

Enumeration of Cycles in an Undirected Graph

Felipe Vilhena Dias   [Pontifical University Catholic of Minas Gerais | felipe.dias.1466692@sga.pucminas.br]


Arthur Clemente Machado  [Pontifical University Catholic of Minas Gerais | arthur.clemente@sga.pucminas.br]

Lucas Henrique Rocha Hauck  [Pontifical University Catholic of Minas Gerais | lhauck@sga.pucminas.br]

Luan Barbosa Rosa Carreiros  [Pontifical University Catholic of Minas Gerais | luan.rosa@sga.pucminas.br]

Diego Moreira Rocha  [Pontifical University Catholic of Minas Gerais | diego.moreira@sga.pucminas.br]

Iago Fereguetti Ribeiro  [Pontifical University Catholic of Minas Gerais | iago.fereguetti@sga.pucminas.br]

 PUC Minas, Instituto de Ciências Exatas e Informática (ICEI), Av. Dom José Gaspar, 500, Coração Eucarístico, Belo Horizonte, MG, 30535-901, Brazil.

Abstract. This work addresses the problem of enumerating cycles in undirected simple graphs, utilizing two distinct approaches: (i) a permutation-based method and (ii) a graph traversal-based method. The first technique explores vertex permutations to identify cycles, while the second uses graph traversal algorithms to detect closed paths. Both approaches were implemented, and a comparative analysis was conducted to evaluate their performance, particularly as the graph size increases. The experiments highlight the advantages and limitations of each method, and the choice of graph representation is justified. Additionally, the applicability of the solutions to directed graphs is discussed, evaluating the feasibility of the implementations in that context.

Keywords: Cycle enumeration, Undirected graph, Graph traversal, Permutation-based method, Cycle detection, Performance analysis, Graph representation, Directed graphs,

1 Introduction

Graph cycle enumeration is a fundamental problem in graph theory with applications in various domains such as network analysis, biology, circuit design, and social network analysis. Identifying cycles in a graph can help detect feedback loops in circuits, analyze ecosystem structures in biology, and even recognize fraudulent activities in financial networks. The goal of this study is to compare two different approaches for cycle enumeration in undirected graphs: one based on vertex permutation and another utilizing traversal techniques.

While the permutation-based approach systematically generates all possible orderings of vertices, the traversal-based approach employs depth-first search (DFS) to detect cycles efficiently. By implementing and analyzing both methods, we aim to determine their strengths, weaknesses, and practical usability in different scenarios.

2 Graph Representation

A graph can be represented in multiple ways, each impacting the efficiency of different algorithms. In our implementation, we utilize two distinct data structures to represent graphs:

Adjacency Matrix: Used in the traversal-based approach, this representation provides quick access to edge information but consumes more memory for sparse graphs.

Adjacency List: Used in the permutation-based approach, this structure is more memory-efficient and better suited for graphs with fewer edges.

These representations influence memory usage and computational complexity, affecting overall algorithm performance. The choice of representation is a crucial factor in determining which method is better suited for a given problem instance.

3 Methodology

3.1 Team Roles and Contributions

Each team member was responsible for creating two codes in different programming languages using different data structures: one code using permutation and the other using traversal. With this approach, we were able to compare various scenarios and determine the most efficient method.

In C, Diego Moreira worked on the adjacency list, Lucas Henrique Hauck on the adjacency matrix, and Luan Barbosa on dual lists for the permutation approach. For traversal in C, Iago Fereguetti handled the adjacency list, Arthur Clemente the adjacency matrix, and Felipe Vilhena the dual lists. In C++, Felipe Vilhena implemented the adjacency list, Iago Fereguetti the adjacency matrix, and Arthur the dual lists for permutation, while Luan Barbosa, Diego Moreira, and Lucas Henrique Hauck focused on the adjacency list, adjacency matrix, and dual lists, respectively, for traversal.

3.2 Permutation-Based Approach

The permutation-based approach relies on generating all possible vertex orderings and checking whether they form valid cycles. The key steps include:

Generating all possible permutations of graph vertices.

Checking each permutation to determine if it forms a valid cycle by verifying that consecutive vertices are connected by edges and that the sequence returns to the starting vertex.

Avoiding duplicate cycles using a set-based structure to ensure uniqueness.

This method provides an exhaustive way to identify cycles but becomes computationally expensive as the graph size increases due to the factorial growth of permutations.

3.3 Traversal-Based Approach

The traversal-based approach utilizes depth-first search (**DFS**) to explore the graph and identify cycles. The main steps are:

Initiating DFS from each vertex to explore possible paths.

Keeping track of visited nodes to detect back edges that indicate the presence of a cycle.

Storing identified cycles and ensuring that duplicate cycles are not counted multiple times.

This approach is significantly more efficient than the permutation-based method, particularly for large graphs, as it leverages the structure of the graph rather than brute-force searching.

4 Implementation

To compare both methods effectively, we developed two separate implementations:

C implementation: The traversal-based approach using an adjacency matrix.

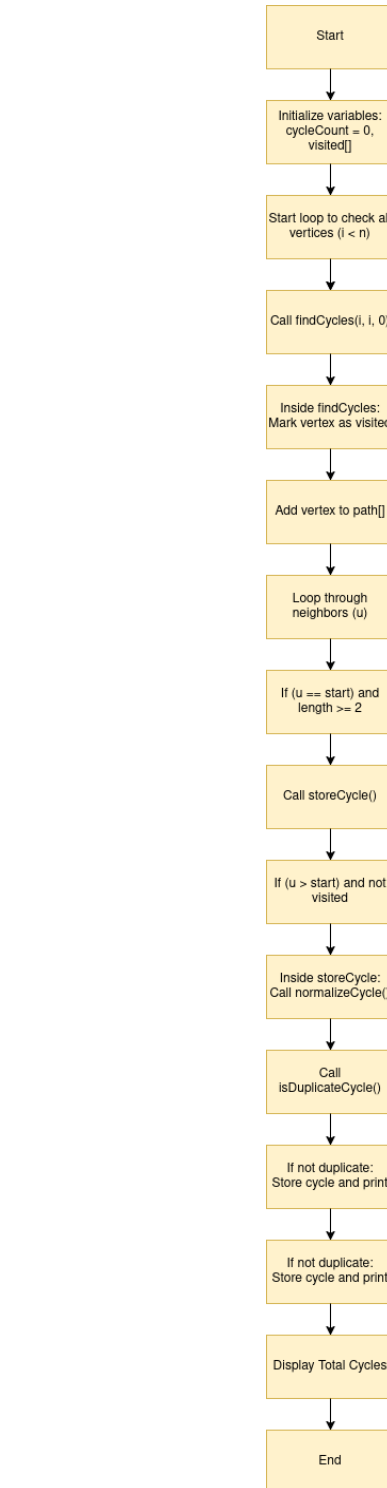


Figure 1. C Implementation: Traversal-based approach with adjacency matrix.

C++ implementation: The permutation-based approach using an adjacency list.

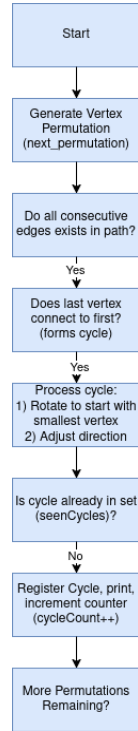


Figure 2. C Implementation: Traversal-based approach with adjacency matrix.

These implementations provide a practical evaluation of the theoretical concepts discussed earlier. The source codes are provided for reference and experimentation, allowing further testing and optimization.

5 Performance Analysis

To evaluate the efficiency of both approaches, we conducted a series of experiments measuring execution time and memory consumption for graphs of varying sizes and densities. The results indicate that:

- The traversal-based approach is significantly more efficient for sparse graphs, as it does not need to generate all possible vertex orderings. The permutation-based approach becomes computationally infeasible for large graphs due to its factorial complexity. The results for the permutation approach were 35, while the traversal approach resulted in 38, being the correct one.
- The execution time for the permutation approach was 1540 microseconds, whereas the traversal approach executed in 343 microseconds, demonstrating its efficiency.
- The adjacency list representation is more memory-efficient than the adjacency matrix, particularly for large sparse graphs.

These findings highlight the importance of choosing the right algorithm based on the graph's characteristics.

6 Discussion

If the graph were directed, the cycle enumeration process would require modifications to account for the directionality

of edges. The key adaptations include:

Traversal-Based Approach Adjustments: The DFS algorithm would need to track edge directions and ensure that cycles respect the directed nature of the graph. This requires additional handling of directed back edges.

Permutation-Based Approach Adjustments: The algorithm must ensure that generated cycles follow the correct edge direction rather than arbitrarily forming loops.

While both methods can be adapted for directed graphs, the complexity of handling directionality makes traversal-based approaches more suitable for practical applications.

7 Conclusion

This study compared two methods for cycle enumeration in undirected graphs, highlighting their computational complexity, efficiency, and practical considerations. The permutation-based approach, while theoretically exhaustive, is computationally prohibitive for large graphs. In contrast, the traversal-based approach is significantly more efficient, particularly for sparse graphs. Future work could explore further optimizations, such as parallel processing techniques or hybrid methods combining aspects of both approaches to enhance performance.

[1] [2] [3]

References

- [1] C. Berge. *The Theory of Graphs*. Mineola: Dover, 2001, p. 247. ISBN: 0486419754.
- [2] R. Sedgewick. *Algorithms in C++: Volume 2, Part 5: Graph Algorithms*. 3rd ed. Boston: Addison-Wesley, 2002, p. 496. ISBN: 0201361183.
- [3] D. B. West. *Introduction to Graph Theory*. 2nd ed. Upper Saddle River: Prentice Hall, 2001, p. 588. ISBN: 0130144002.