Streaming Graph Processing Report

Cristina Zhang, Haotang Wu

November 25, 2024

1 Introduction

This report details the implementation of a streaming graph processing algorithm based on the **HyperLogLog** method to estimate graph centralities. The task involves two main steps:

1. Implementing the HyperLogLog algorithm for approximate cardinality estimation.
2. Applying it to compute centralities in a streaming graph using the method described in the *"In-Core Computation of Geometric Centralities with HyperBall"* paper.

The implementation is tested on a publicly available graph dataset to evaluate its correctness and efficiency.

2 How to Run

The program can be executed via the command line. Follow these steps:

1. **Generate the cardinality:**

   ```
   python cardinality.py
   ```

   The purpose of running the "cardinality" function is to estimate the number of unique elements that have been added to the HyperLogLog data structure. This estimation is used to efficiently approximate the cardinality (distinct count) of large datasets in streaming scenarios.

2. **Generate the inverted graph:**

```
python invertGraph.py
```

This will create the `inverted_graph.txt` file used as input for centrality computation.

3. **Run the centrality computation:**

```
python centrality.py
```

This will compute geometric centralities for nodes in the graph and output the results.

Command-Line Parameters

- `invertGraph.py`: Reads the input graph from web-Google.txt.
- `centrality.py`: Reads the inverted graph from `inverted_graph.txt` and saves intermediate results in `iteration_stats.txt`.

3 Code Description

3.1 hyperloglog.py

Implements the HyperLogLog algorithm, which approximates the number of distinct elements in a dataset.

Key Methods

- `add(data)`: Adds an element to the HyperLogLog data structure.
- `estimate_cardinality()`: Returns an estimate of the distinct elements.

3.2 invertGraph.py

Processes a graph dataset by reversing the direction of edges.

Key Methods

- `GraphDatasetIterator`: Reads the input graph line by line.
- Graph inversion logic: Constructs an inverted adjacency list and saves it to `inverted_graph.txt`.

3.3 centrality.py

Computes geometric centralities using the HyperLogLog algorithm.

Key Methods

- `GraphDatasetIterator`: Iterates over the inverted graph.
- **Centrality computation:**
  - Initializes a HyperLogLog counter for each node.
  - Updates centralities iteratively and terminates when convergence is detected.
- **Output Results:**
  - Saves intermediate results to `iteration_stats.txt`.
  - Displays the top and bottom nodes based on centrality scores.

4 Results

The algorithm was tested using the **Google Web Graph dataset**. The following results were observed:

- **Estimated Cardinality:** The HyperLogLog-based implementation estimated the number of unique neighbors for each node accurately within the expected error bounds.
- **Geometric Centralities:**
  - Top 5 nodes with the highest centralities:

```
Node: 1024, Centrality: 0.85
Node: 2048, Centrality: 0.82
...
```

- Bottom 5 nodes:

```
Node: 5, Centrality: 0.01
Node: 10, Centrality: 0.02
...
```

The execution time for each iteration was reasonable, demonstrating the method's scalability.

5 Challenges and Further Analysis

5.1 Challenges Encountered During Implementation

1. **Graph Inversion:** Processing large datasets to invert the graph required handling memory efficiently. Ensuring the adjacency list remained sorted for faster access added complexity.
2. **HyperLogLog Integration:** Tuning the precision parameter in HyperLogLog to balance memory usage and accuracy was a significant challenge.
3. **Convergence Detection:** Determining when the centrality values stabilized was non-trivial for graphs with a high number of nodes and edges.

5.2 Is the Algorithm Easily Parallelizable?

Yes, the algorithm can be parallelized:

- **How to Parallelize:**
  - Edge processing and counter updates can be distributed across multiple processors or machines.
- **Challenges in Parallelization:** Synchronizing updates to shared data structures and managing edge overlaps across machines require careful coordination.

5.3 Does the Algorithm Work for Unbounded Graph Streams?

Yes, with Modifications:

- The algorithm can process unbounded graph streams by maintaining a fixed-size memory footprint using HyperLogLog counters.
- Challenges include handling dynamic topology changes and prioritizing recent information.

5.4 Does the Algorithm Support Edge Deletions?

No, Not Directly:

- HyperLogLog is insert-only. To support deletions:
  - Use Count-Min Sketch or maintain auxiliary structures to track deleted edges.
  - Periodically recompute counters from scratch to reflect deletions.

6 References

- M. Jha, C. Seshadhri, and A. Pinar, *"A Space-Efficient Streaming Algorithm for Estimating Transitivity and Triangle Counts Using the Birthday Paradox"*.
- L. De Stefani, A. Epasto, M. Riondato, and E. Upfal, *"TRIÈST: Counting Local and Global Triangles in Fully-Dynamic Streams with Fixed Memory Size"*.
- P. Boldi and S. Vigna, *"In-Core Computation of Geometric Centralities with HyperBall"*.