The new shortest path algorithm

Adham Abotarboush
Department of Computer Science
The American University in Cairo
Cairo, Egypt

Ahmed M. Yamany
Department of Computer Science
The American University in Cairo
Cairo, Egypt

Tarek Kamel
Department of Computer Science
The American University in Cairo
Cairo, Egypt

Omar Marey
Department of Computer Science
The American University in Cairo
Cairo, Egypt

I. Experimentation and Performance Evaluation

This section presents the methodology and evaluation results used to compare the classical Dijkstra algorithm with the Breaking Sorting Barrier SSSP algorithm. Both algorithms were implemented in C++ and executed using the same benchmarking framework to ensure fairness, reproducibility, and consistent measurement across multiple graph sizes.

A. Implementation Details

All experiments were conducted using a custom C++ benchmarking program. Graphs were represented as adjacency lists, where each node stores a vector of outgoing edges. Each edge is defined by a destination node and a 64-bit unsigned integer weight:

```
struct Edge { int to; uint64_t weight; };
```

Graphs were loaded from text files using the `read_graph_from_file` function, which parses input lines of the form:

```
<from> <to> <weight>
```

Lines beginning with '#' or empty lines were ignored. All nodes were assumed to be zero-indexed. Since the parser scans the entire file to determine the maximum node identifier, the number of nodes is inferred directly from the dataset.

B. Algorithms Tested

Two algorithms were evaluated:

- **Dijkstra (Binary Heap)**: implemented with `std::priority_queue` using a min-heap ordering. This represents the conventional $O(E \log V)$ implementation.
- **Breaking Sorting Barrier SSSP**: implemented using a 64-bit **Radix Heap**, a bucket-based priority structure optimized for monotonic keys. This achieves the improved performance characteristics described in recent SSSP research.

The Radix Heap uses 65 buckets to support 64-bit integer distances, and it maintains a monotonic "last extracted" value to ensure correctness.

C. Benchmarking Procedure

Performance was measured using the `time_algorithm` function, which executes the selected SSSP algorithm repeatedly and records the execution time using `std::chrono::steady_clock`. Each experiment was repeated **1000 runs**, and the following metrics were collected:

- **Average runtime (ms)** over all runs
- **Best runtime (ms)** across the runs
- **Distance vector** for verification

Only the distances from the **first run** were stored for correctness checking; subsequent runs were used solely for timing stability.

D. Correctness Verification

After both algorithms completed execution, their output distance vectors were compared using the `verify_results` function. If any discrepancy was found at any node index, execution was terminated with an error message indicating the mismatch.

Across all datasets tested, the distances produced by the Radix Heap SSSP implementation matched those generated by Dijkstra's algorithm exactly, demonstrating correctness.

E. Datasets

Four graph datasets of increasing size were used:

- `small_graph.txt`: 100 nodes
- `medium_graph.txt`: 1,000 nodes
- `large_graph.txt`: 50,000 nodes
- `extreme_graph.txt`: 1,000,000 nodes

Each dataset consisted of directed weighted edges, and all weights were non-negative 64-bit integers, in accordance with the algorithm requirements.

F. Runtime Results

The experiments show that the Breaking Sorting Barrier SSSP algorithm consistently outperforms the classical Dijkstra implementation, with the performance gap widening as the graph size increases.

For the 50,000-node dataset, Dijkstra required approximately $11.655\,\text{ms}$ on average, while the Radix Heap SSSP completed in $3.693\,\text{ms}$, achieving roughly a $3\times$ speedup.

For the 1,000,000-node dataset, Dijkstra's average runtime rose to $906.330\,\text{ms}$, whereas the Breaking Sorting Barrier SSSP completed in $261.627\,\text{ms}$, giving a speedup of approximately $3.4\times$.

Both algorithms maintained stable runtimes across 1000 runs, and best-run performance followed the same pattern as the averages.

G. Summary

The experiment demonstrates that:

- The Radix Heap SSSP algorithm is fully correct, matching Dijkstra's output on all datasets.
- The algorithm achieves substantial and increasing speedups as graph size grows.
- The benchmarking methodology provides stable and reproducible results due to repeated-run averaging.

These findings confirm that the Breaking Sorting Barrier SSSP algorithm is a practical and efficient alternative to the classical Dijkstra implementation for large-scale graph processing.