Dijkstra's Algorithm: Performance Analysis with Different Heap Structures

B Pranav Karthik, Bharath Sooryaa M, Hari Karthik V, Prem N, Rohan Ramesh
Department of Artificial Intelligence, Amrita Vishwa Vidyapeetham

Abstract—This study evaluates the performance of Dijkstra's algorithm using five heap implementations: Binary Heap, AVL Tree, Binomial Heap, Fibonacci Heap, and Leftist Heap. Experiments were conducted on a large-scale graph with 264,346 vertices and 733,846 edges. Results show that Binary Heap outperforms others with an execution time of 2,021.53 ms, while Fibonacci Heap, despite its theoretical advantages, underperforms with 15,760.42 ms. The findings highlight the importance of practical efficiency over theoretical complexity in real-world applications.

Index Terms—Dijkstra's Algorithm, Binary Heap, Fibonacci Heap, AVL Tree, Binomial Heap, Leftist Heap, Performance Analysis

1 Introduction

Dijkstra's algorithm is a fundamental algorithm in graph theory, widely used for solving the single-source shortest path problem. Its efficiency depends heavily on the choice of the priority queue data structure. This study compares five heap implementations: Binary Heap, AVL Tree, Binomial Heap, Fibonacci Heap, and Leftist Heap, to evaluate their practical performance in Dijkstra's algorithm.

2 Dataset and Experimental Setup

2.1 Graph Characteristics

The dataset used in this study consists of:

- **Vertices:** 264,346
- **Edges:** 733,846 (bidirectional, weighted)
- **Edge Weights:** Randomly assigned (1-1000)

2.2 Heap Implementations

The following heap structures were implemented and tested:

- **Binary Heap:** Array-based, $O(\log N)$ for all operations.
- **AVL Tree:** Self-balancing binary search tree, $O(\log N)$ for all operations.
- **Binomial Heap:** Tree-based, efficient merging, $O(\log N)$ for most operations.
- **Fibonacci Heap:** Amortized $O(1)$ for insertion and decrease-key, $O(\log N)$ for extraction.
- **Leftist Heap:** Merge-optimized binary heap, $O(\log N)$ for all operations.

3 Results and Performance Analysis

3.1 Theoretical Time Complexities

TABLE 1
Theoretical Time Complexities for Priority Queue Operations

Heap Type	Insertion	Extract-Min	Decrease-Key
Binary Heap	$O(\log N)$	$O(\log N)$	$O(\log N)$
AVL Tree	$O(\log N)$	$O(\log N)$	$O(\log N)$
Binomial Heap	$O(\log N)$	$O(\log N)$	$O(\log N)$
Fibonacci Heap	$O(1)$ (amortized)	$O(\log N)$ (amortized)	$O(1)$ (amortized)
Leftist Heap	$O(\log N)$	$O(\log N)$	$O(\log N)$

3.2 Execution Time Comparison

TABLE 2
Execution Time of Dijkstra's Algorithm

Heap Type	Execution Time (ms)
Binary Heap	2,021.53
Leftist Heap	2,212.42
Binomial Heap	3,977.75
AVL Tree	5,046.97
Fibonacci Heap	15,760.42

4 Conclusion

The study demonstrates that Binary Heap is the most efficient choice for Dijkstra's algorithm in practice, despite its $O(\log N)$ time complexity for all operations. Fibonacci Heap, while theoretically optimal, suffers from significant practical overhead. Future work could explore hybrid heap structures or optimizations to reduce implementation complexity.

References

[1] R. Lewis, "A Comparison of Dijkstra's Algorithm Using Fibonacci Heaps, Binary Heaps, and Self-Balancing Binary Trees," 2023.
[2] M. Thakral and R. Dheemanth, "Practical Performance of Dijkstra's Algorithm: A Heap Comparison," *Journal of Computer Science*, vol. 59, pp. 72-85, 2022.
[3] T. H. Cormen et al., *Introduction to Algorithms*, 4th ed., MIT Press, 2022.