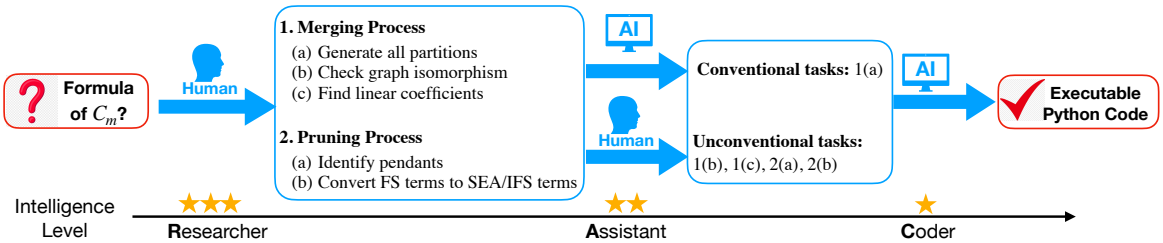


Figure 7. The pipeline of our hugAI approach for counting cycles.

For simplicity, we focus on DeepSeek-R1 (DS-R1), a recent reasoning model that demonstrates competitive performance on benchmark tasks (Guo et al., 2025). We also include additional LLMs in our study; further details are provided in Section C.4.

The two-stage benchmark prompt used in the evaluation is reproduced below. In the first stage, we give the system prompt and the Stage 1 prompt to LLMs; after verification, we give the system prompt, the model's own Stage 1 code, and the Stage 2 prompt to LLMs in a separate chat. The prompt specifies the input representation, the output code format, and the detailed instructions for the merging and pruning processes.

Benchmark prompt. **System Prompt (Shared by Both Stages)**

```
You are a mathematician implementing a combinatorial algorithm.  Your task is to derive an explicit formula for
a cycle count statistic and output it as executable Python code.  Follow the algorithm described below
precisely, and produce code output as instructed.
```

Stage 1 User Prompt: Merging Process

```
PROBLEM. For a symmetric matrix A in R{n x n} with zero diagonal (A_{ii} = 0), the order-m cycle count statistic
is:
C_m(A) = sum_{i_1,...,i_m all distinct} A_{i_1,i_2} * A_{i_2,i_3} * ...  * A_{i_m,i_1}
The brute-force cost is O(nm).  We want a Computationally Efficient Equivalent Form (CEEF): a formula
expressing C_m as a linear combination of terms, each computable in polynomial time using matrix operations.
YOUR TASK (Step 1 of 2):  Implement the MERGING PROCESS that decomposes C_m into canonical multi-graph types
with aggregated Mobius coefficients.  Your code must accept m as a parameter and work for any m >= 2.
STEP 1:  MERGING PROCESS (Partitions -> Multi-graphs -> Canonical Types)
1(a) Generate all multi-graphs.
-- Fix the index set S = {i_1,...,i_m} with a cycle graph:  edges i_1-i_2, i_2-i_3, ..., i_m-i_1.
-- A partition sigma of S merges some indices, producing a multi-graph on k nodes.  The edge weight w(a,b)
equals the total number of edges (in both directions) between blocks a and b after merging.
-- Exclude any multi-graph with self-loops (A_{ii} = 0).
1(b) Check graph isomorphism.
-- Group multi-graphs by simultaneous row-and-column permutation of the edge weight matrix.  Pick the
lexicographically smallest representative.
1(c) Find linear coefficients.
-- mu(0hat, sigma) = (-1){m-k} * prod_{i=1}{k} (|B_i| - 1)!
-- Sum these Mobius values over all partitions in the same isomorphism class.
>>> CODE OUTPUT <<<
Write a self-contained Python script defining step1(m) that returns a list of (weight_matrix, coefficient)
pairs.
Sanity checks:
-- m=2:  1 surviving partition, 1 canonical type
-- m=3:  1 surviving partition, 1 canonical type
-- m=4:  4 surviving partitions, 3 canonical types
REMINDERS:
-- In Stage 1, graph isomorphism is checked using the symmetric edge-weight matrix, because before pruning all
edge labels are inherited from the symmetric matrix A.
-- The Stage 1 weight_matrix is only a canonical pre-pruning representation.
-- Stage 2 will convert each weight_matrix into FAM edge records before pruning.
-- The cycle wraps around:  position m connects back to position 1.
-- Self-loop filtering is essential.
```

Stage 2 User Prompt: Pruning Process

```
YOUR TASK (Step 2 of 2):  Using the Step 1 output, implement the PRUNING PROCESS that converts each FS term
into a computationally efficient SEA or IFS expression.
STEP 2:  PRUNING PROCESS (FS terms -> SEA/IFS terms -> Final Formula)
An FAM is represented by:
-- A dictionary node_label[v], where each node v has a vector label.  Initially every node label is ones_n.
-- A list of oriented edge records (src, dst, matrix_label).  The edge contributes matrix_label[i_src, i_dst] to
the Full Sum.
-- Parallel edges are allowed.  Do not assume that matrix labels are symmetric after pruning.
Initialization from a Stage-1 weight_matrix:
-- For each pair u < v with weight_matrix[u][v] = w > 0, create one oriented edge record (u, v, A_hadamard_w),
where A_hadamard_w means A Hadamard-powered w times.
-- This initial orientation u -> v is arbitrary but fixed.  It is valid because the initial labels are
symmetric.
-- After pruning, newly created matrix labels may be non-symmetric, so every later access must respect the
stored orientation.
Orientation helper:
For an edge between nodes a and b with stored record (src, dst, M):
-- If src == a and dst == b, then the matrix oriented from a to b is M.
-- If src == b and dst == a, then the matrix oriented from a to b is M.T.
This rule must be used whenever a pruning formula requires a matrix oriented in a specific direction.
2(a) Identify pendants.
-- A Type-I pendant has exactly 1 distinct neighbor, ignoring edge directions.
-- A Type-II pendant has exactly 2 distinct neighbors, ignoring edge directions.
2(b-I) Type-I pruning.
Suppose p is the pendant and h is the hinge.  Let v_p be the node label of p and u_h be the node label of h.
Collect all edge labels between h and p, oriented from h to p using the orientation helper.
Let M_prod be their Hadamard product.
Update the hinge label by
node_label[h] = node_label[h] * (M_prod @ node_label[p])
where * is element-wise vector product and @ is matrix-vector multiplication.
Remove p and all edges incident to p.
2(b-II) Type-II pruning.
Suppose p is the pendant and x,y are the two hinges.  Use a deterministic order for the hinges, for example x <
y.
Collect all edge labels between p and x, oriented from p to x.  Let their Hadamard product be Q.
Collect all edge labels between p and y, oriented from p to y.  Let their Hadamard product be R.
Create a new oriented edge from x to y with label
E = Q.T @ diag(node_label[p]) @ R
Remove p and all edges incident to p.  Add the new edge record (x, y, E).
Important:
-- Do not assume E is symmetric.
-- Do not replace an existing x-y edge by E unless you correctly preserve the product of all parallel edge
labels.
```

```
-- The safest implementation is to keep E as an additional parallel oriented edge record.
Repeat until:
-- If only 1 node remains, output/evaluate an SEA term.
-- If no pendant remains and more than one node remains, output/evaluate an IFS term.
IFS rule:
For an IFS term, each oriented edge record (src, dst, M) contributes M[index[src], index[dst]].
Do not symmetrize M. Do not replace M[index[src], index[dst]] by M[index[dst], index[src]] unless the stored
orientation requires a transpose through the orientation helper.
>>> CODE OUTPUT <<<
Produce a self-contained Python script defining C_m(A, m) that works for any m >= 2.
REMINDERS:
-- Edge orientation matters after pruning.
-- Initial labels are symmetric, but pruning can create non-symmetric matrix labels.
-- Type-I pruning requires all incident edge labels oriented from hinge to pendant.
-- Type-II pruning requires Q and R oriented from pendant to each hinge before computing Q.T @ diag(v_p) @ R.
-- Keep parallel edges unless you are absolutely sure the Hadamard combination preserves orientation.
-- Carefully distinguish @, *, Hadamard product, transpose, and diag.
-- For IFS terms, use directed edge indices exactly as stored.
```

Next, we divide our project into 5 major tasks, 3 for the merging process (1(a), 1(b), 1(c)), 2 for the pruning process (2(a), 2(b)), as described in Figure 7 and the caption of Table 6. Based on the difficulty level, 1(a) is a *conventional task* for AI, and others are *unconventional tasks* for AI.

Task 1(a) is well-studied in the literature and AI is able to accomplish it without much struggle. For this task, we only provide a verbal description in the stage-1 prompt above, and DS-R1 is able to find an appropriate algorithm on its own. This demonstrates that DS-R1 has *level-A (Assistant) intelligence*.

The other 4 tasks are unconventional for AI, and for each of them, AI needs detailed guidance and carefully written prompts. For reasons of space, we only briefly discuss these tasks.

For Task 1(b), DS-R1 does not know what *isomorphism* means in the current setting, so we need to inform it with the precise definition first. After that, DS-R1 knows how to check isomorphism in a brute-force fashion, but the algorithm is very slow. To address the problem, we inform DS-R1 that for checking isomorphism, we only need to consider degree-preserved permutations. This significantly speeds up the computing. Such kind of communication between DS-R1 and us is very common during our study: many times, DS-R1 struggles to understand some concepts, but after several rounds of revision in the prompts, it is able to write a code and produce the right answer.

For Task 1(c), despite that the Möbius function (i.e., the function $\mu(\hat{0}, \sigma_{m,k,t})$ in Theorem 1) is well-known, we must teach DS-R1 how to relate it to our problem. As an example, we inform DS-R1 in the prompt above that we only need consider the pairs of partitions $\mu(\pi, \sigma)$ with $\pi = \hat{0}$ (the finest partition). With a step-by-step guidance, DS-R1 is able to produce the right results.

For Tasks 2(a)-2(b), since SEA and IFS terms are new definitions, we need to inform DS-R1 with the precise definitions first. The Stage 2 prompt reproduced above starts with the prescribed input format, followed by detailed instructions on the two types of pruning.

In summary, first, DS-R1 is able to accomplish conventional tasks, with correct answers. This demonstrates that DS-R1 has *level-A (Assistant) intelligence*. Second, DS-R1 is able to accomplish unconventional tasks, provided with a step-by-step guidance and carefully written prompts. This demonstrates that DS-R1 has *level-C (Coder) intelligence* for these unconventional tasks: it is able to accomplish the task but requires a detailed step-by-step guidance. What is exciting is that DS-R1 also has preliminary *level-R (Researcher)* intelligence. One example is that, when we initially tried to prove Theorem 1, we didn't realize it was connected to the Möbius function. DS-R1 discovered this connection independently, and we followed its hint to identify the precise linear coefficients in our formula. Another example is, DS-R1 is able to detect inconsistency in our prompt. In one of our prompts, we provided DS-R1 with both a definition and an example of the linear coefficients, but unfortunately, the coefficients in the examples are not consistent with those in the definition. DS-R1 detected such an inconsistency. Using its hint, we revised the prompt, and this time, DS-R1 fully understood our point and provided the correct answer for our question.

C.2. Results

The final outcome of our hugAI approach is a Python code, which takes an integer $m \geq 3$ as input and output the desirable formula as in Theorem 1 and Theorem 4, where each FS term is converted to either a SEA term or an IFS term. Running this code for a given m permits us to obtain the CEEF for C_m.

Here we present the complexity of the formulas of C_m by running this code for $m = 3, 4, \ldots, 12$. Table 4 presents the time complexity of computing C_m using such a formula. Note that the time complexity to compute each SEA term is dominated by the cost of calculating matrix multiplications. The time complexity of each IFS term is $O(n^\ell)$, where ℓ is the number of summation layers, i.e. the number of nodes left after the final pruning process. IFS terms only emerge when $m \geq 8$. In this table, we have grouped the IFS terms according to their calculation time complexity.

Table 4. Term count for the combinatorial formula of C_m for $m = 3, \cdots, 12$.

m	3	4	5	6	7	8	9	10	11	12
#Terms (a_m)	1	3	3	10	15	44	89	254	633	1900
#SEA Terms	1	3	3	10	15	43	86	239	581	1666
#$O(n^4)$ IFS Terms						1	3	13	46	190
#$O(n^5)$ IFS Terms								2	6	37
#$O(n^6)$ IFS Terms										7

C.3. Validation of the results

We now validate the correctness of these formulas. As established in Section B, our proof strategy has already been rigorously justified; as a result, any correct execution of this strategy is guaranteed to yield a valid formula for C_m. Therefore, the purpose of this validation is not to re-examine the strategy itself, but rather to assess whether the AI correctly understands and executes the prescribed procedure.

Manual validation For $m \leq 8$, the formulas contain relatively few terms (see Table 4) and can be derived manually. We therefore use these hand-derived expressions as the ground truth and find that the formulas produced by our method are correct for all $m \leq 8$. For larger values of m, there are few known results in the literature, except when A is binary (e.g., (Perepechko & Voropaev, 2009)). In this setting, the formulas are much simpler, as many terms are identical and can be combined (for example, $A \circ \cdots \circ A = A$). We apply our method and simplify the resulting expressions accordingly, and find that they agree with those reported in (Perepechko & Voropaev, 2009) for $m \leq 13$.

Numerical validation Manual validation becomes intractable for large m or the general (non-binary) case. To address this limitation, we introduce a numerical validation pipeline that enables verification of the formulas for both large m and the general case. In this approach, we generate many $n \times n$ matrices A, compute the true C_m, and compare them with the values based on our formulas.

One question is how to compute the true value of C_m. We may search $\binom{n}{m}$ indices in a brute-force way, but a better solution is to use dynamic programming (DP); see the paragraph below for details. The DP method is guaranteed to output the true C_m, but its complexity is $O(n^2 2^n)$ for a fixed m. As a consequence, the DP-benchmark is feasible to compute only for small n. We thereby restrict our experiments to $n \leq 16$.

Another question is the following: if we observe matching C_m values across a large number of small-size matrices A, how can we conclude that this provides evidence for the correctness of the formulas? An important observation is that the space of all possible formulas produced by our method is *finite* (see Definitions 1-2 and note that the coefficients in our formulas can only take integer values), whereas the data matrices A we generate take *continuous* values. As a result, the probability that an incorrect formula generates correctly matching values is exactly zero, as formally established in the following lemma:

Lemma 5. *Fix $m \geq 3$. Let $C_m^{\mathrm{DS}}(A)$ be the CEEF of $C_m(A)$ produced by the Python code output by DS-R1. If $C_m^{\mathrm{DS}}(A)$ is not algebraically equivalent to $C_m(A)$, then for any $n \geq 2$ and random matrix $A \in \mathbb{R}^{n\times n}$ whose entries have a jointly continuous distribution, $\mathbb{P}(C_m^{\mathrm{DS}}(A) = C_m(A)) = 0$.*

In our experiments, we sample A from the Gaussian Orthogonal Ensemble (GOE), where $A_{ij} = A_{ji} \sim \mathcal{N}(0, 1/n)$ for $i < j$ and $A_{ii} \sim \mathcal{N}(0, 2/n)$; and for each setting, we use $t = 50$ samples of A with a fixed random seed. An important issue is the numerical precision required to declare two values as "matching." For large m, individual terms in the formula may be small in magnitude, so that discrepancies can become effectively indistinguishable at finite precision. Fortunately, when A is generated from the GOE, we can characterize the asymptotic order of each term in the formula, allowing us to calibrate an appropriate tolerance used in validation. Figure 8 presents the histograms of discrepancies between values from our formulas and the DP-benchmark, for a set of (n, m). We observe that the values are well-matched up to tolerable numerical precision, confirming the correctness of our formulas.

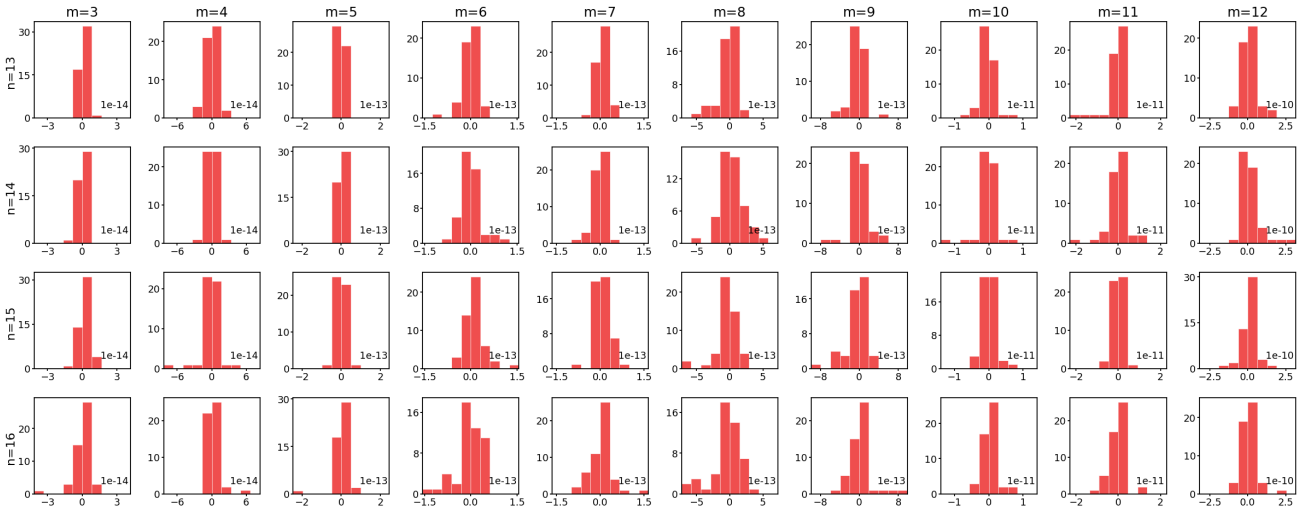


Figure 8. Numerical validation (the histogram of discrepancies between the values by our formulas and the DP-benchmarks, where $n = 13, 14, 15, 16$ and $m = 3, \ldots, 12$).

Table 5. Average runtime (in seconds) of each C_m for different values of n. Since this experiment is to validate our formula using the computationally expensive DP algorithm, we only consider small values of n. Experiments with much larger values of n are deferred to Section D.

m	$n = 13$		$n = 14$		$n = 15$		$n = 16$	
	Our Approach	DP	Our Approach	DP	Our Approach	DP	Our Approach	DP
3	2.84×10^{-4}	2.30×10^{-2}	2.93×10^{-4}	3.89×10^{-2}	1.61×10^{-4}	8.14×10^{-2}	1.87×10^{-4}	1.70×10^{-1}
4	1.41×10^{-4}	3.97×10^{-2}	2.51×10^{-4}	5.77×10^{-2}	1.04×10^{-4}	1.07×10^{-1}	1.18×10^{-4}	2.04×10^{-1}
5	1.05×10^{-4}	1.03×10^{-1}	2.30×10^{-4}	1.33×10^{-1}	8.08×10^{-5}	2.21×10^{-1}	8.75×10^{-5}	3.75×10^{-1}
6	3.01×10^{-4}	2.50×10^{-1}	3.68×10^{-4}	3.32×10^{-1}	2.23×10^{-4}	5.56×10^{-1}	2.34×10^{-4}	9.25×10^{-1}
7	4.43×10^{-4}	4.86×10^{-1}	4.76×10^{-4}	6.95×10^{-1}	3.38×10^{-4}	1.24×10^{0}	3.44×10^{-4}	2.18×10^{0}
8	4.13×10^{-3}	7.48×10^{-1}	3.75×10^{-3}	1.17×10^{0}	3.98×10^{-3}	2.27×10^{0}	3.96×10^{-3}	4.33×10^{0}
9	1.14×10^{-2}	9.47×10^{-1}	9.40×10^{-3}	1.62×10^{0}	9.18×10^{-3}	3.42×10^{0}	9.50×10^{-3}	7.09×10^{0}
10	4.49×10^{-2}	1.06×10^{0}	4.04×10^{-2}	1.92×10^{0}	3.77×10^{-2}	4.35×10^{0}	4.94×10^{-2}	9.73×10^{0}
11	1.26×10^{-1}	1.08×10^{0}	1.21×10^{-1}	2.06×10^{0}	1.22×10^{-1}	4.88×10^{0}	1.23×10^{-1}	1.15×10^{1}
12	5.13×10^{-1}	1.11×10^{0}	4.95×10^{-1}	2.10×10^{0}	5.01×10^{-1}	5.10×10^{0}	5.02×10^{-1}	1.26×10^{1}

This small-scale experiment also supports the computational value of having a CEEF formula. In Table 5, we report the runtime: For small n, our method is already substantially faster than DP. Note that for large n, DP is infeasible to implement, while our method is still implementable.

C.4. Comparison of representative LLMs

While we primarily focus on DS-R1 in the main experiments, it is also of interest to examine the performance of other LLMs under the same prompts. We test 11 models spanning 7 providers. All models are queried with web search disabled, and reasoning or thinking mode is enabled whenever it is supported by the model interface.

The AI implementation of our method contains two stages, Merging and Pruning, with five tasks (1(a)-1(c), and 2(a)-2(b); see the caption of Table 6 for details). To facilitate the comparison across LLMs, we decompose the prompting-and-response pipeline into two components, rather than requiring a single monolithic response:

- Stage 1 (Merging). An LLM receives Stage-1-User-Prompt (see Section C.1) and is asked to produce a Python function `step1(m)` that enumerates all partitions of the m-cycle, filters self-loops, groups the resulting multi-graphs by isomorphism, and computes the aggregated Möbius coefficient for each canonical type.

- Stage 2 (Pruning). An LLM receives Stage 2 User Prompt (see Section C.1) together with its own verified Stage 1 code as context. It is asked to produce a function `C_m(A, m)` that applies pendant pruning to each canonical type and constructs the corresponding SEA or IFS terms.

Table 6. Evaluation of 11 LLMs for the two-stage implementation task. Stage 1 consists of four checks: **Syntax(1)** (code runs), **1a** (valid partition enumeration), **1b** (canonical graph types), **1c** (Möbius coefficients). Stage 2 consists of two checks: **Syntax(2)** (code runs) and **2a/2b** (the generated formula matches the numerical benchmark). A ✓ indicates success throughout the reported validation range; ✗ indicates at least one failure.

Model	**Stage 1: Merging**				**Stage 2: Pruning**		**Summary**
	Syntax(1)	1a	1b	1c	Syntax(2)	2a/2b	
Claude Opus 4.8	✓	✓	✓	✓	✓	✓	Successful
Claude Sonnet 4.6	✓	✓	✓	✓	✓	✓	
DeepSeek V4 Pro	✓	✓	✓	✓	✓	✓	
GPT-5.5	✓	✓	✓	✓	✓	✓	
Gemini 3.1 Pro	✓	✓	✓	✓	✓	✓	
Gemini 3.5 Flash	✓	✓	✓	✓	✓	✓	
GLM 5.1	✓	✓	✓	✓	✓	✓	
Kimi K2.6	✓	✓	✓	✓	✓	✓	
MiniMax-M3	✓	✓	✓	✓	✓	✓	
MiniMax-M2.7	✓	✓	✓	✓	✗	✗	Stage 1 only
DeepSeek V3.2	✗	✗	✗	✗	✗	✗	Unsuccessful

This two-component design avoids a potential output-length truncation issue: a single prompt frequently yields an output that exceeds the 32 k-token limit before completing all tasks. It also enables a better diagnosis—when the formula is incorrect, we can identify whether it is from Stage 1 or Stage 2.

The Stage 1 performance is verified by comparing the canonical graph types and their coefficients with the ground truth. For Stage 2, we validate the final formulas by evaluating `C_m(A, m)` on fixed zero-diagonal Wigner matrices and comparing the outputs with DP-benchmark. A formula is considered "correct" if the discrepancy with the benchmark is below 10^{-8}. The results are presented in Table 6.

Our findings are as follows. First, ten of the eleven models successfully complete the tasks in Stage 1. For instance, the successful Stage 1 models correctly report 41 surviving partitions and 10 canonical types at $m = 6$, and 715 surviving partitions and 44 canonical types at $m = 8$. This suggests that the combinatorial enumeration task—while non-trivial—is within the reliable capability of most frontier models when given a precise algorithmic description. Second, Stage 2 further separates the models. Nine models that pass Stage 1 also produce numerically correct formulas in the reported validation range. MiniMax-M2.7 passes the Stage 1 structural checks but its Stage 2 code cannot be validated because the generated program fails to import. DeepSeek V3.2 fails already in Stage 1 and its Stage 2 output also fails the numerical checks. This pattern illustrates the diagnostic value of separating Merging from Pruning: a Stage 2 failure can be interpreted differently depending on whether the canonical Stage 1 input has first been verified.

D. Applications

In this section, we present several applications of the cycle count statistic C_m, and illustrate the advantage of using a large value of m in these applications. Since these applications require large values of m, solving the CEEF problem is particularly important.

D.1. Testing Results

We next turn to the testing problem considered in the paper (Jin et al., 2025b). As in that paper, the goal is to distinguish a pure Wigner matrix from a rank-two spiked alternative. Here we compare the Cycle Count (CC) statistics with the Empirical Eigenvalues (EE) method. For the two reported data sets, the matrix dimensions are $n = 500$ and $n = 1000$, the spike strengths are $\lambda_1 = \lambda_2 = 1$, and we use the first $t = 100$ samples in each file. The cycle-based test uses the first m cycle statistics, and we report the ideal testing error as m varies from 1 to 12; for the precise form of the aggregated statistic, we refer to the testing paper.

The numerical trend is fully consistent with the conclusions of the testing paper: CC is substantially more effective than EE in the weak-signal regime. For $n = 1000$, the ideal error decreases from about 0.410 at $m = 1$ to about 0.130 at $m = 12$, while EE remains at about 0.390. For $n = 500$, the best CC error is about 0.250, attained at $m = 12$, whereas the EE error is about 0.430. The main qualitative message is that higher-order cycle information materially improves testing performance:

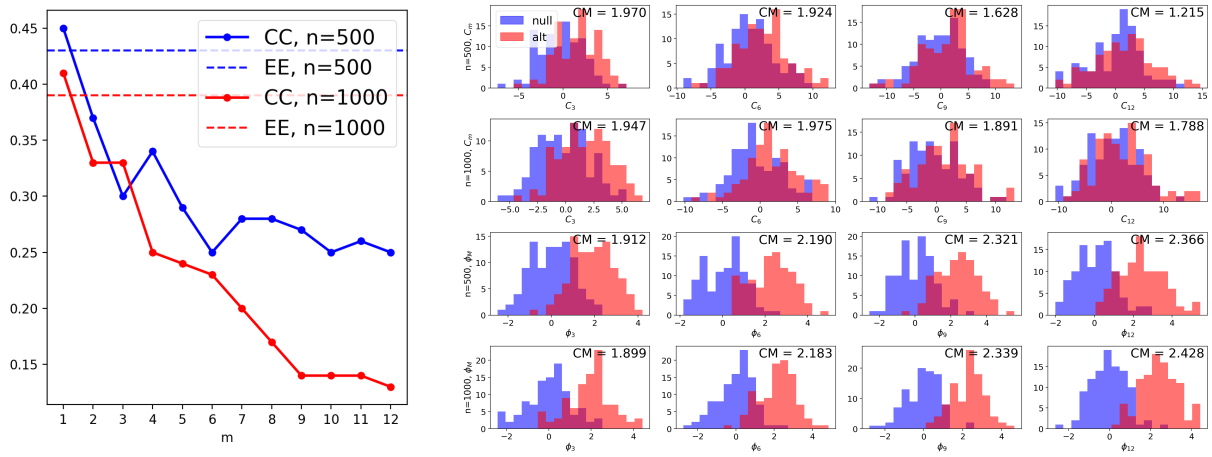


Figure 9. Testing results for $n = 500$ and $n = 1000$ with $t = 100$. Left: ideal error versus m. Right: histograms of C_m and ϕ_m for $m = 3, 6, 9, 12$.

as more cycle statistics are aggregated, the CC error decreases steadily and its advantage over EE becomes increasingly pronounced. This trend is especially clear at the larger dimension, where the gain from moving to higher-order statistics is the most substantial. Figure 9 summarizes the testing evidence in two complementary ways. The left panel shows the ideal testing error as a function of the truncation order m, while the right panel displays the histograms of both the raw cycle quantities C_m and the aggregated statistics ϕ_m for $m = 3, 6, 9, 12$. In both $n = 500$ and $n = 1000$, the separation between the null and alternative distributions becomes clearer after aggregation into ϕ_m, and the separation becomes more visible for larger values of m, which is consistent with the sharper testing performance observed in the error curves.

D.2. Estimation Results

We finally consider the estimation problem studied in the paper (Jin et al., 2026). In this experiment the target spikes are -1 and 1.5, the ambient dimension is $n = 1000$, and we use the first $t = 100$ samples from the simulation output. For each sample, the Newton-Prony estimator is built from the first m cycle statistics, with $m = 2, \ldots, 12$, and is used to recover the two signal eigenvalues. The performance metric is the normalized mean squared error of the estimated spike vector. The comparison method is the direct EE estimator formed from the two extreme observed eigenvalues; for the detailed construction of the estimator, we refer to the estimation paper.

The results again match the qualitative message of the estimation paper. The cycle-based estimator becomes clearly superior to EE once more cycle information is incorporated. Quantitatively, the EE estimator has normalized MSE about 0.443, whereas the Newton-Prony estimator decreases from about 0.532 at $m = 2$ to about 0.075 at $m = 12$. In particular, the cycle estimator already outperforms EE from $m = 3$ onward, and its error continues to improve substantially as higher-order cycle statistics are included. The main qualitative conclusion is therefore the same as in testing: using more high-order cycle information leads to visibly better recovery. The distribution plots provide complementary evidence: for $m = 2, 4, 6, 8, 10, 12$, the estimated spikes progressively concentrate near the targets -1 and 1.5, with the separation becoming cleaner at larger truncation levels. Altogether, these results show that the Newton-Prony estimator benefits substantially from higher-order cycle statistics and can clearly outperform the direct eigenvalue method in this setting.

D.3. Higher-Order Pairwise IBM Simulation

To investigate higher-order IBM statistics in a setting aligned with the pairwise-comparison framework of (Jin et al., 2025a), we follow the undirected DCMM simulation design used there as closely as possible. For fixed (n, K), we first generate $\theta_{iu} \sim \mathrm{Unif}(2, 3)$ and set $\theta_i = \beta_n \theta_{iu} / \|\theta_u\|_2$, so that $\|\theta\|_2 = \beta_n$. Next we draw $\pi_i \sim \mathrm{Dir}(1.6, 0.4)$ and $\widetilde{\pi}_i \sim \mathrm{Dir}(1, 1)$, and define the community interaction matrix by $P = (1 - b_n) I_2 + b_n \mathbf{1}_2 \mathbf{1}_2^\top$. With $\Theta = \mathrm{diag}(\theta_1, \ldots, \theta_n)$, $\Pi = [\pi_1, \ldots, \pi_n]^\top$, and $\widetilde{\Pi} = [\widetilde{\pi}_1, \ldots, \widetilde{\pi}_n]^\top$, we form $\Omega = \Theta \Pi P \Pi^\top \Theta$ and $\widetilde{\Omega} = \Theta \widetilde{\Pi} P \widetilde{\Pi}^\top \Theta$. The parameter b_n is selected numerically so that the SNR proxy in equation (2.16) of the paper matches a prescribed target value. In the experiments reported below, we take $n = 500$, $K = 2$, $\beta_n = 6$, and $t = 200$ pair instances, and compare two signal levels, namely target SNR $= 2$ and target SNR $= 1.5$. We then generate independent adjacency matrices from Ω and $\widetilde{\Omega}$ and consider the signed difference matrix

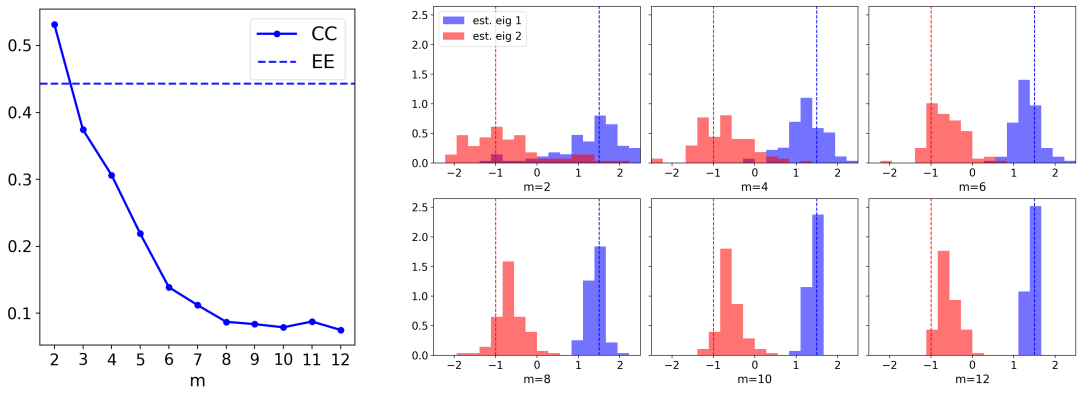


Figure 10. Estimation results for spikes -1 and 1.5 with $t = 100$. Left: normalized eigenvalue MSE versus m. Right: estimated-eigenvalue distributions for $m = 2, 4, 6, 8, 10, 12$.

$B = A - \widetilde{A}$. Following the simulation protocol of the paper, null pairs are constructed as (A_t, A_{t+t}), where both graphs are sampled from Ω, while alternative pairs are constructed as $(A_t, \widetilde{A}_t)$, where the second graph is sampled from $\widetilde{\Omega}$. Since the model is undirected, B is symmetric, and the order-r IBM statistic reduces to the signed cycle count of length $2r$:

$$U_r = C_{2r}(B), \qquad r = 2, \ldots, m.$$

The corresponding standardized statistic is

$$\Phi_r = \frac{U_r}{\left(2^{2r+1} r \left[C_{2r}(A) + C_{2r}(\widetilde{A})\right]\right)^{1/2}},$$

which matches the normalization constants proved in the paper for $r = 2$ and $r = 3$ and extends the same variance pattern to higher orders. We evaluate the orders $r = 2, 3, 4, 5, 6$.

Figure 11 compares the resulting behavior at these two signal levels. In both panels, the upper portion displays the empirical distributions of the standardized statistics $\Phi_2, \ldots, \Phi_6$ under the null and alternative, together with the reference $N(0, 1)$ law, while the lower portion reports the corresponding ideal classification error and the empirical means under the two hypotheses. The same qualitative pattern appears in both settings: after standardization, the null distributions remain concentrated around zero, whereas the alternative distributions move farther and farther to the right as r increases. Correspondingly, the ideal error decreases with the cycle order in both SNR regimes, so the highest reported orders produce the strongest separation. When the target SNR is reduced from 2 to 1.5, the separation becomes weaker and the ideal errors are uniformly larger, but the same monotone improvement with increasing order remains clearly visible. The main takeaway is therefore that higher-order cycle statistics are consistently more informative, even after the overall signal level is reduced.

E. Proofs

In this section, we provide complete proofs for all lemmas and theorems presented in the main manuscript.

E.1. Properties of the induced multi-graphs

Recall that $S = \{i_1, i_2, \ldots, i_m\}$ and $\mathcal{G}$ is the undirected simple graph whose node set is S and there are edges between $i_1 \& i_2$, $i_2 \& i_3$, …, $i_m \& i_1$, but nowhere else. We use $\sigma = \{S_1, S_2, \ldots, S_k\}$ to denote a general size-k partition on S, where $S_1, S_2, \ldots, S_k$ are non-overlapping subsets whose union is equal to S, and k is the size of this partition, for which we often write $|\sigma| = k$. As described in Section B.2, each partition σ induces a multi-graph $\mathcal{G}_\sigma$: It modifies $\mathcal{G}$ by merging all nodes in each S_j to one, while keeping the edges unchanged (note that $\mathcal{G}_\sigma$ has $|\sigma|$ nodes in this case).

Definition 13. We call $\mathcal{G}_\sigma$ non-trivial if it does not have self-loops, and we call a partition σ non-trivial if $\mathcal{G}_\sigma$ is non-trivial.

The next lemma gives some useful properties for these induced $\mathcal{G}_\sigma$:

Lemma 6. *A non-trivial induced multi-graph $\mathcal{G}_\sigma$ has the following properties: (a) It is connected. (b) Its size satisfies that $k \geq 2$. (c) The degree of each node is even.*

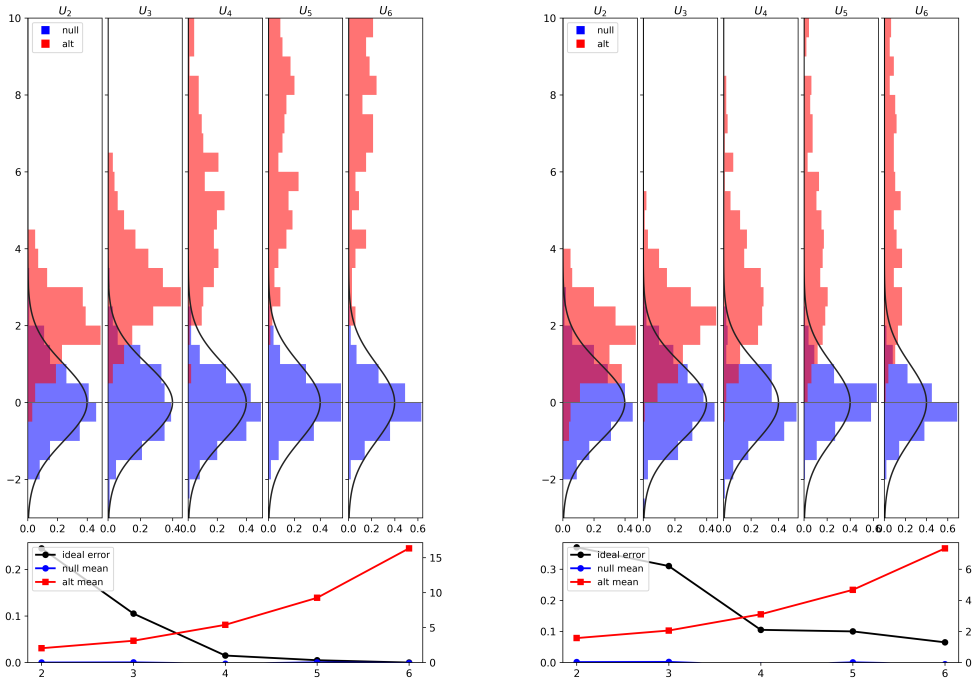


Figure 11. Standardized higher-order IBM statistics for the undirected DCMM experiment with $n = 500$, $K = 2$, $\beta_n = 6$, Dirichlet memberships $\mathrm{Dir}(1.6, 0.4)$ versus $\mathrm{Dir}(1, 1)$, and $t = 200$. Left: target SNR $= 2$. Right: target SNR $= 1.5$.

Proof of Lemma 6: For (a), we recall that $\mathcal{G}_\sigma$ is obtained from merging nodes in $\mathcal{G}$. If a graph is connected before the merging process, it will also be connected afterwards. Since the original graph $\mathcal{G}$ is connected, each $\mathcal{G}_\sigma$ must also be connected. For (b), we note that a 1-node multi-graph can only have self-loops, which contradict the definition of a non-trivial multi-graph.

To prove (c), we study how merging two nodes a and b in a multi-graph affects node degrees. Suppose the degree of each node is even before merging. For all nodes other than a and b, their degrees are unchanged during merging. For a and b, let d_a and d_b be their respective degrees before merging, and let w_{ab} be the number of edges between a and b. Here, both d_a and d_b are even, and w_{ab} can be either even or odd. After these two nodes are merged, the degree of the new node $a\&b$ becomes

$$d^* = (d_a - w_{ab}) + (d_b - w_{ab}), \qquad \text{which must be an even number.}$$

So far, we have shown the following: If each node has an even degree before merging, the same is true after merging. Therefore, the claim in (c) follows immediately by noting that each node has an even degree in the original graph $\mathcal{G}$. $\square$

E.2. Proof of the preliminary decomposition

Recall that $C_m = \sum_{i_1,\ldots,i_m\,(\text{distinct})} A_{i_1 i_2} A_{i_2 i_3} \ldots A_{i_m i_1}$. Without the requirement of having distinct indices, we can easily deduce $\sum_{i_1,\ldots,i_m} A_{i_1 i_2} A_{i_2 i_3} \ldots A_{i_m i_1} = \mathrm{tr}(A^m)$. As a result,

$$C_m = \mathrm{tr}(A^m) - R_m, \qquad \text{with} \quad R_m := \sum_{\substack{1 \le i_1, i_2, \ldots, i_m \le n \\ \text{(not distinct)}}} A_{i_1 i_2} A_{i_2 i_3} \ldots A_{i_m i_1}. \tag{9}$$

We study R_m. When $i_1, \ldots, i_m$ are non-distinct, there can be $k \le m-1$ distinct indices. When $k = 1$, since the diagonal entries of A are zero, we have $A_{i_1 i_2} A_{i_2 i_3} \ldots A_{i_m i_1} = 0$. It suffices to consider $2 \le k \le m-1$, and we can write

$$R_m = \sum_{k=2}^{m-1} \sum_{\substack{1 \le i_1, i_2, \ldots, i_m \le n \\ |\{i_1, i_2, \ldots, i_m\}| = k}} A_{i_1 i_2} A_{i_2 i_3} \ldots A_{i_m i_1}.$$

When $|\{i_1, i_2, \ldots, i_m\}| = k$, these m indices are divided into k non-overlapping sets and correspond to a size-k partition. To this end, we define a mapping $\mathcal{M}$ that maps each $(i_1, i_2, \ldots, i_m)$ to a partition σ. Let $\Pi(S)$ denote the collection of all

partitions with $2 \le |\sigma| \le m-1$. We can further write

$$R_m = \sum_{k=2}^{m-1} \sum_{\sigma \in \Pi(S):|\sigma|=k} \sum_{\substack{1 \le i_1, i_2, \ldots, i_m \le n \\ \mathcal{M}(i_1, i_2, \ldots, i_m) = \sigma}} A_{i_1 i_2} A_{i_2 i_3} \ldots A_{i_m i_1}. \tag{10}$$

For each σ with $|\sigma| = k$, let $j_1, j_2, \ldots, j_k$ denote the nodes in the corresponding multi-graph $\mathcal{G}_\sigma$. When $\mathcal{M}(i_1, i_2, \ldots, i_m) = \sigma$, we map each of $i_1, i_2, \ldots, i_m$ to one of $j_1, j_2, \ldots, j_k$. For example, when $m = 4$ and $\sigma = \{\{i_1, i_3\}, \{i_2\}, \{i_4\}\}$, the original indices i_1, i_2, i_3, i_4 are mapped to the new indices j_1, j_2, j_1, j_3, respectively. Consequently,

$$A_{i_1 i_2} A_{i_2 i_3} A_{i_3 i_4} A_{i_4 i_1} = A_{j_1 j_2} A_{j_2 j_1} A_{j_1 j_3} A_{j_3 j_1} = A_{j_1 j_2}^2 A_{j_1 j_3}^2.$$

More generally, let $w(a, b)$ denote the edge weight between nodes j_a and j_b in $\mathcal{G}_\sigma$. We observe that each $A_{j_a j_b}$ is repeated $w(a, b)$ times in $A_{i_1 i_2} A_{i_2 i_3} \ldots A_{i_m i_1}$, which implies that

$$A_{i_1 i_2} A_{i_2 i_3} \ldots A_{i_m i_1} = \prod_{1 \le a < b \le k} A_{j_a j_b}^{w(a,b)}, \qquad \text{when } \mathcal{M}(i_1, i_2, \ldots, i_m) = \sigma.$$

Furthermore, when we sum over $1 \le i_1, \ldots, i_m \le n$ under the constraint that $\mathcal{M}(i_1, \ldots, i_m) = \sigma$, it is equivalent to summing over $1 \le j_1, \ldots, j_k \le n$ under the constraint that $j_1, \ldots, j_k$ are distinct. Combining the above arguments gives

$$\sum_{\substack{1 \le i_1, i_2, \ldots, i_m \le n \\ \mathcal{M}(i_1, i_2, \ldots, i_m) = \sigma}} A_{i_1 i_2} A_{i_2 i_3} \ldots A_{i_m i_1} = \sum_{\substack{1 \le j_1, j_2, \ldots, j_k \le n \\ \text{(distinct)}}} \Big(\prod_{1 \le a < b \le k} A_{j_a j_b}^{w(a,b)} \Big). \tag{11}$$

We plug (11) into (10) to obtain:

$$R_m = \sum_{k=2}^{m-1} \sum_{\sigma \in \Pi(S):|\sigma|=k} \sum_{\substack{1 \le j_1, j_2, \ldots, j_k \le n \\ \text{(distinct)}}} \Big(\prod_{1 \le a < b \le k} A_{j_a j_b}^{w(a,b)} \Big). \tag{12}$$

Write $f_\sigma(A, j_1, j_2, \ldots, j_k) := \prod_{1 \le a < b \le k} A_{j_a j_b}^{w(a,b)}$ for brevity. We can express (12) equivalently as

$$R_m = \sum_{k=2}^{m-1} \sum_{\sigma \in \Pi(S):|\sigma|=k} R_\sigma, \qquad \text{with} \quad R_\sigma := \sum_{\substack{1 \le j_1, j_2, \ldots, j_k \le n \\ \text{(distinct)}}} f_\sigma(A, j_1, j_2, \ldots, j_k).$$

We group all multi-graphs $\mathcal{G}_\sigma$ first by size and then by isomorphism. Let $b_{m,k}$ be the total number of isomorphic classes for size-k multi-graphs, and let $\mathcal{G}_{m,k,t}$ denote the t-th isomorphic class, for $1 \le t \le b_{m,k}$. It follows that

$$R_m = \sum_{k=2}^{m-1} \sum_{t=1}^{b_{m,k}} \sum_{\sigma:\, \mathcal{G}_\sigma \in \mathcal{G}_{m,k,t}} R_\sigma. \tag{13}$$

Consider two partitions σ and $\tilde{\sigma}$ such that $\mathcal{G}_\sigma$ and $\mathcal{G}_{\tilde{\sigma}}$ are isomorphic multi-graphs. The definition of isomorphism implies that there exists a permutation $\mathcal{P}$ on $\{j_1, j_2, \ldots, j_k\}$ such that

$$f_{\tilde{\sigma}}(A, j_1, j_2, \ldots, j_k) = f_\sigma\big(A, \mathcal{P}(j_1), \mathcal{P}(j_2), \ldots, \mathcal{P}(j_k)\big).$$

When we sum over $1 \le j_1, \ldots, j_k \le n$ and require them to be distinct, the above permutation does not matter. As a result,

$$R_\sigma = R_{\tilde{\sigma}}, \qquad \text{whenever } \mathcal{G}_\sigma \text{ and } \mathcal{G}_{\tilde{\sigma}} \text{ are isomorphic.} \tag{14}$$

To this end, we pick the canonical partition $\sigma_{m,k,t} \in \mathcal{G}_{m,k,t}$ and define $f_{m,k,t}(A, j_1, j_2, \ldots, j_k) := f_{\sigma_{m,k,t}}(A, j_1, j_2, \ldots, j_k)$. Combining (14) with (13), we obtain:

$$\begin{aligned} R_m &= \sum_{k=2}^{m-1} \sum_{t=1}^{b_{m,k}} d_{m,k,t} \cdot R_{\sigma_{k,m,t}}, \qquad \text{where} \quad d_{m,k,t} = |\mathcal{G}_{m,k,t}|, \\ &= \sum_{k=2}^{m-1} \sum_{t=1}^{b_{m,k}} d_{m,k,t} \sum_{\substack{1 \le j_1, j_2, \ldots, j_k \le n \\ \text{(distinct)}}} f_{m,k,t}(A, j_1, j_2, \ldots, j_k). \end{aligned} \tag{15}$$

Plugging (15) into (9), we obtain:

$$C_m = \mathrm{tr}(A^m) - \sum_{k=2}^{m-1} \sum_{t=1}^{b_{m,k}} d_{m,k,t} \sum_{\substack{1\le j_1,j_2,\ldots,j_k\le n\\ \text{(distinct)}}} f_{m,k,t}(A, j_1, j_2, \ldots, j_k), \tag{16}$$

which completes the proof. □

E.3. Proof of Theorem 1

Write $S = \{i_1, i_2, \ldots, i_m\}$ and let $\Pi(S)$ be the collection of all partitions over S. For any $\sigma \in \Pi(S)$, let $\{j_1, j_2, \ldots, j_k\}$ be the nodes in the induced multi-graph $\mathcal{G}_\sigma$, and let $w(a, b)$ be the edge weight between any two nodes j_a and j_b. Let $f_\sigma(A, j_1, j_2, \ldots, j_k) := \prod_{1\le a<b\le k} A_{j_a j_b}^{w(a,b)}$. We introduce two quantities:

$$R_\sigma^{\mathrm{DS}} := \sum_{\substack{1\le j_1,j_2,\ldots,j_k\le n\\ \text{(distinct)}}} f_\sigma(A, j_1, j_2, \ldots, j_k), \qquad R_\sigma^{\mathrm{FS}} := \sum_{1\le j_1,j_2,\ldots,j_k\le n} f_\sigma(A, j_1, j_2, \ldots, j_k). \tag{17}$$

We call them the (merged) σ-DS term and σ-FS term, respectively.

Given partitions σ and τ, we use $\tau \preceq \sigma$ to denote that τ is obtained by splitting blocks in σ, and we use $\tau \prec \sigma$ to denote that $\tau \preceq \sigma$ but $\tau \neq \sigma$. Our proof is based on the following key lemma:

Lemma 7. *Let $\mathcal{D}(\sigma)$ be an arbitrary mapping that maps each $\sigma \in \Pi(S)$ to a real value. Then,*

$$\sum_{\sigma\in\Pi(S)} \mathcal{D}(\sigma) R_\sigma^{FS} = \sum_{\sigma\in\Pi(S)} \mathcal{F}_1(\sigma) R_\sigma^{DS}, \qquad \textit{with} \quad \mathcal{F}_1(\sigma) := \sum_{\tau:\tau\preceq\sigma} \mathcal{D}(\tau),$$

and

$$\sum_{\sigma\in\Pi(S)} \mathcal{D}(\sigma) R_\sigma^{DS} = \sum_{\sigma\in\Pi(S)} \mathcal{F}_2(\sigma) R_\sigma^{FS}, \quad \textit{with} \quad \mathcal{F}_2(\sigma) := \begin{cases} \mathcal{D}(\sigma), & \textit{if } \sigma = \hat{0}, \\ \mathcal{D}(\sigma) - \sum_{\tau:\tau\prec\sigma} \mathcal{F}_2(\tau), & \textit{if } \sigma \neq \hat{0}. \end{cases}$$

Proof of Lemma 7: Before proving this lemma, we argue that $\mathcal{F}_2(\cdot)$ is well-defined. For each $\sigma \in \Pi(S)$, let $\mathrm{Child}(\sigma) = \{\tau : \tau \prec \sigma\}$. We use Π^* to denote the collection of σ for which $\mathcal{F}_2(\sigma)$ has been defined. This set initially only contains one element, which is $\hat{0}$. Next, for any σ such that $\mathrm{Child}(\sigma) \subset \Pi^*$, we can use the formula to obtain $\mathcal{F}_2(\sigma)$. We then add these σ into Π^*. After Π^* is updated, we search again those σ such that $\mathrm{Child}(\sigma) \subset \Pi^*$, use the formula to obtain $\mathcal{F}_2(\sigma)$, and add them to Π^*. This procedure is repeated until every σ is in Π^*. Therefore, $\mathcal{F}_2(\cdot)$ is well-defined.

Consider the first claim. Let $\tau_0 = \hat{0}$. We introduce a mapping $\mathcal{M}$ that maps each $(i_1, i_2, \ldots, i_m) \in \{1, 2, \ldots, n\}^m$ to a partition σ, depending on which of these m indices are equal to each other (this mapping has also been used in (10), with a more careful explanation there). We can easily see that

$$R_{\tau_0}^{\mathrm{FS}} = \sum_{1\le i_1,\ldots,i_m\le n} f_{\tau_0}(A, i_1, \ldots, i_m) = \sum_{\sigma\in\Pi(S)} \sum_{\substack{1\le i_1,\ldots,i_m\le n,\\ \mathcal{M}(i_1,\ldots,i_m)=\sigma}} f_{\tau_0}(A, i_1, \ldots, i_m).$$

Additionally, in (11), we have proved the following result: For each σ, letting $j_1, j_2, \ldots, j_k$ denote the nodes in $\mathcal{G}_\sigma$, it holds that

$$\sum_{\substack{1\le i_1,\ldots,i_m\le n,\\ \mathcal{M}(i_1,\ldots,i_m)=\sigma}} f_{\tau_0}(A, i_1, \ldots, i_m) = \sum_{\substack{1\le j_1,j_2,\ldots,j_k\le n\\ \text{(distinct)}}} f_\sigma(A, j_1, j_2, \ldots, j_k) = R_\sigma^{\mathrm{DS}}.$$

Combining the above two equations and noticing that $\tau_0 \preceq \sigma$ for all $\sigma \in \Pi(S)$, we obtain:

$$R_{\tau_0}^{\mathrm{FS}} = \sum_{\sigma\in\Pi(S)} R_\sigma^{\mathrm{DS}} = \sum_{\sigma:\tau_0\preceq\sigma} R_\sigma^{\mathrm{DS}}, \qquad \text{for } \tau_0 = \hat{0}.$$

For $\tau \neq \tau_0$, we can mimic the above proof and consider all partitions over the nodes in $\mathcal{G}_\tau$. We can similarly show that

$$R_\tau^{\mathrm{FS}} = \sum_{\sigma:\tau\preceq\sigma} R_\sigma^{\mathrm{DS}}, \qquad \text{for all } \tau \in \Pi(S). \tag{18}$$

It follows by (18) that

$$\sum_{\tau\in\Pi(S)} \mathcal{D}(\tau)R_\tau^{\text{FS}} = \sum_{\tau\in\Pi(S)} \mathcal{D}(\tau)\Big(\sum_{\sigma:\tau\preceq\sigma} R_\sigma^{\text{DS}}\Big) = \sum_{\sigma\in\Pi(S)} \Big[\sum_{\tau:\tau\preceq\sigma} \mathcal{D}(\tau)\Big] R_\sigma^{\text{DS}}.$$

This proves the first claim.

Consider the second claim. When there exists a mapping $\mathcal{F}_2(\sigma)$ such that $\sum_{\sigma\in\Pi(S)} \mathcal{F}_2(\sigma)R_\sigma^{\text{FS}} = \sum_{\sigma\in\Pi(S)} \mathcal{D}(\sigma)R_\sigma^{\text{DS}}$, we can apply the first claim of this lemma to conclude that

$$\mathcal{D}(\sigma) = \sum_{\tau:\tau\preceq\sigma} \mathcal{F}_2(\tau), \qquad \text{for all } \sigma\in\Pi(S). \tag{19}$$

We can rewrite (19) as $\mathcal{D}(\sigma) = \mathcal{F}_2(\sigma) + \sum_{\tau:\tau\prec\sigma} \mathcal{F}_2(\tau)$, which implies that

$$\mathcal{F}_2(\sigma) = \mathcal{D}(\sigma) - \sum_{\tau:\tau\prec\sigma} \mathcal{F}_2(\tau), \qquad \text{for all } \sigma\in\Pi(S).$$

This proves the second claim of Lemma 7. □

We now proceed to prove Theorem 1. In the special case where $\sigma = \hat{0}$ (the finest partition), the induced $\mathcal{G}_\sigma$ has m nodes and a single edge between each of $i_1\&i_2$, $i_2\&i_3$, $\ldots$, $i_m\&i_1$. It follows that $f_{\hat{0}}(A, j_1, j_2, \ldots, j_k) = A_{i_1i_2}A_{i_2i_3}\ldots A_{i_mi_1}$. As a result,

$$C_m = \sum_{\substack{1\le i_1,i_2,\ldots,i_m\le n\\ \text{(distinct)}}} f_{\hat{0}}(A, i_1, i_2, \ldots, i_m) = R_{\hat{0}}^{\text{DS}}.$$

We can write it equivalently as

$$C_m = \sum_{\sigma\in\Pi(S)} \mathcal{D}(\sigma)R_\sigma^{\text{DS}}, \qquad \text{with} \quad \mathcal{D}(\sigma) := \begin{cases} 1 & \text{if } \sigma = \hat{0},\\ 0 & \text{otherwise.}\end{cases} \tag{20}$$

Using the second claim of Lemma 7, we immediately obtain that

$$C_m = \sum_{\sigma\in\Pi(S)} \mathcal{F}(\sigma)R_\sigma^{\text{FS}}, \qquad \text{with} \quad \mathcal{F}(\sigma) = \begin{cases} 1 & \text{if } \sigma = \hat{0},\\ -\sum\limits_{\hat{0}\preceq\tau\prec\sigma} \mathcal{F}(\tau) & \text{if } \sigma \ne \hat{0}.\end{cases} \tag{21}$$

We recall the recursive definition of the Möbius function: for any $\pi, \sigma \in \Pi(S)$ with $\pi \prec \sigma$, $\mu(\pi,\pi) = 1$, and $\mu(\pi,\sigma)$ is defined recursively by $\mu(\pi,\sigma) = -\sum_{\pi\preceq\tau\prec\sigma} \mu(\pi,\tau)$. Comparing $\mathcal{F}(\sigma)$ in (21) with this definition, we conclude that

$$\mathcal{F}(\sigma) = \mu(\hat{0},\sigma), \qquad \text{for } \sigma\in\Pi(S). \tag{22}$$

The Möbius function $\mu(\hat{0},\sigma)$ has an explicit expression, which is well-known in the literature and presented in the following lemma (e.g., see (Stanley, 2011) for a proof):

Lemma 8 (Explicit formula of Möbius functions on partition lattices)**.** *For any partition σ with k blocks $\{S_1, S_2, \ldots, S_k\}$ of $\{1, 2, \ldots, m\}$, we have $\mu(\hat{0},\sigma) = (-1)^{m-k}\prod_{i=1}^k (|S_i| - 1)!$.*

We plug (22) into (21) and let $\mathcal{G}_{m,k,t}$ be the same isomorphic class as in (15). It yields that

$$C_m = \sum_{\sigma\in\Pi(S)} \mu(\hat{0},\sigma)R_\sigma^{\text{FS}} = \sum_{k=2}^{m}\sum_{t=1}^{b_{m,k}} \sum_{\sigma:\,\mathcal{G}_\sigma\in\mathcal{G}_{m,k,t}} \mu(\hat{0},\sigma)R_\sigma^{\text{FS}}, \tag{23}$$

For partitions σ and $\tilde{\sigma}$ such that their corresponding multi-graphs are isomorphic, we have $R_\sigma^{\text{FS}} = R_{\tilde{\sigma}}^{\text{FS}}$ (the derivation for this equality is similar to that for (14), except for a minor difference that the distinct sum there is replaced by the full sum here). Additionally, from the expression in Lemma 8, we also observe that $\mu(\hat{0},\sigma) = \mu(\hat{0},\tilde{\sigma})$. Let $\sigma_{m,k,t}$ be the canonical partition in $\mathcal{G}_{m,k,t}$. It is seen that

$$\mu(\hat{0},\sigma)R_\sigma^{\text{FS}} = \mu(\hat{0},\sigma_{m,k,t})R_{\sigma_{m,k,t}}^{\text{FS}}, \qquad \text{for all } \sigma \text{ such that } \mathcal{G}_\sigma\in\mathcal{G}_{m,k,t}.$$

Recall that $d_{m,k,t} = |\mathcal{G}_{m,k,t}|$. We plug the above equality into (23) to obtain:

$$C_m = \sum_{k=2}^{m-1} \sum_{t=1}^{b_{m,k}} d_{m,k,t} \cdot \mu\big(\hat{0}, \sigma_{m,k,t}\big) \cdot R^{\text{FS}}_{\sigma_{m,k,t}}. \tag{24}$$

Finally, we let $f_{m,k,t}(A, j_1, j_2, \ldots, j_k) := f_{\sigma_{m,k,t}}(A, j_1, j_2, \ldots, j_k)$ and use the expression of R^{FS}_{σ} in (22). It follows that

$$R^{\text{FS}}_{\sigma_{m,k,t}} = \sum_{1 \le j_1, j_2, \ldots, j_k \le n} f_{m,k,t}(A, j_1, j_2, \ldots, j_k).$$

Let $a_{m,k,t} := d_{m,k,t} \cdot \mu\big(\hat{0}, \sigma_{m,k,t}\big)$. We plug the above equality into (24) to get

$$C_m = \sum_{k=2}^{m-1} \sum_{t=1}^{b_{m,k}} a_{m,k,t} \sum_{j_1, j_2, \ldots, j_k} f_{m,k,t}(A, j_1, j_2, \ldots, j_k). \tag{25}$$

Let $h_{m,k,t} = \prod_{i=1}^{k} (|S_i| - 1)!$ as defined in (6), with $\{S_1, S_2, \ldots, S_k\}$ being the blocks of partition $\sigma_{m,k,t}$. It follows by Lemma 8 that $\mu(\hat{0}, \sigma_{m,k,t}) = (-1)^{m-k} \cdot h_{m,k,t}$. It follows that

$$a_{m,k,t} = d_{m,k,t} \cdot (-1)^{m-k} \cdot h_{m,k,t}. \tag{26}$$

The claim follows by combining (26) with (25). □

E.4. Proof of Lemma 3

It is sufficient to show that the value of the Full-Sum remains the same before and after either types of the pruning.

Consider the Type I pruning first. Without loss of generality, suppose node 1 is the Pendant and node 2 is the Hinge. Let v and u be their respective node labels, and $M^{(1)}, M^{(2)}, \ldots, M^{(s)}$ be the labels of the edges between them. We can express the Full-Sum of the original FAM $\mathcal{G}$ as:

$$\begin{aligned} \text{FS}(\mathcal{G}) &= \sum_{j_1, j_2, \ldots, j_k} f^{(1,2)}(A, j_3, \ldots, j_k) \cdot u_{j_2} \cdot \prod_{l=1}^{s} M^{(l)}_{j_2 j_1} \cdot v_{j_1} \\ &= \sum_{j_2, \ldots, j_k} f^{(1,2)}(A, j_3, \ldots, j_k) \cdot u_{j_2} \cdot \sum_{j_1} \left(\prod_{l=1}^{s} M^{(l)}_{j_2 j_1} \right) \cdot v_{j_1} \end{aligned}$$

where $f^{(1,2)}(A, j_3, \ldots, j_k)$ captures all other terms in the product that are independent of j_1 and j_2.

Let $M = M^{(1)} \circ M^{(2)} \circ \cdots \circ M^{(s)}$ denote the Hadamard product of edge matrices. Then:

$$\begin{aligned} \text{FS}(\mathcal{G}) &= \sum_{j_2, \ldots, j_k} f^{(1,2)}(A, j_3, \ldots, j_k) \cdot u_{j_2} \cdot \sum_{j_1} M_{j_2 j_1} \cdot v_{j_1} \\ &= \sum_{j_2, \ldots, j_k} f^{(1,2)}(A, j_3, \ldots, j_k) \cdot [u \circ (M \cdot v)]_{j_2} \end{aligned} \tag{27}$$

We can see that after pruning, the new FAM $\mathcal{G}^{\text{new}}$ has node 1 removed and the hinge node 2 now has an updated label $u^{\text{new}} = u \circ (M \cdot v)$. Its Full-Sum is:

$$\begin{aligned} \text{FS}(\mathcal{G}^{\text{new}}) &= \sum_{j_2, \ldots, j_k} f^{(1,2)}(A, j_3, \ldots, j_k) \cdot u^{\text{new}}_{j_2} \\ &= \sum_{j_2, \ldots, j_k} f^{(1,2)}(A, j_3, \ldots, j_k) \cdot [u \circ (M \cdot v)]_{j_2} \end{aligned} \tag{28}$$

Therefore, $\text{FS}(\mathcal{G}) = \text{FS}(\mathcal{G}^{\text{new}})$.

Consider the Type II pruning. Without loss of generality, suppose node 1 is the Pendant, and nodes 2 and 3 are its Hinges. Let y, x, and z be their respective node labels. Let $Q^{(1)}, \ldots, Q^{(s)}$ be the labels of edges between nodes 1 and 2, and $R^{(1)}, \ldots, R^{(t)}$ be the labels of edges between nodes 1 and 3.

We can express the Full-Sum of the original FAM $\mathcal{G}$ as:

$$\mathrm{FS}(\mathcal{G}) = \sum_{j_1,\ldots,j_k} f^{(1,2,3)}(A, j_4, \ldots, j_k) \cdot x_{j_2} \cdot \left(\prod_{l=1}^{s} Q^{(l)}_{j_2 j_1}\right) \cdot y_{j_1} \cdot \left(\prod_{l=1}^{t} R^{(l)}_{j_1 j_3}\right) \cdot z_{j_3}$$

where $f^{(1,2,3)}(A, j_4, \ldots, j_k)$ captures all other terms independent of j_1, j_2, and j_3.

Let $Q = Q^{(1)} \circ \cdots \circ Q^{(s)}$ and $R = R^{(1)} \circ \cdots \circ R^{(t)}$. Then:

$$\begin{aligned} \mathrm{FS}(\mathcal{G}) &= \sum_{j_2,j_3,\ldots,j_k} f^{(1,2,3)}(A, j_4, \ldots, j_k) \cdot x_{j_2} \cdot z_{j_3} \cdot \sum_{j_1} Q_{j_2 j_1} \cdot y_{j_1} \cdot R_{j_1 j_3} \\ &= \sum_{j_2,j_3,\ldots,j_k} f^{(1,2,3)}(A, j_4, \ldots, j_k) \cdot x_{j_2} \cdot z_{j_3} \cdot [Q \cdot d(y) \cdot R]_{j_2 j_3} \end{aligned} \tag{29}$$

where $d(y)$ is the diagonal matrix with entries y_{j_1} on the diagonal.

After pruning, the new FAM $\mathcal{G}^{\text{new}}$ has node 1 removed and a new edge with label $E = Q \cdot d(y) \cdot R$ between nodes 2 and 3. Its Full-Sum is:

$$\begin{aligned} \mathrm{FS}(\mathcal{G}^{\text{new}}) &= \sum_{j_2,j_3,\ldots,j_k} f^{(1,2,3)}(A, j_4, \ldots, j_k) \cdot x_{j_2} \cdot E_{j_2 j_3} \cdot z_{j_3} \\ &= \sum_{j_2,j_3,\ldots,j_k} f^{(1,2,3)}(A, j_4, \ldots, j_k) \cdot x_{j_2} \cdot [Q \cdot d(y) \cdot R]_{j_2 j_3} \cdot z_{j_3} \end{aligned} \tag{30}$$

Therefore, we also have $\mathrm{FS}(\mathcal{G}) = \mathrm{FS}(\mathcal{G}^{\text{new}})$.

In both cases, the pruning operation reduces the number of nodes by exactly 1 while preserving the value of the Full-Sum. This completes the proof. □

E.5. Proof of Theorem 4

Before we prove Theorem 4, we show that the pruning algorithm eventually stops.

Consider each pruning operation. Since we remove exactly 1 node in each pruning operation, the number of nodes reduces by exactly 1 before and after each pruning operation. Consider the termination condition. The algorithm terminates either when only one node remains or no prunable nodes (pendants) remain. In either case, when the algorithm terminates, there is at least one node remaining. Therefore, since the algorithm starts with a finite number of nodes k, we conclude that the algorithm must terminate after at most $(k-1)$ pruning steps, which shows that the pruning algorithm eventually stops.

We now proceed to prove Theorem 4. Consider the first case. Let v be the node label for the only one node remaining in the final FAM. There is no other node for the only remaining node to connect to, therefore there is no edge in the final FAM. Hence, the Full-Sum simplifies to:

$$\mathrm{FS}(\mathcal{G}) = \sum_{j_1} v_{j_1} = \mathbf{1}_n' v \tag{31}$$

This is a SEA term by definition, since it is expressible in a succinct algebraic form without any remaining sums over indices.

Consider the second case. In this case, no pendant node of either type is left. First we state a useful lemma.

Lemma 9. *The pruning algorithm preserves the connectivity of the FAM. That is, if a FAM is connected, then after one step of either Type I pruning or Type II pruning, the pruned FAM is still connected.*

Proof of Lemma 9: Consider Type I pruning. Type I pruning deletes a node with 1 immediate neighbor, which does not affect the connectivity of the remaining nodes. This shows that after one step of Type I pruning, the pruned FAM is still connected.

Consider Type II pruning. Type II pruning deletes a node with 2 immediate neighbors, which does not affect the connectivity of the remaining nodes. This shows that after one step of Type II pruning, the pruned FAM is still connected. □

We now proceed to prove Theorem 4. We first show that each of the remaining node in the final FAM has at least 3 immediate neighbors. We prove this by contradiction. Assume that there exists a node with fewer than 3 immediate neighbors. If it has 2 neighbors, it would be a Type II pendant, contradicting our assumption that no pendant nodes remain. If it has 1 neighbor, it would be a Type I pendant, contradicting our assumption that no pendant nodes remain. If it has 0 neighbor, then the FAM would be disconnected. By Lemma 6, the original multi-graph is connected. By Lemma 9, the pruning process preserves connectivity of the FAM. Therefore, when the algorithm terminates, the final FAM must be connected. This leads to a contradiction. Therefore, we conclude that each of the remaining node in the final FAM has at least 3 immediate neighbors.

We then show that the final FAM has at least 4 nodes. Pick a node in the final FAM. Using the above argument, it has at least 3 immediate neighbors. Consider the picked node and three of its immediate neighbors in the final FAM. These are 4 different nodes in the final FAM, therefore we conclude that there must be at least 4 nodes remaining in the final FAM.

Finally, we show that the FS term has at most $\lfloor m/2 \rfloor$ layers of sum. We call a node in the original FAM prunable, if it does not remain in the final FAM, i.e. it is pruned during a step of pruning operation. Denote k the number of nodes in the original FAM, denote p the number of prunable nodes in the original FAM, and denote d_2 the number of degree 2 nodes in the original FAM.

Consider the number of prunable nodes in the original FAM. Note that every node with degree 2 must be prunable, since they can have no more than 2 neighbors. Therefore, we conclude that

$$p \geq d_2. \tag{32}$$

Consider the sum of degrees of all nodes in the original FAM. Since the original FAM has m edges, the sum of degrees of all nodes in the original FAM is $2m$. By Lemma 6, the original FAM is connected, and every node in the original FAM has even degrees. This means that the degree of each node in the original FAM can only be positive even numbers. Therefore, any node whose degree is not 2 has a degree of at least 4. Therefore, the sum of degrees of all nodes in the original FAM is no less than $2d_2 + 4(k - d_2)$. Hence, we have the following inequality:

$$2m \geq 2d_2 + 4(k - d_2). \tag{33}$$

By (33), we conclude that $4(k - d_2) \leq 2m$. Therefore, by (32), we have

$$k - p \leq k - d_2 \leq m/2. \tag{34}$$

Since the number of remaining nodes in the final FAM must be an integer, we conclude that the number of remaining nodes in the final FAM is no greater than $\lfloor m/2 \rfloor$. Since each node in the final FAM corresponds to one layer of summation, we conclude that the FS term has at most $\lfloor m/2 \rfloor$ layers of sum. This completes the proof. □

References

Alfarano, A., Charton, F., and Hayat, A. Global lyapunov functions: A long-standing open problem in mathematics with symbolic transformers. In *Advances in Neural Information Processing Systems 38*, 2024. URL https://openreview.net/forum?id=kOMrm4ZJ3m. Poster.

Bondy, A. and Murty, U. *Graph Theory with Applications*. Wiley, 1991. ISBN 9780471363248. URL https://books.google.com/books?id=7EWKkgEACAAJ.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. Language models are few-shot learners. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H. (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 1877–1901. Curran Associates, Inc., 2020.

Chang, Y. and Fu, H. The number of 6-cycles in a graph. *Bulletin of the Institute of Combinatorics and its Applications*, 39, 01 2003.

Davies, A., Veličković, P., Buesing, L., Blackwell, S., Zheng, D., Tomašev, N., Tanburn, R., Battaglia, P., Blundell, C., Juhász, A., Lackenby, M., Williamson, G., Hassabis, D., and Kohli, P. Advancing mathematics by guiding human intuition with AI. *Nature*, 600(7887):70–74, 2021. doi: 10.1038/s41586-021-04086-x. URL https://doi.org/10.1038/s41586-021-04086-x.

Drori, I., Zhang, S., Shuttleworth, R., Tang, L., Lu, A., Ke, E., Liu, K., Chen, L., Tran, S., Cheng, N., Wang, R., Singh, N., Patti, T. L., Lynch, J., Shporer, A., Verma, N., Wu, E., and Strang, G. A neural network solves, explains, and generates university math problems by program synthesis and few-shot learning at human level. *Proceedings of the National Academy of Sciences*, 119(32):e2123433119, 2022. doi: 10.1073/pnas.2123433119. URL https://www.pnas.org/doi/abs/10.1073/pnas.2123433119.

Evstafev, E. Token-hungry, yet precise: Deepseek r1 highlights the need for multi-step reasoning over speed in math. *ArXiv*, 2024.

Gao, C. and Lafferty, J. Testing for global network structure using small subgraph statistics. *arXiv preprint arXiv:1710.00862*, 2017.

Guo, D., Yang, D., Zhang, H., Song, J., Zhang, R., Xu, R., Zhu, Q., Ma, S., Wang, P., Bi, X., et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Harary, F. *Graph Theory*. Addison-Wesley Series in Mathematics. Addison-Wesley Longman, Incorporated, 1969. ISBN 9780201027877. URL https://books.google.com/books?id=0A4rmgEACAAJ.

Harary, F. and Manvel, B. On the number of cycles in a graph. *Matematicky casopis*, 21(1):55–63, 1971.

Hendrycks, D., Burns, C., Kadavath, S., Arora, A., Basart, S., Tang, E., Song, D., and Steinhardt, J. Measuring mathematical problem solving with the MATH dataset. In *Advances in Neural Information Processing Systems 34 (Datasets & Benchmarks Track)*, 2021. doi: 10.48550/arXiv.2103.03874. URL https://arxiv.org/abs/2103.03874.

Jin, J., Ke, Z. T., and Luo, S. Optimal adaptivity of signed-polygon statistics for network testing. *The Annals of Statistics*, 49(6):3408–3433, 2021.

Jin, J., Ke, Z. T., Luo, S., and Wang, M. Optimal estimation of the number of network communities. *Journal of the American Statistical Association*, 118(543):2101–2116, 2023.

Jin, J., Ke, Z. T., Luo, S., and Ma, Y. Optimal network pairwise comparison. *Journal of the American Statistical Association*, 120(550):1048–1062, 2025a. doi: 10.1080/01621459.2024.2393471. URL https://doi.org/10.1080/01621459.2024.2393471.

Jin, J., Ke, Z. T., Sui, B., and Wang, J. Detecting weak spikes by counting cycles. Manuscript, 2025b.

Jin, J., Ke, Z. T., Tang, J., and Wang, J. Network goodness-of-fit for the block-model family. *Journal of the American Statistical Association*, pp. 1–27, 2025c.

Jin, J., Ke, Z. T., Sui, B., and Wang, J. Estimating spikes with newton and prony. Manuscript, 2026.

Kong, W. and Valiant, G. Spectrum estimation from samples. *The Annals of Statistics*, 45(6):2218–2247, 2017.

Lample, G., Lachaux, M.-A., Lavril, T., Martinet, X., Hayat, A., Ebner, G., Rodriguez, A., and Lacroix, T. Hypertree proof search for neural theorem proving. In *Advances in Neural Information Processing Systems 35*, 2022. doi: 10.48550/arXiv.2205.11491. URL https://arxiv.org/abs/2205.11491.

Li, J. and Chen, S. X. Two sample tests for high-dimensional covariance matrices. *The Annals of Statistics*, 40(2):908–940, 2012.

Movarraei, N. and Boxwala, S. On the number of cycles in a graph. *Open Journal of Discrete Mathematics*, 6:41–49, 03 2016. doi: 10.4236/ojdm.2016.62005.

Perepechko, S. and Voropaev, A. The number of fixed length cycles in an undirected graph. explicit formulae in case of small lengths. *Mathematical Modeling and Computational Physics (MMCP2009)*, 148, 2009.

Romera-Paredes, B., Barekatain, M., Novikov, A., Balog, M., Kumar, M. P., Dupont, E., Ruiz, F. J. R., Ellenberg, J. S., Wang, P., Fawzi, O., Kohli, P., and Fawzi, A. Mathematical discoveries from program search with large language models. *Nature*, 625:468–475, 2024. doi: 10.1038/s41586-023-06924-6. URL https://doi.org/10.1038/s41586-023-06924-6.

Stanley, R. P. *Enumerative Combinatorics*. Cambridge Studies in Advanced Mathematics. Cambridge University Press, 2 edition, 2011.

Trinh, T. H. and *et al*. Solving olympiad geometry without human demonstrations. *Nature*, 625(7995):4760–482, 2024.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., ichter, b., Xia, F., Chi, E., Le, Q. V., and Zhou, D. Chain-of-thought prompting elicits reasoning in large language models. In Koyejo, S., Mohamed, S., Agarwal, A., Belgrave, D., Cho, K., and Oh, A. (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 24824–24837. Curran Associates, Inc., 2022.

Zheng, K., Han, J. M., and Polu, S. miniF2F: A cross-system benchmark for formal olympiad-level mathematics. In *International Conference on Learning Representations*, 2022. doi: 10.48550/arXiv.2109.00110. URL https://arxiv.org/abs/2109.00110.