MPI Project Report: Single Source Shortest Paths (SSSP)

Maria Wysogląd (mw431552)
m.wysoglad@student.uw.edu.pl

June 10, 2025

Contents

1 Introduction **2**

2 Implemented optimizations **2**

3 Experiments **2**

4 Results **2**

5 Scaling **8**

6 Remarks **9**

1 Introduction

The aim of this project is to implement the distributed Δ-stepping algorithm for solving the Single Source Shortest Paths (SSSP) problem using MPI. This algorithm balances between Dijkstra and Bellman-Ford, offering a parallelizable and efficient approach. We further integrate and evaluate multiple optimization strategies to enhance performance.

2 Implemented optimizations

I implemented the following:

- Edge classification (IOS heuristic)
- Pruning with Push vs. Pull heuristic
- Hybridization

All optimizations follow the methods described in the paper:

Parallel Δ-stepping for SSSP on Distributed Memory Systems (Buluç et al.)

3 Experiments

I tried to generate testing graphs as described in paper. I generated tests with 2^{10} to 2^{15} vertices and edge factor 16. In hybridization, I used $\tau = 0.4$, as described in paper.

I tested the following configurations:

- BASIC-EXP: Simple Δ-stepping algorithm without other optimizations.
- IOS-EXP: Implemented IOS heuristic.
- PRUNING-EXP: Implemented pull model; decision process between pull vs. pull; it was performed on edges classified with edge classification, but without full IOS.
- HYBRID-EXP: Implemented collecting all vertices after reaching certain threshold and performing Bellman-Ford algorithm on the remaining vertices; it was performed on edges classified with edge classification, but without full IOS.
- OPT-EXP: Optimal code according to the paper, with IOS, pruning and hybridization.

In my experiments, I used delta=[1, 10, 25, 40, 80].

I run my tests on various number of workers n-workers=[24, 48, 96].

To measure performance, I calculated $TEPS = \frac{\text{traversed-edges}}{\text{time[s]}}$.

4 Results

Here you can see the obtained results of the experiments. OPT-EXP performed best on rmat2 while PRUNING-EXP on rmat1.

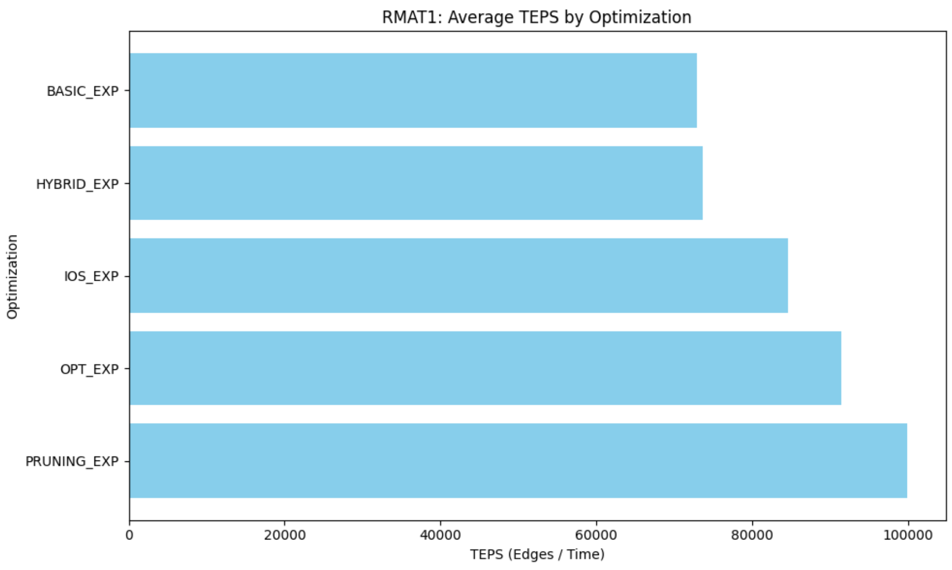

Figure 1

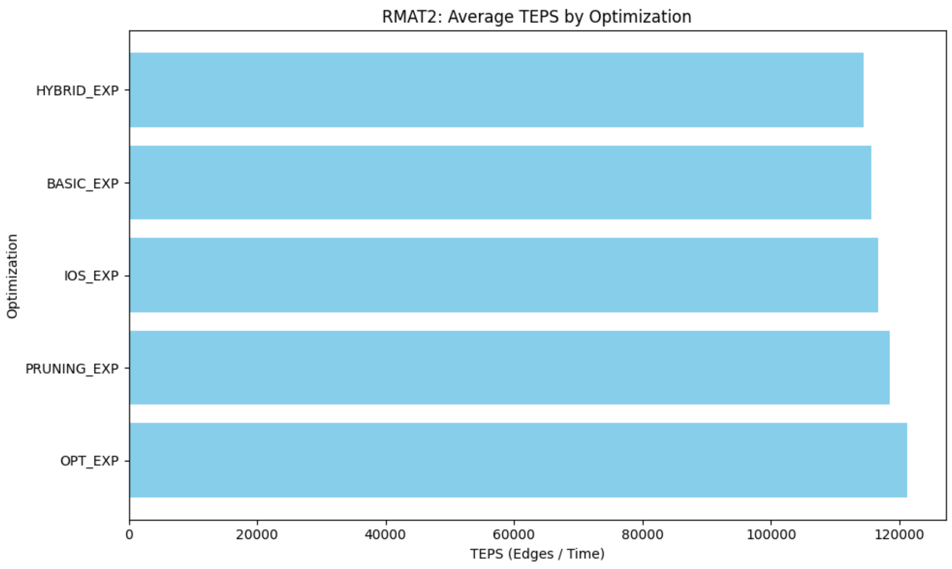

Figure 2

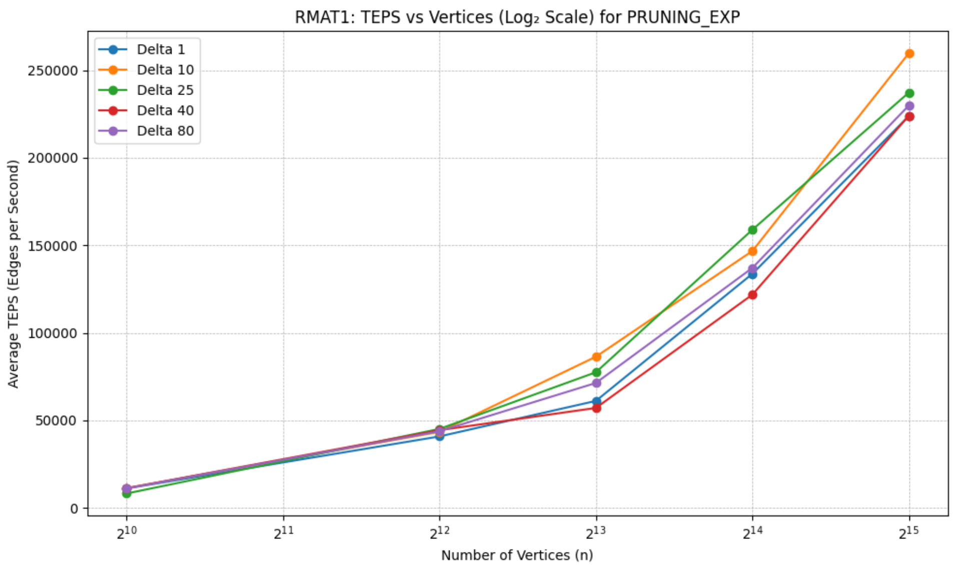

Figure 3

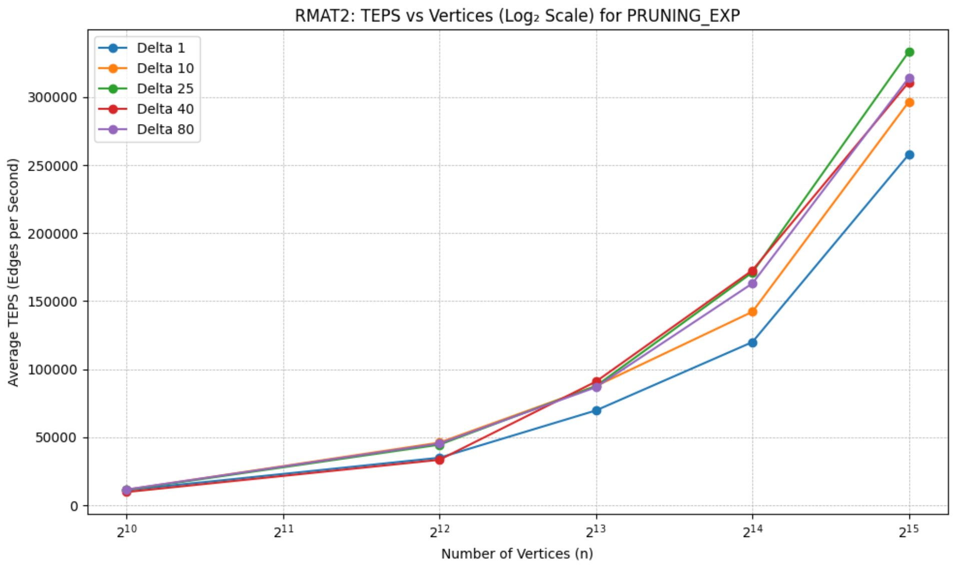

Figure 4

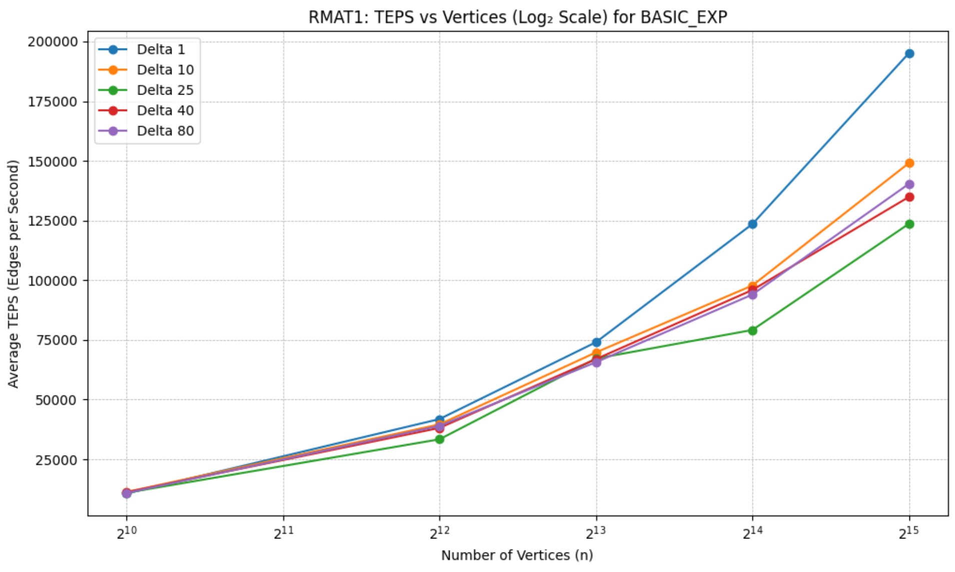

Figure 5

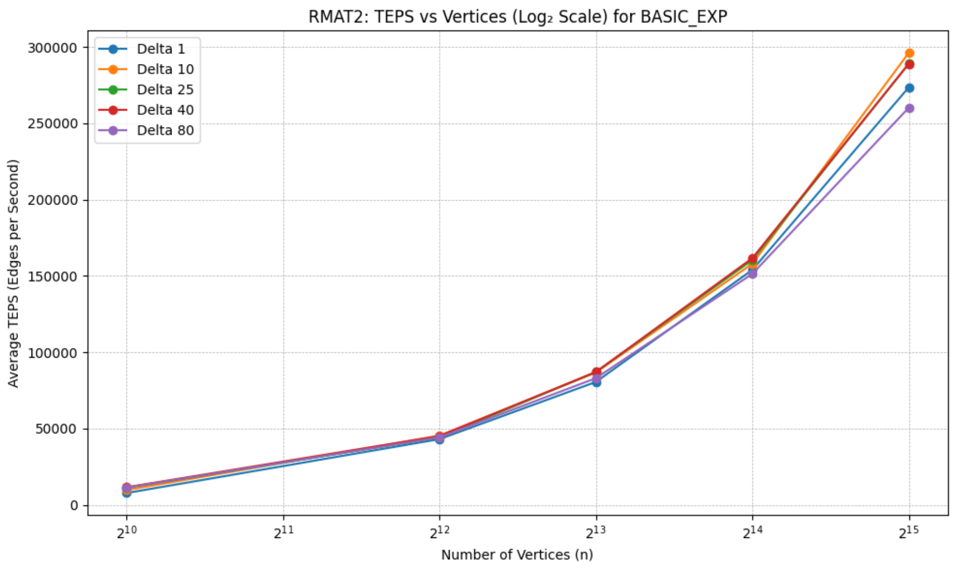

Figure 6

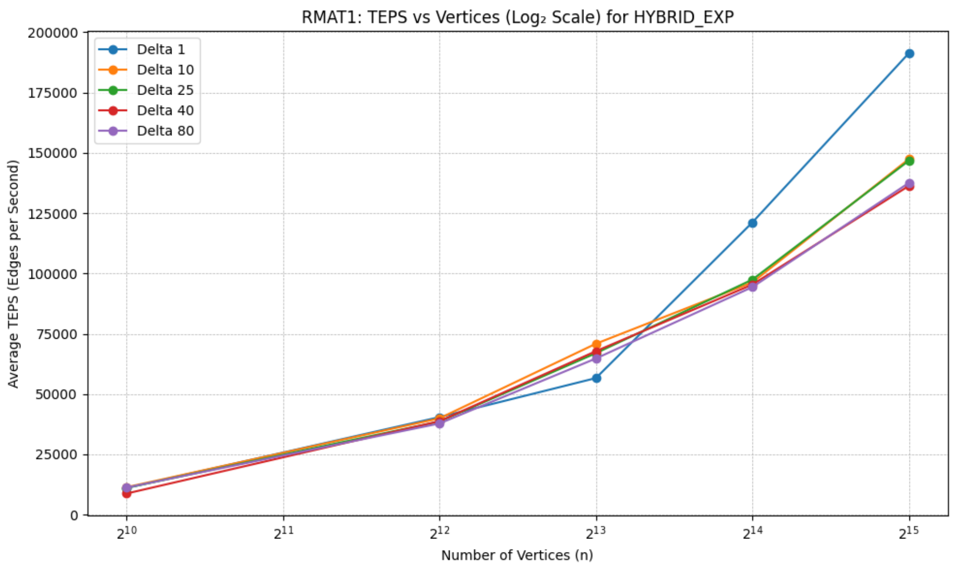

Figure 7

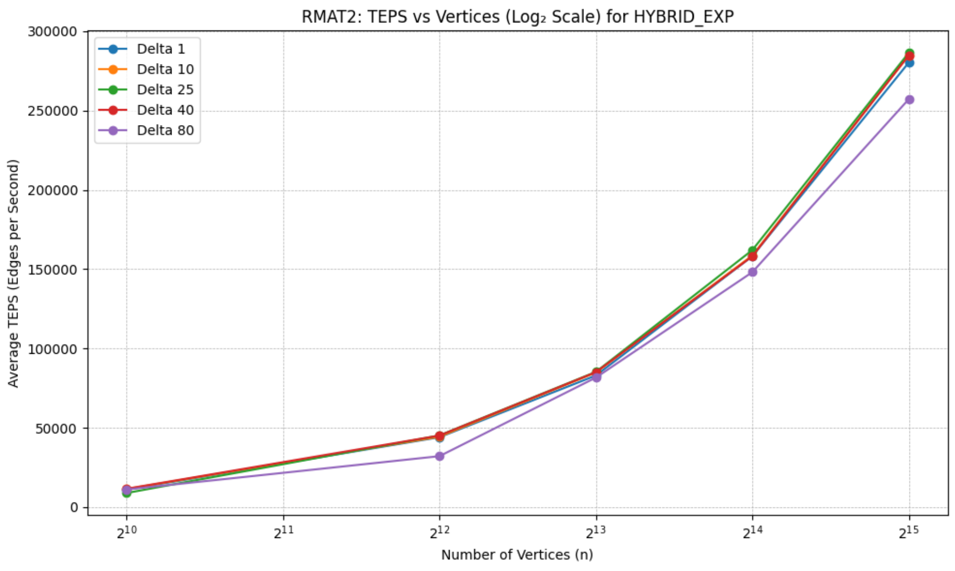

Figure 8

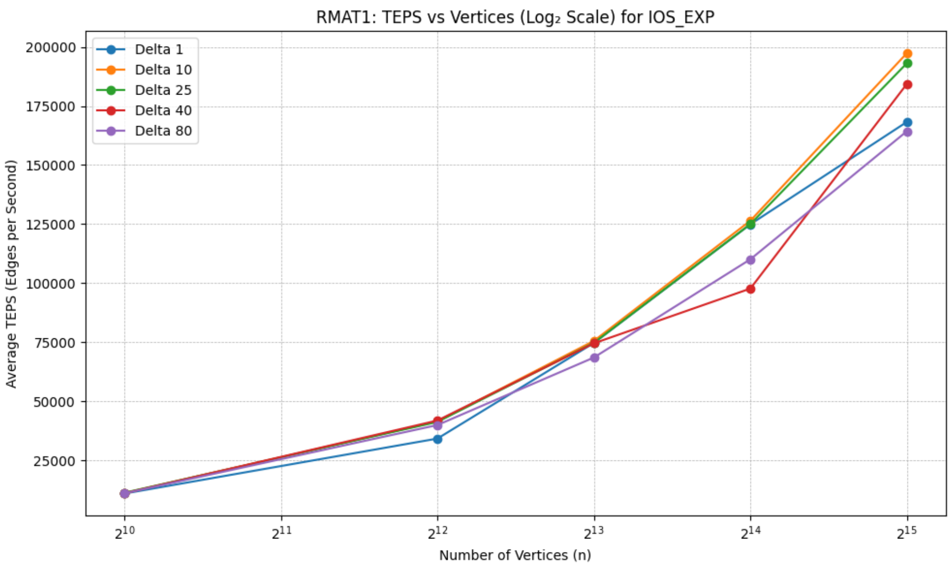

Figure 9

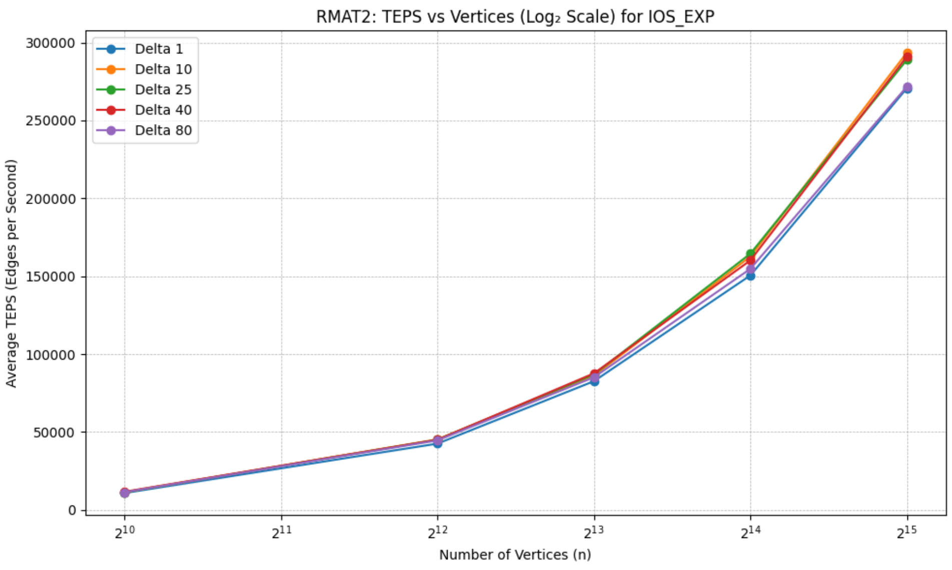

Figure 10

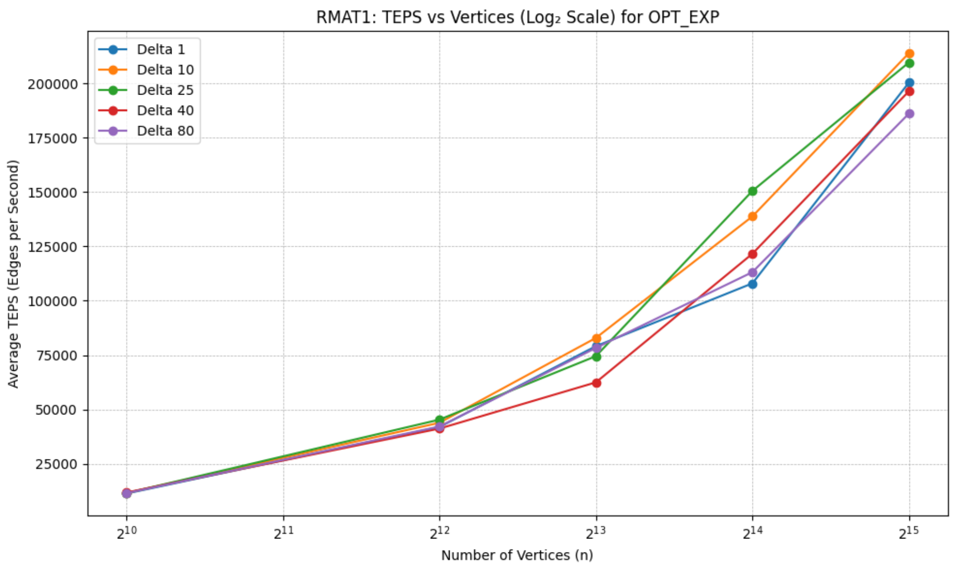

Figure 11

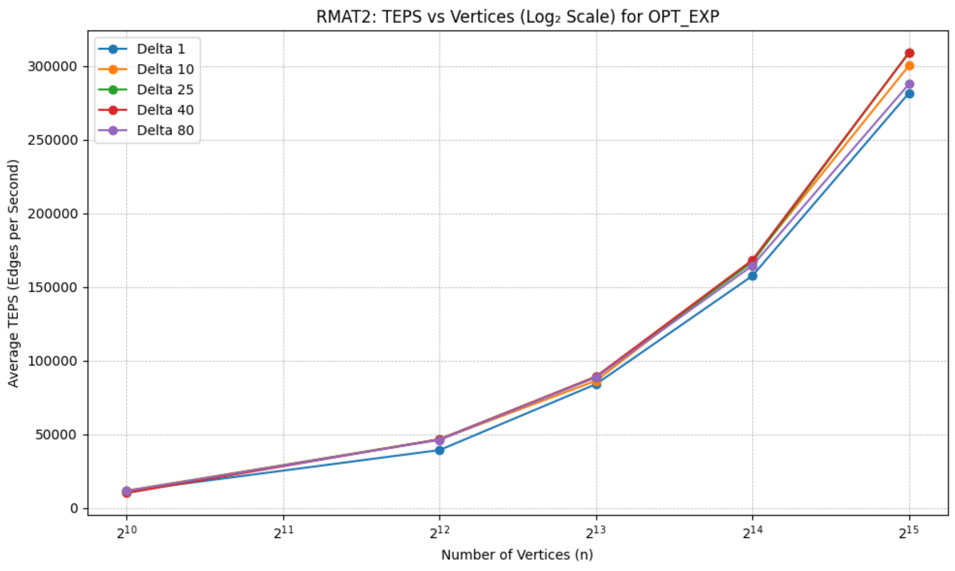

Figure 12

Best results for OPT-EXP and PRUNING-EXP were obtained with circa $\Delta = 25$ and it seems to be consistent with paper results.

5 Scaling

Unfortunately, due to the fact that I did not implement load balancing, the weak scaling criterion was not fully satisfied. I calculated scaling factor as $\frac{\text{number-of-vertices}}{\text{number-of-workers}}$.

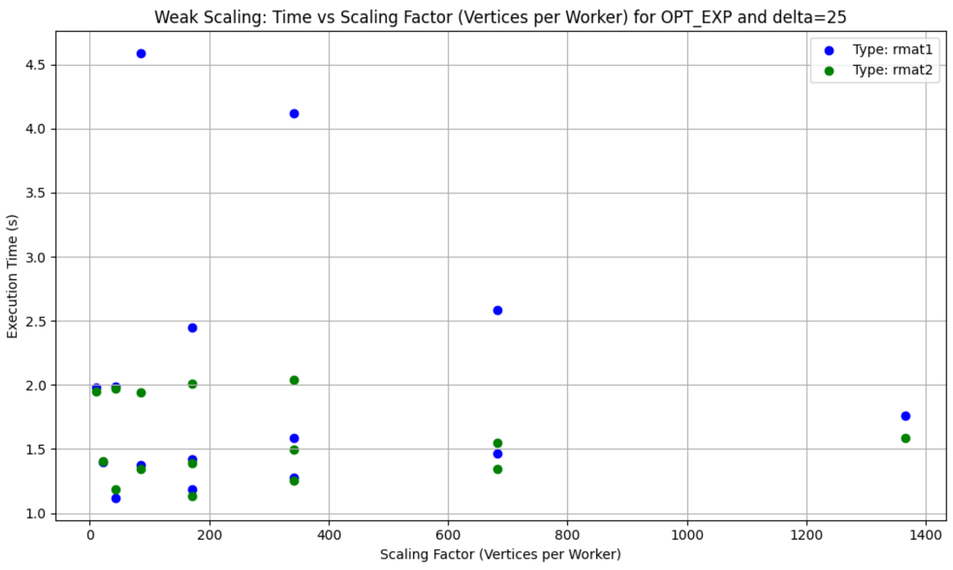

Figure 13

Here you can see the time per test considering delta=25 and OPT-EXP. In weak scaling, it should be expected that the dots (times) for given scaling factor should be close to one another, but unfortunately it is not the case here. We can see that scaling is better for rmat2 than rmat1 though.

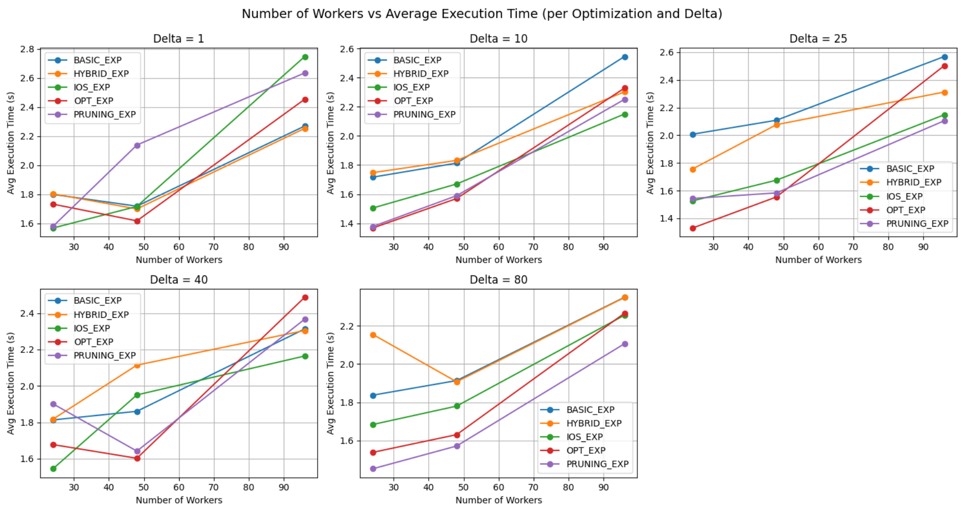

Figure 14

On this chart we can see the problems related to the need of communication - as it takes resources and may lead to poorer performance. It is not always the case though - optimization and optimal division of vertices might lead to even shorter execution time in some cases.

6 Remarks

While working on this report as well as solution, I was using ChatGPT and Github Copilot.