**Meeting Notes 08-02-2022**

**Progress**:  
I Implemented three different graph edit distance algorithms. The exact algorithm is far too slow to even calculate the edit distance for the trace graphs produced by the smallest test harness. After this I tried approximate GED algorithms. I implemented assignment edit distance AED, which runs in cubic time and hausdorff edit distance, runs in quadratic time. AED overestimates the distance, while HED underestimates the distance. Since both algorithms approximate and I am only interested in the relative similarity between trace graphs produced by different delays, I decided to go with HED for its improved speed.

To verify that similar delays result in similar trace graphs and vice versa, I computed the similarity of different trace graphs. The similarity is calculated as 1 – (GED/Max(GED)), where GED is HED and Max(GED) = |V\_max| + |E\_1| + |E\_2|, where |V\_max| is the number of nodes in the graph with the most nodes and |E\_i| is the number of edges in graph i.

In the experiment I compared 2 traces from 0 delays, 2 traces from 1 seconds delays and 2 traces from random delays between 0 and 1 seconds. The trace graphs include send events for smaller graphs. The expected outcome is that traces from the same delays are more similar than traces from different delays. To be more precise, the similarity between the 0 delay and 1 second delays should be higher than the similarity between the 0 delay and random delay, because the relative time each message gets delayed is still the same. Therefore, the causal dependencies should only change because of internal events in the system.

Use only receive events in trace graphs

Paper on Mattermost  
How much does changing the delay influence the execution. Is it the same as dropping? This can be used to decrease the size of the trace graph.

Logit normalization / minmax normalization. Better to differentiate the different traces / use in the EA

Look at the influence of what changing one gene has on the distance of traces. Look at what strategy is the most fruitful one to improve diversity.

1. Take 1000 random tests and increase diversity, see if bugs appear (naïve)
2. Take a couple random test and start intelligently mutating these tests to increase diversity

Stay away from population-based algorithms at first, start by experimenting with hill-climber type algorithms first. The simpler the algorithm the better.