Graph Biconnectivity Testing for Tarjan-Vishkin and Slota-Madduri Algorithm

[Intro to Algorithm Engineering Course Project]

GitHub Repo Link : https://github.com/avi1o1/Benchmarking-Graph-Connectivity-Problems

Problem Description

This project addresses the problem of biconnectivity detection in graphs, ie, identifying subgraphs that remain connected upon the removal of any single vertex. Biconnected components are critical for understanding structural robustness in networks.

We compare the performance of two algorithms :

- ➔ **Tarjan-Vishkin** : A classical, DFS-based algorithm with linear-time complexity.
- ➔ **Slota-Madduri** : A parallel-friendly, BFS-based algorithm optimized for large-scale and high-density graphs.

The objective is to evaluate their runtime performance across a variety of large graph datasets, with particular focus on scalability and the potential for parallelization.

Algorithms Overview

Before proceeding to the experimental evaluation and runtime analysis of the algorithms, it is important to establish a foundational understanding of their core principles, implementation strategies, and practical considerations. This section presents an overview of the algorithms studied in this work, focusing on their computational design, strengths, and limitations in the context of biconnectivity detection in graphs.

1. Tarjan-Vishkin Algorithm

- A classical depth-first search (DFS) based method for identifying biconnected components in an undirected graph. It operates by assigning pre-order numbers during traversal and computing low values for each vertex, which capture the earliest reachable ancestor in the DFS tree. These values are used to detect articulation points, bridges, and ultimately delineate the biconnected components.
- A notable aspect of the algorithm is its use of disjoint-set data structures (union-find) to efficiently group vertices into components as edges are explored. This approach ensures that the algorithm operates in linear time, O(V+E), with respect to the number of vertices and edges, making it a robust choice for sequential execution on small to medium-sized graphs.
- The algorithm is relatively straightforward to implement and analyze, with well-defined theoretical guarantees. While it does not exploit parallelism, its clarity and efficiency make it a strong baseline for performance comparisons and correctness verification in biconnectivity analysis.

2. **Slota-Madduri Algorithm**
    - The Slota-Madduri algorithm is a parallel, breadth-first search (BFS) based approach designed for multi-core architectures and large-scale graphs. Unlike DFS-based methods, it employs a level-synchronous traversal, processing vertices in layers to expose opportunities for concurrent execution.
    - To efficiently scale across threads, the algorithm partitions the graph and distributes work dynamically, employing atomic operations and lock-free techniques to reduce contention. This enables effective utilization of shared-memory systems, even in the presence of irregular graph topologies.
    - The algorithm is particularly well-suited for dense or high-degree graphs, where its ability to exploit work-efficient parallelism yields substantial runtime improvements. Further enhancements, such as locality-aware scheduling and dynamic load balancing, ensure that the workload is evenly distributed and memory access patterns remain efficient.
    - In summary, the Slota-Madduri algorithm represents a modern, performance-optimized method that leverages hardware parallelism to address the challenges of biconnectivity detection in massive graphs.

Thus from a theoretical standpoint, the Tarjan-Vishkin algorithm offers an optimal O(V + E) time complexity in the sequential setting, relying on deterministic DFS traversal and efficient union-find operations. Its simplicity and bounded memory access patterns make it suitable for smaller or moderately sized graphs where parallelism is unnecessary. In contrast, the Slota-Madduri algorithm trades algorithmic simplicity for parallel scalability, emphasizing work-efficient design and concurrency-aware techniques. While it may not outperform Tarjan-Vishkin on small graphs due to parallel overhead, it demonstrates superior throughput on large, high-density graphs by exploiting hardware-level parallelism.

In a nutshell, Tarjan-Vishkin approach is preferred on smaller use cases like code dependency analysis and local network simulations; while Slota-Madduri approach is preferred for larger graph use cases like social network analysis, web crawling and telecommunication.

The Experiment

This section outlines the experimental setup and methodology used to evaluate the performance of selected algorithms for detecting biconnected components in graphs. The experiments were designed to analyze runtime characteristics across a diverse range of input graphs, with an emphasis on scalability with input size.

Objectives

- Measure the execution time of each algorithm on graphs of varying sizes.
- Observe how graph size impacts runtime.
- Establish a comparative baseline between algorithms in practical scenarios.

Setup and Environment

- Implementation Language : C++
- Compilation Flags: -O3 optimization for performance
- Execution Platform: Standard multi-core CPU (single-threaded execution only).
- Timing Mechanism: High-resolution wall-clock time using gettimeofday() for microsecond-level accuracy.

Graph Dataset

Dataset Name	Web URL	# Verts.	# Edges	Description
custom-Built	-	50	128	Custom built small test set
congress-Twitter	https://snap.stanford.edu/data/congress-twitter.html	475	13, 289	Twitter interaction network for the US Congress
email-Eu-Core	https://snap.stanford.edu/data/email-Eu-core.html	1,005	25,571	E-mail network
ego-Facebook	https://snap.stanford.edu/data/ego-Facebook.html	4,039	88,234	Facebook social circles network
ca-HepPh	https://snap.stanford.edu/data/ca-HepPh.html	12,008	118,521	Collaboration network of Arxiv High Energy Physics
musae-Facebook	https://snap.stanford.edu/data/facebook-large-page-page-network.html	22,469	171,002	Facebook page-page network with page names
musae-Github	https://snap.stanford.edu/data/github-social.html	37,700	289,003	Social network of Github developers
soc-Slashdot0922	https://snap.stanford.edu/data/soc-Slashdot0902.html	82,167	948,464	Slashdot social network from February 2009

Total eight graph dataset, ranging from small, synthetic graphs (~50 nodes and 128 edges) to large-scale real-world networks (82,000+ nodes and ~950,000 edges).

Execution Flow

For each algorithm–dataset pair, the following steps were executed :

1. **Graph Loading** : Edges were read and converted to adjacency list format. For large datasets, periodic progress logs were printed.
2. **Computation** : The biconnected component algorithm was executed. Time was measured exclusively around the computation logic (excluding file I/O).
3. **Result Reporting** : The number of biconnected components found was logged. Execution time was recorded and stored in result files. Outputs were organized by dataset and algorithm for systematic comparison.
4. **Post-Processing** : Output logs were parsed to extract runtimes.
5. **Plotting** : Tabulated results were prepared for visual and analytical comparisons.

Analysis Focus

- **Scalability** : Tracking how execution time increases with graph size.
- **Efficiency** : Understanding which algorithm handles certain graph types more effectively.
- **Behavioral Trends** : Identifying thresholds or graph characteristics that significantly affect runtime.

Results and Analysis

Both Tarjan-Vishkin and Slota-Madduri algorithms were executed sequentially across all graph datasets. For each run, the total runtime (excluding I/O) was recorded in seconds. The results are summarized as follows :

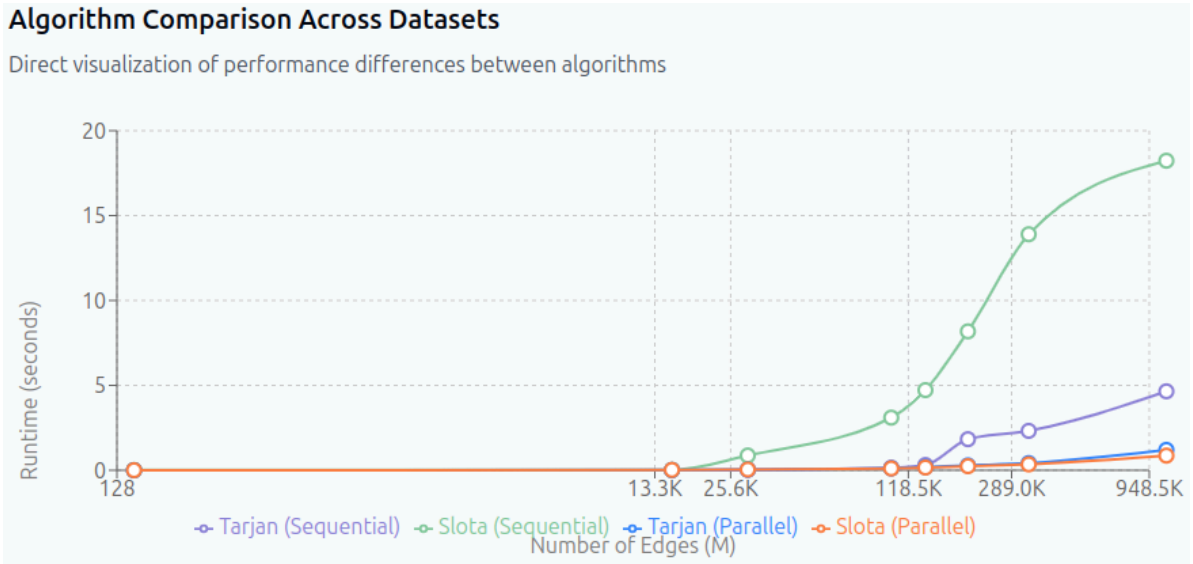
1. **On Sequential Setup**

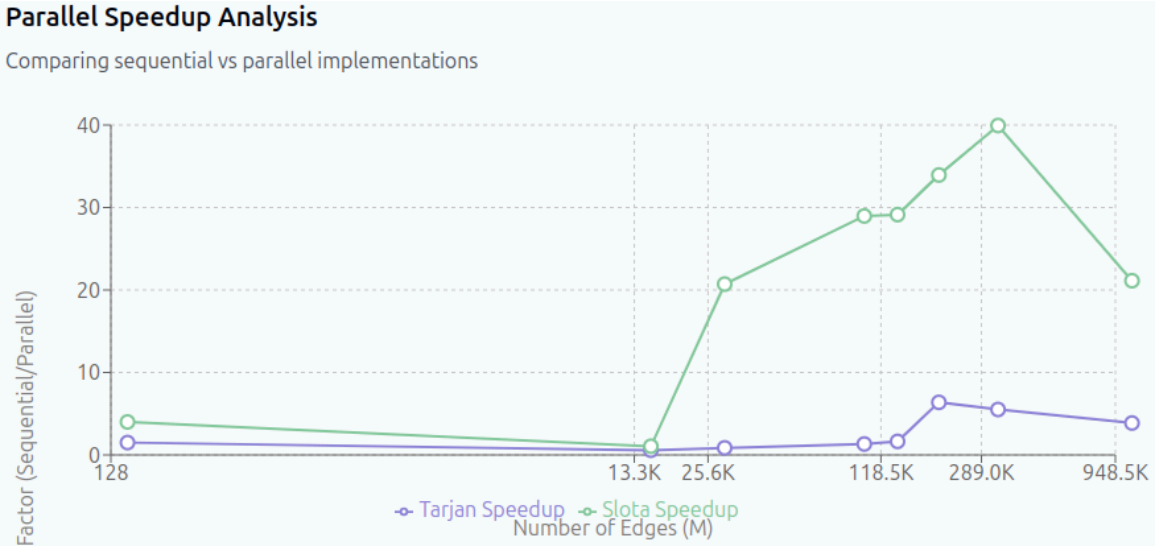
N	M	Tarjan	Slota
50	128	0.003s	0.012s
475	13,289	0.008s	0.02s
1,005	25,571	0.03s	0.87s
4,039	88,234	0.15s	3.1s
12,008	118,521	0.31s	4.72s
22,469	171,002	1.83s	8.18s
37,700	289,003	2.32s	13.9s
82,167	948,464	4.65s	18.23s

2. **On Parallel Setup**

N	M	Tarjan	Slota
50	128	0.002s	0.003s
475	13,289	0.014s	0.019s
1,005	25,571	0.035s	0.042s
4,039	88,234	0.113s	0.107s
12,008	118,521	0.189s	0.162s
22,469	171,002	0.287s	0.241s
37,700	289,003	0.421s	0.348s
82,167	948,464	1.201s	0.863s

Runtime Plots





Observed Trends

1. **Tarjan-Vishkin** : Consistent and Predictable Performance
    a. In both sequential and parallel executions, Tarjan-Vishkin exhibited a steady, linear growth in runtime as graph size increased.
    b. Sequential Tarjan outperformed all other configurations for graphs with up to ~100K edges, thanks to its low-overhead DFS traversal and efficient disjoint-set operations.
    c. In the parallel setup, while some improvements were observed, the gains were modest—suggesting that Tarjan's structure does not naturally lend itself to parallelism, especially without restructuring DFS.
2. **Slota-Madduri** : High Overhead, High Reward (When Parallelized)
    a. In sequential mode, Slota-Madduri showed consistently higher runtimes, owing to its BFS-based design and coordination-heavy structure, which incur extra cost when not parallelized.
    b. However, in the parallel setup, its performance dramatically improved, especially on graphs with more than ~100K edges.
    c. For the largest dataset (948K edges), parallel Slota achieved the best absolute runtime (0.863s), outperforming even the parallel Tarjan (1.201s).
    d. This confirms that Slota-Madduri is architected for parallel scalability, becoming increasingly efficient as graph density and size grow.
3. **Sequential vs. Parallel Performance Comparison**
    a. For small to medium graphs, the overhead introduced by parallel infrastructure (thread management, memory contention) is not offset by speed-up, making sequential algorithms more efficient.
    b. For large graphs (≥ 500K edges), parallelism starts to pay off—reducing runtimes significantly for Slota-Madduri, while Tarjan-Vishkin sees limited gains, hinting at the ceiling of its parallel scalability without algorithmic redesign.
    c. Parallel Slota shows superlinear scaling benefits in some large graphs, suggesting better resource utilization and reduced bottlenecks due to intelligent load distribution.
4. **Crossover Point & Density Sensitivity**
    a. The performance crossover point—where parallel Slota begins to outperform parallel Tarjan—typically occurs around 100K–200K edges, aligning with when graph density starts stressing single-threaded computation.

b. Beyond this point, Slota-Madduri's parallel granularity and BFS layer-wise processing begin to leverage the full bandwidth of multicore architectures.

Conclusion and Future Prospects

This study benchmarked the Tarjan-Vishkin and Slota-Madduri algorithms for identifying biconnected components across diverse graph datasets. The results clearly show that Tarjan-Vishkin consistently outperforms Slota-Madduri under sequential execution, owing to its lightweight DFS-based structure and low coordination overhead.

However, Slota-Madduri demonstrates significant performance gains in parallel settings, particularly on large and dense graphs. A crossover trend was observed around the 100K edge mark, where Slota-Madduri begins to surpass Tarjan-Vishkin in speed when run with multiple threads.

These findings reinforce that algorithm choice should depend on both graph characteristics and execution environment. While Tarjan-Vishkin is well-suited for fast sequential processing, Slota-Madduri holds promise in parallelized workloads. Future work could explore optimized parallel variants and test scalability on heterogeneous platforms like GPUs and distributed systems.

Despite the algorithms being evaluated sequentially, the findings provide a solid foundation for future enhancements that could significantly improve performance :

1. **Parallelisation** : Implementing parallel versions of both algorithms, particularly Slota-Madduri, could lead to substantial performance gains on larger graphs, leveraging multi-core and distributed computing environments.
2. **Hybrid Approaches** : A dynamic algorithm selection based on graph properties could optimize performance, combining the strengths of both algorithms depending on the graph's structure.
3. **Memory and Representation Optimizations** : As graphs continue to grow, focusing on memory-efficient algorithms and experimenting with alternative graph representations like adjacency matrices could improve scalability.
4. **Real-World Applications** : This research has practical implications for network analysis, social media, and cybersecurity, where efficient biconnectivity analysis is critical for detecting vulnerabilities and understanding complex structures.

In conclusion, while the current study sets the stage for further exploration, the next steps will involve parallelizing these algorithms, improving memory usage, and adapting them for large-scale, real-time applications. The results of this project will contribute to the ongoing development of more efficient and scalable graph algorithms in various real-world contexts.

Team BiConGraphs
Aviral Gupta (2023111023)
Sudheera YS (2023111002)