

Verifying the minimal cubic graph without Gallai vertice in parallel with code generated by LLM: An initial study

Arthur H. D. Rodrigues¹, Alfredo Goldman¹

¹Instituto de Matemática, Estatística e Ciência da Computação – Universidade de São Paulo (USP)
São Paulo – SP – Brasil

{arthur,gold}@ime.usp.br

Abstract. A Gallai vertex in a graph is a vertex that belongs to all of its longest paths. In this paper, we study the implementation of a parallel algorithm to check if a given graph has a Gallai vertex. We implemented this algorithm purely with genAI and evaluated how well it could parallelize the algorithm using OpenMP. We confirmed a preexisting conjecture about the smallest cubic graph with no Gallai vertex and observed that genAI could implement and parallelize the algorithm in different granularities.

1. Introduction

For any two longest paths of a graph, it is a well known fact that there is at least one vertex that belongs to both of them. In 1968, Gallai conjectured that for any graph, there is a vertex that belongs to *all* longest paths. Such a vertex is called *Gallai vertex*. This conjecture is false and the smallest counterexample is shown in Figure 1a [Chen 2018]. [de Lazari Bento 2026] extended this counterexample to a cubic graph, illustrated in Figure 1b, and conjectured that it is the smallest cubic graph with no Gallai vertex.

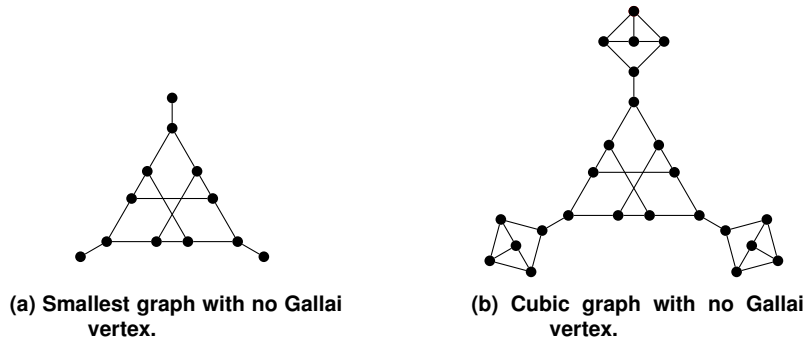


Figure 1. The smallest graph with no Gallai vertex and the smallest cubic graph with no Gallai vertex. Reproduction of [de Lazari Bento 2026].

The motivations of this work are to verify this conjecture by implementing a parallel algorithm that checks for Gallai vertices in a given graph and to validate how well a commercial LLM such as Gemini Pro can implement such algorithm. We confirmed that the graph illustrated in Figure 1b is the smallest cubic graph with no Gallai vertex.

2. The Algorithm

For a fixed $k \geq 0$, the decision problem: “Does G contain a path of length $\leq k$?” is NP-complete. In short, this means that finding a longest path is computationally expensive

and usually is resolved with brute-force algorithms, which must be parallelized to be feasible to tackle big instances of the problem. Our implementation is no exception.

We start by writing an algorithm that finds a longest path starting from a given vertex v . This subroutine is described in Algorithm 1 and is a modified version of depth first search (DFS). This version was presented by [Fieger et al. 2019].

Each vertex is either marked or unmarked. All vertices start unmarked. The algorithm receives an unmarked vertex v and the longest currently known path P as arguments. We mark v and for each of its unmarked neighbors we recursively call `mDFS` with the path Pv , which is the path P with v concatenated at the end. In case the new found path is longer than the current one, P_{max} is updated. After iterating over the neighborhood of v , we unmark it. Doing so is important to allow v to appear in possible longer paths.

Algorithm 1 <code>mDFS</code> (v, P)	Algorithm 2 <code>FLP</code> (G)
1: $P_{max} \leftarrow P$	1: $P_{max} \leftarrow \emptyset$
2: mark v	2: for $v \in V(G)$ do
3: for $vw \in E$ and w is unmarked do	3: $P \leftarrow \text{mDFS}(v, \emptyset)$
4: $P \leftarrow \text{mDFS}(w, Pv)$;	4: if $ P > P_{max} $ then
5: if $ P > P_{max} $ then	5: $P_{max} \leftarrow P$
6: $P_{max} \leftarrow P$	6: end if
7: end if	7: end for
8: end for	8: return P_{max}
9: unmark v	9:
10: return P_{max}	10:

Our implementation of this algorithm is purely sequential. If the maximum degree of the graph is d and n is its number of vertices, then the running time of `mDFS` is $O(d^n)$ [Fieger et al. 2019].

To find the overall longest path, we need to call `mDFS` for each vertex of the graph and keep track of the currently longest one. This is what `FLP`, short for *find longest path* and described in Algorithm 2, does. The work of `FLP` is $O(nd^n)$, because it calls `mDFS` n times. The loop at the line 2 may be parallelized by running each iteration in a different thread. This yields a depth of $O(d^n)$ in a n -processor machine.

The next step is to use `FLP` to find a Gallai vertex, if G contains any. To do so, we use the following lemma. We denote by $G - v$ the graph obtained from G by removing the vertex v .

Lemma 1. *Let G be a graph and v a vertex. If a longest path of $G - v$ is strictly shorter than a longest path in G , then v is a Gallai vertex of G .*

Proof. We prove by contradiction. Suppose v is not a Gallai vertex, then there is a longest path P in G that does not contain v . Then P is also a longest path of $G - v$, but longest paths of $G - v$ are strictly shorter than longest paths in G . A contradiction. $\square$

If v is a Gallai vertex and P a longest path of G , then $v \in V(P)$. Thus, to check if a graph has a Gallai vertex, it is sufficient to check the vertices of a longest path. So

the algorithm to check for Gallai vertices, called FGV (short for *find Gallai vertex*) and described in Algorithm 3, first finds a longest path P of G and then, for each vertex $v \in V(P)$, finds some longest path P' in $G - v$ and compares their lengths. If P' is shorter than P , then, by Lemma 1, v is a Gallai vertex.

Algorithm 3 FGV (G)

```

1:  $P \leftarrow \text{FLP}(G)$ 
2: for  $v \in V(P)$  do
3:    $P' \leftarrow \text{FLP}(G - v)$ 
4:   if  $|P'| < |P|$  then
5:     return  $v$ 
6:   end if
7: end for
8: return nil

```

In the worst case, the loop at line 2 iterates over all vertices of P . Since $|V(P)|$ is $O(n)$, then FLP is called $O(n)$ times and thus the total work of FGV is $O(n^2 d^n)$. Similar to FLP, we may parallelize the loop of the line 2, which yields a depth of $O(d^n)$.

3. Experiment and Use of genAI

Although verifying the minimal counterexample mentioned in Section 1 was the main motivation of this paper, we are also interested in evaluating the use of genAI for code generation for parallel combinatorial algorithms. We used Gemini Pro to generate all code in C. The code is available at https://github.com/ArthurHDRodrigues/ERAD_SP_2026_Gallai, licensed under GPL-2.0.

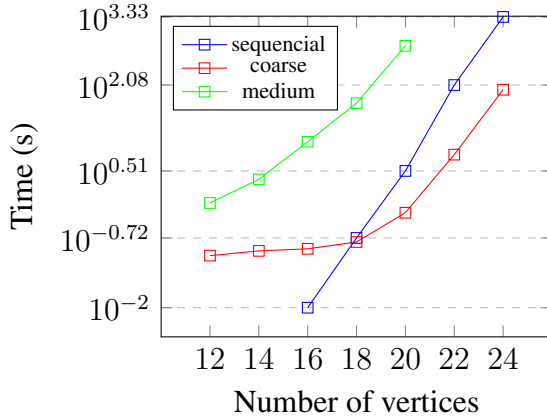
We first prompted Gemini to code Algorithm 1, making sure to point out it is a modified version of DFS. We then prompted for a serial version of the Algorithm 2, commenting on how to use Algorithm 1 as a subroutine and then prompted for a serial version of Algorithm 3.

We purposefully did not mention any optimization. Two were suggested. The first one is that if, at any moment, the current path has length n , then it is the longest possible path and all vertices are Gallai vertices.

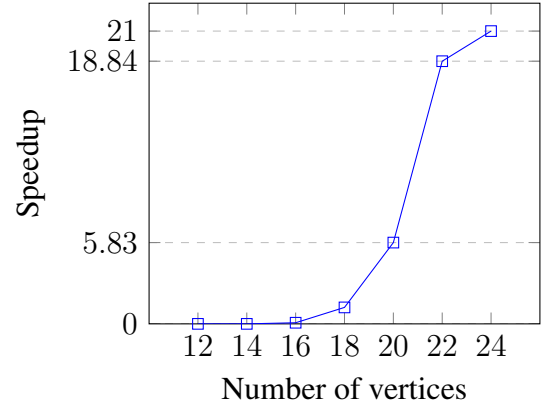
The second is that, in FGV, instead of explicitly removing v , a new parameter is added to mDFS. It receives a vertex u which is “forbidden”, the DFS will ignore u in its traversal through the graph. This optimization avoided manipulating the graph, which reduced the amount of memory (de)allocation.

The first implementation had a use-after-free bug, which Gemini successfully debugged without human intervention. After that, we prompted it to parallelize the algorithm using OpenMP. Gemini proposed three parallelism granularities: At coarse level, processing multiple graphs at once. At medium level, processing one graph in parallel. At fine-grained level, parallelizing DFS. After implementing the coarse level, the algorithm consumed too much RAM at once and crashed. To fix it, Gemini decided to process the graphs in batches.

After implementing the medium level of granularity, we observed that it performed worse than the sequential implementation, as seen in Figure 2a. Gemini blamed



(a) Execution time of the different variants of the algorithms



(b) Speedup between the coarse and sequential algorithms.

this on parallelization overhead and suggested we discard this implementation and pointed out that it wasn't worth to implement the fine-grained parallelism due to this overhead. After analyzing the generated code, we agreed with it.

We consider this implementation with genAI a success and believe it was due to two factors. First, brute-force algorithms are simple to describe in natural language and to parallelize; and the core of our application is DFS, a very well known algorithm in the literature which was used to train it.

We downloaded all cubic graphs sized between 12 and 24 vertices from the House of Graphs [Coolsaet et al. 2023] and ran the sequential and both the parallels versions of the algorithm in a machine with AMD EPYC 9734 112-Core Processor and 1.5 Terabytes of RAM. We then noticed a speedup of up to 21 in the coarse version when compared to the serial one, as seen in Figure 2b. During this work, computational resources from the Multi-User Equipment were used, thanks to the FAPESP project: 2022/16618-2 (Heterogeneous Parallel achine), linked to grant: 2019/26702-8 (Trends in High-Performance Computing, from Resource Management to New Computer Architectures).

References

- Chen, F. (2018). Order of the smallest counterexample to Gallai's conjecture. *Czechoslovak Mathematical Journal*, 68(2):341–369.
- Coolsaet, K., D'hondt, S., and Goedgebeur, J. (2023). House of graphs 2.0: a database of interesting graphs and more. *Discrete Applied Mathematics*, 325:57–65.
- de Lazari Bento, H. (2026). Minimum longest path transversals in cubic graphs. Master thesis, University of São Paulo.
- Fieger, K., Balyo, T., Schulz, C., and Schreiber, D. (2019). Finding optimal longest paths by dynamic programming in parallel. In *Proceedings of the 12th Annual SoCS*.