Practical Implications of BMSSP: Shortest Path Computation on Urban Road Networks

Ina Papadhopulli, Amina Sokoli

Polytechnic University of Tirana, Tirana, Albania

Abstract

The single-source shortest path problem has a new theoretical benchmark set by the Bounded Multi-Source Shortest Path algorithm with an asymptotic complexity of $O(m \log^{2/3} n)$. This work presents an empirical benchmarking of BMSSP against the classical Dijkstra's algorithm across diverse graph configurations. Despite its theoretical advantage, BMSSP underperforms Dijkstra by 7-10× on most small to medium-sized graphs. Our findings underscore the critical difference between asymptotic optimality and practical, real-world algorithm performance.

Keywords

Single-source shortest paths, Dijkstra, BMSSP, algorithm benchmarking, sparse graphs, dense graphs, asymptotic complexity

1. Introduction

The single-source shortest path problem is a cornerstone of graph theory, fundamental to applications ranging from network routing and logistics to operational research.[4] Dijkstra's algorithm, the classic solution for SSSP on graphs with non-negative edge weights, has served as the practical standard since 1959. When implemented efficiently, the algorithm solves the problem with a time complexity of $O((m+n)\log n)$ (where m signifies the number of edges and n the number of nodes in a given graph) [3]. Despite the efficiency of Dijkstra's algorithm, the theoretical limit of its time complexity has been a long-standing challenge known as the sorting barrier. This barrier was recently overcome by the introduction of the new deterministic SSSP algorithm, the Bounded Multi-Source Shortest Path [2]. This groundbreaking approach achieves an asymptotically time complexity of $O(m \log^{2/3} n)$ by employing a recursive partitioning strategy that avoids the complete sorting of all vertices by distance.

This work empirically investigates the question of whether this significant theoretical advantage translates into efficient performance gains in practical, real-world execution. To achieve this, our benchmarking uses a diverse set of graphs, including three real-world directed weighted road network graphs: a small graph representing Shkodra, Albania; a medium graph representing Tirana, Albania; and a dense graph representing Tokyo, Japan.

The structure of the paper is as follows: section 2 provides an overview of the presented approach; section 3 analyzes the results taken from the experimental runs of the two algorithms on the provided datasets; in the end, section 4 concludes with key findings.

2. Presented approach

To evaluate the practical performance of the BMSSP algorithm against Dijkstra's algorithm, we adopted an experimental simulation approach[5]. Both algorithms were implemented from scratch in C++ based on their formal definitions and pseudocode from the original sources [1, 2].

For benchmarking, both algorithms were executed under identical test environments. We used two categories of test graphs:

ICITEE25: 3rd International Conference on Information Technologies and Educational Engineering, December 18–19, 2025, Tirana, Albania
EMAIL: ipapadhopulli@fti.edu.al; amina.sokoli@fti.edu.al
ORCID: 0000-0003-2040-6860; 0009-0003-9183-6451;

- Synthetic graphs: small sized dense and sparse graphs, mostly used for correctness validation purposes. These graphs were randomly generated using a Python script based on different edge probabilities in a graph node [5];
- Real-world urban road networks: three edge-weighted digraphs that represent varied topologies of different cities.
  The datasets were extracted as raw data in `.xml` file format from OpenStreetMap and then converted and stored in a `.txt` file through a Python script as edge weighted digraphs in a `source,target,weight` file format [5], where `weight` is shown as the calculated distance in meters between two given points. This was done in order to eliminate unnecessary information from the original file and keep only nodes that represent buildings, hotels, bus stations and key locations.

Table 1
Benchmark graph datasets

Dataset	№ of vertices	№ of edges
Shkodra, Albania	1744	3523
Tirana, Albania	9433	18365
Tokyo, Japan	4968	10620

All edge weights were positive floating-point values. Each configuration was executed five times per algorithm, and average runtimes were computed for performance measurement.

3. Results and discussion

The empirical benchmarking reveals a significant disconnection between the BMSSP algorithm's promised asymptotic complexity and its practical performance on real-world datasets.

Across all tested urban road networks, BMSSP was substantially slower than Dijkstra. Specifically, on the Shkodra graph, BMSSP was approximately `10.8` times slower (`25.276 ms` vs `2.3312 ms`), and on the larger Tirana graph, it was over `15.7` times slower (`267.8895 ms` vs `17.00805 ms`). Even on the Tokyo graph, which represents a highly complex, large-scale network, BMSSP was still nearly `9.5` times slower (`96.16615 ms` vs `10.1334 ms`). Overall, the general runtime comparison showed BMSSP to be approximately `9.7` times slower than Dijkstra.

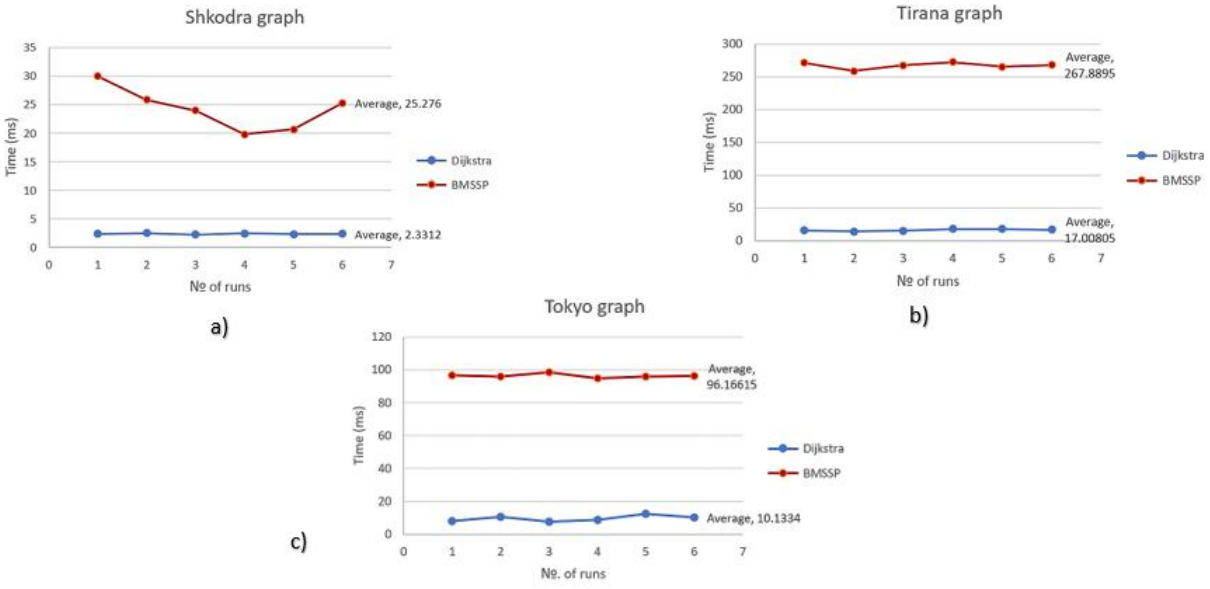

Figure 1: Runtime comparison of Dijkstra's and BMSSP algorithms on (a) Shkodra, (b) Tirana and (c) Tokyo graph urban road network.

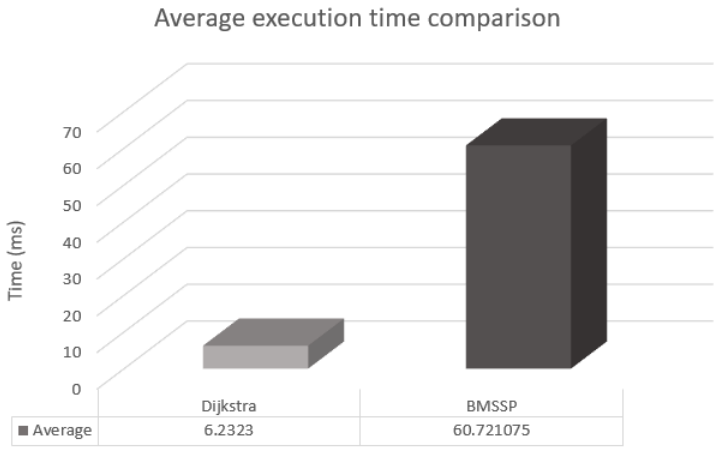

Figure 2: Algorithm average results

This practical sub-optimality is primarily rooted in the substantial constant factors that dominate BMSSP's execution time on graphs of practical size. The overhead stems from several implementation-specific challenges that are not reflected in the algorithm's big-O notation:

I. *The BMSSP algorithm involves too many operations and procedures.* [2, 5]
II. The paper of BMSSP explicitly mentions *a "large constant C"* that bounds all hidden costs.[2] This constant multiplies every operation throughout the execution, meaning that even though BMSSP has better asymptotic complexity, the actual number of primitive operations performed is substantially higher than what the big-O notation suggests.
III. *Recursion cost*
The BMSSP algorithm employs approximately `logn/t` levels of recursion. Each recursive level introduces function call overhead, parameter passing costs, as well as the computational expense of handling return values [2, 5].
IV. *Memory access patterns*
The BMSSP algorithm suffers from poor cache locality due to its complex data structures and recursive nature.
V. `FindPivots` *computational cost*
At each recursion level, the `findPivots` procedure [2, 5] must be executed which involves a `k` complete relaxation passes through the current frontier.

4. Conclusions

This empirical study reveals a fundamental truth in algorithm design that theoretical optimality does not always translate to practical superiority. While the BMSSP algorithm represents a significant theoretical breakthrough by breaking `O(m+nlogn)` sorting barrier, our experimental results demonstrate that Dijkstra's algorithm still remains the choice for practical applications. Across the test suites of both dense and sparse graphs at various scales, Dijkstra consistently outperformed BMSSP by factors ranging from `7× to 10×`.

The BMSSP algorithm's theoretical conclusion is undermined by substantial practical overhead. This because complex multi-level data structures, expensive pivot-finding procedures and poor cache locality all contribute to hidden constants that dominate performance on real-world problem instances.

This work proves that the gap between theoretical innovation and practical utility can be vast, and the classical algorithms that have stood the test of time often do so for compelling reasons based in real-world performance.

Declaration of Generative AI and AI-assisted Technologies in the Writing Process

During the preparation of this work, the authors used Grammarly/ChatGPT for language, grammar checks, and improvement. After using these services, the authors reviewed and edited the content as needed and take full responsibility for the content of the publication.

References

[1] R. Sedgewick and K. Wayne, Algorithms, 4th ed. Addison-Wesley Professional, 2011, ch. 4.4: Shortest Paths.

[2] R. Duan, J. Mao, X. Mao, X. Shu, and L. Yin, “Breaking the Sorting Barrier for Directed Single-Source Shortest Paths,” *arXiv preprint* arXiv:2504.17033, 2025

[3] E. W. Dijkstra, "A note on two problems in connexion with graphs," Numer. Math., vol. 1, pp. 269–271, 1959.

[4] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, Introduction to Algorithms, 3rd ed. MIT Press; McGraw-Hill, 2009, ch. 24: Single-Source Shortest Paths.

[5] Amina Sokoli, “bmssp”, Computer Software, 2025. Available: https://github.com/aminatpwk