Implementation and Empirical Analysis of the Color-Coding Algorithm for Subgraph Detection

March 28, 2025

Abstract

This project implements the color-coding technique introduced by Alon, Yuster, and Zwick [1] for detecting small subgraphs in large graphs. We focus on practical implementations of both the randomized and derandomized variants, benchmarking their performance against theoretical bounds. Experimental evaluation will analyze runtime scaling on worst-case inputs (e.g., sparse graphs with long paths) and naturally distributed graphs (e.g., Erdős-Rényi, planar). The project aims to identify practical optimizations and validate theoretical complexity claims, thereby bridging the gap between algorithmic theory and real-world performance.

1 Technical Background

The color-coding method [1] is a randomized framework designed to simplify subgraph detection in large graphs. Its main ideas are as follows:

- **Core Mechanism:** Vertices in a graph $G = (V, E)$ are randomly assigned colors from the set $\{1, \ldots, k\}$. A subgraph is termed *colorful* if every vertex within it receives a distinct color. This property allows the use of dynamic programming to detect colorful paths and cycles. In particular, Theorem 3.3 in [1] shows that if G contains a simple (colorful) path of length $k-1$, then it can be found in $2^{O(k)} \cdot V$ expected time for undirected graphs and $2^{O(k)} \cdot E$ expected time for directed graphs.
- **Derandomization:** The randomized algorithm can be made deterministic by replacing the random color assignments with a k-perfect family of hash functions (see Schmidt and Siegel [2]). This derandomization adds only an extra $O(\log V)$ factor in the worst-case running time.
- **Extensions:** Beyond paths and cycles, the method extends to detecting any subgraph with bounded tree-width. In such cases (see Theorem 6.3 in [1]), if the subgraph H has k vertices and tree-width t, then a copy of H in G can be found in $O(2^{O(k)} \cdot V^{t+1})$ expected time.
- **Alternative Approaches:** The original work also describes an alternative based on *random orientations* (Section 2 in [1]), which provides simple algorithms with competitive runtime bounds.

2 Implementation Objectives

2.1 Algorithm Variants

- **Randomized Path and Cycle Detection**
  - Implement the dynamic programming approach to detect colorful paths as described in Lemma 3.1 of [1]. For undirected graphs, the algorithm should run in $2^{O(k)} \cdot V$ expected time; for directed graphs, in $2^{O(k)} \cdot E$ expected time.
  - Optimize the memory usage when storing dynamic programming tables, targeting the theoretical space complexity of $O(k \cdot 2^k \cdot V)$.
- **Derandomized Version**
  - Replace random color assignments with a precomputed k-perfect family of hash functions using Schmidt-Siegel constructions [2].

  - Note that the derandomized version incurs an additional $O(\log V)$ factor in its worst-case runtime.

- **Generalization to Bounded Tree-Width Subgraphs**
  - Extend the implementation to detect subgraphs with bounded tree-width (e.g., forests or series-parallel graphs) using a similar dynamic programming approach. The expected runtime should be on the order of $O(2^{O(k)} \cdot V^{t+1})$ for a subgraph of tree-width t.
- **Alternative Method: Random Orientations**
  - Optionally implement the random orientations method (Section 2 in [1]) as an alternative approach. This method directs edges based on a random permutation and then finds simple paths or cycles in the resulting acyclic graph.

2.2 Performance Targets

- Achieve empirical runtime scaling that deviates by less than 10% from the theoretical $2^{O(k)}$ factor for $k \leq 15$.
- Optimize cache usage during dynamic programming table traversal to reduce hidden constant factors.

3 Experimental Methodology

3.1 Graph Generation

- **Worst-Case Inputs:** Generate sparse graphs with long paths by carefully controlling the number of edges (e.g., through backedge-limited depth-first search).
- **Natural Distributions:**
  - **Erdős-Rényi Graphs:** Generate graphs $G(n, p)$ with $p = \Theta(1/n)$.
  - **Planar Graphs:** Use Delaunay triangulation or other planar graph generation methods.
  - **Power-law Networks:** Generate graphs based on the Barabási-Albert model.

3.2 Benchmarking Framework

- **Runtime Metrics:** Measure the running time $T(k, V)$ for various combinations of k (e.g., 5, 10, 15) and graph sizes V (e.g., 10^3, 10^4, 10^5).
- **Memory Profiling:** Compare the empirical memory usage with the theoretical bound $O(k \cdot 2^k \cdot V)$.
- **Statistical Analysis:** Fit the empirical runtimes to models of the form $c \cdot 2^{ak} \cdot V^b$ using nonlinear regression. Compare the fitted constants a, b, and c with theoretical predictions.
- **Heuristic Validation:** Experiment with early termination and adaptive color sampling techniques to explore additional practical optimizations.

4 Expected Contributions

4.1 Implementation Artifacts

- A modular, open-source codebase (in C++ and/or Python with optimized libraries) implementing both randomized and derandomized color-coding algorithms.
- Precomputed k-perfect hash families for values of k up to at least 20.

4.2 Empirical Insights

- Quantitative analysis comparing the $2^{O(k)}$ theoretical bounds with observed runtimes (for example, whether $O(2^{3.2k})$ is observed versus $O(2^{2.8k})$ as predicted).
- Identification of graph classes (e.g., planar, power-law) where the derandomization overhead outweighs the benefits of randomization.

4.3 Practical Guidelines

- Recommendations on thresholds for preferring the randomized versus derandomized variants based on graph size and subgraph parameters.
- Cache optimization strategies for dynamic programming table traversal.

5 Conclusion

This project will bridge the gap between theoretical algorithm design and practical performance for subgraph detection using the color-coding method. By implementing both the randomized and derandomized approaches—and possibly an alternative random orientations method—we aim to validate the theoretical complexity claims of Alon et al. [1] while uncovering practical insights and optimizations. Empirical evaluations will provide a deeper understanding of how graph structure, density, and input size affect the performance of these algorithms.

References

[1] N. Alon, R. Yuster, and U. Zwick, "Color-Coding," *Journal of the ACM*, vol. 42, no. 4, pp. 844–856, 1995.

[2] J. P. Schmidt and A. Siegel, "The spatial complexity of oblivious k-probe hash functions," *SIAM Journal on Computing*, vol. 19, no. 5, pp. 775–786, 1990.