Implementation Progress Report: MAX-ID Algorithm for Finding Maximum Node ID in Trees

Algorithms and Design Analysis

April 20, 2025

Implementation Summary

This project implements the MAX-ID algorithm as introduced in the paper *"Optimal Deterministic Massively Parallel Connectivity on Forests"* by Balliu et al. The goal of the algorithm is to determine the maximum node ID in a tree through a parallel and memory-efficient approach.

The implementation includes the entire MAX-ID algorithm:

- Light and heavy node classification.
- Subtree probing and controlled exponentiation.
- CompressLightSubTrees and CompressPaths operations.
- The main solve loop reduces the graph through phases.

All components have been implemented and thoroughly tested on three classes of trees: a path graph, a balanced binary tree, and a mixed star-like tree.

Correctness Testing

Correctness was verified through structured test cases:

- **Path Graph:** Ten nodes with increasing IDs tested to ensure the algorithm finds the terminal node as the max.
- **Balanced Tree:** A binary tree where IDs increase with level; the leaf with the highest value is expected.
- **Mixed Tree:** A star graph with deep branches tested the ability to find the maximum across varied subtrees.

Edge cases such as trees with a single node or only two nodes were also tested. Visualizations before and after the execution confirmed the correct behavior.

Complexity & Runtime Analysis

Theoretically, the algorithm runs in $O(\log \hat{D})$ rounds, where $\hat{D}$ is an upper bound on the diameter. Empirically, the runtime scaled well with small and medium-sized trees. The complexity they proposed in the start of the paper is guaranteed to work in $O(\log \hat{D})$ iterations for each subroutine/function in the MAX-ID solver, and we tried to run it for that many iterations at max(which is a naive approach).

No significant bottlenecks were encountered in small tests, though for larger graphs memory consumption grows due to the symmetric view propagation (doubling of state information). The `probe_directions` and `exponentiate` functions dominate runtime.

We could not confirm the algorithm's complexity that was proposed due to our constraints of not being able to mimic an MPC model although we did try using Dask.

Baseline or Comparative Evaluation

Can not be compared to any previous implementation, as there is none. The paper introduces a theoretical framework and provides justification for it only.

Ou

Challenges & Solutions

Key challenges included:

- **Memory-bound View Expansion:** Resolved via directional probing and excluding heavy directions from exponentiation.
- **Testing on Diverse Structures:** Addressed with helper graph generators and visualizations.
- **No Previous Implementations** There no paper or repository that has implemented anything related to MPC models.
- **MPC model:** The computational is completely different from normal parallel processing that we proposed with Dask in Python. We tried implementing however we ran several problems and as of yet of no solutions.

Enhancements

Modified Algorithm

We included a node state classification into `happy`, `sad`, `full`, and `active` to improve phase termination logic and memory safety(in accordance to memory constraints).

Additional Datasets

The algorithm was tested on graph topologies not covered in the original paper to verified correctness.

Other Enhancements

We added colored visualization of nodes pre- and post-compression to analyze convergence behavior. This was crucial in debugging sub tree compression and view expansion logic.