Parallel Algorithm for Latent Hierarchical Causal Structure Discovery with Rank Constraints

Yiyang Bi **Ingvil Ahlsand**

Abstract

The purpose of this study is to address the inefficiency of the algorithm presented in the paper. The algorithm proposed in the paper aims to identify the causal structure of a system consisting of latent variables that form a hierarchical graph structure, generating the measured variables. While the current algorithm successfully locates latent variables, determines their cardinalities, and identifies the latent hierarchical structure, its efficiency is limited. It struggles to handle large datasets and requires a considerable amount of time to identify the causal structure. To overcome these challenges, we are inspired by a Horizontal Parallelization algorithm called cuPC. Our approach leverages the power of Numba and CUDA to significantly enhance the efficiency of the algorithm. By parallelizing the computations, we expedite the process of identifying the causal structure, enabling faster and more scalable analysis of large datasets.

1 Introduction

Many causal discovery algorithms assume their inputs have no latent confounders, whereas, in real-world applications, this assumption is almost impossible to hold and ignoring the latent variables is likely to lead to biased conclusions. Thus, over the past few decades, much effort has been made on exploring the latent structure behind measured variables. One such method Huang, et. al [2] proposed is an efficient algorithm that identifies the latent hierarchical structure for data with linear causal relationships. We aim to reduce the computation time required for the algorithm and make it feasible to analyze larger datasets by implementing a parallel algorithm.

2 Related Work

2.1 Latent Hierarchical Causal Discovery Structure

The method we use to determine the casual structure with latent confounder is enlightened by Huang et al [2]. It is based on two conditions : (1) The graph is Irreducible Linear Latent Hierarchical (IL2H) (2) The rank faithfulness assumption. Now, we want to briefly introduce the procedure.

2.1.1 Phase I: Finding Causal Clusters

The essential idea of this stage is to test for rank deficiency recursively to discover clusters of variables and their latent atomic clusters. First, we will have a set of variable X_G which is the set of measured variable. Then, we will let set $S = X_G$. We start to identify the latent atomic cover with size k = 1. We consider any subset of the latent atomic covers in S and replace them with their pure children, resulting in $S^{'}$. Then we will consider all possible subset $A \subset S'$ with the set size is at least k + 1. By conducting a rank deficiency test of X_A and X_B where X_A and X_B are the measured pure descendants of A and B, respectively, in the currently learned graph. By doing this we will know X_A and X_B are d-separated by a set with size k. We repeated the step on every possible subset A. It is

37th Conference on Neural Information Processing Systems (NeurIPS 2023).

Algorithm 1 The first step in PC-stable algorithm.

Input: $\mathcal{V}$
Output: G, $SepSet$
1: $G =$ fully connected graph
2: $SepSet = \emptyset$
3: $\ell = 0$
4: **repeat**
5: Copy G into G'
6: **for** any edge (V_i, V_j) in G **do**
7: **repeat**
8: Choose a new $S \subseteq adj(V_i, G') \backslash \{V_j\}$ with $|S| = \ell$
9: Perform $I(V_i, V_j | S)$
10: **if** $V_i \perp\!\!\!\perp V_j | S$ **then**
11: Remove (V_i, V_j) from G
12: Store S in $SepSet$
13: **end if**
14: **until** (V_i, V_j) is removed or all sets S are considered
15: **end for**
16: $\ell = \ell + 1$
17: **until** (max degree $-1 \geq \ell$)

Figure 1: First step of PC algorithm

Algorithm 2 Overall view of the proposed solution. Lines 7, 9, and 10 are executed in parallel on GPU.

Input: $\mathcal{V}$
Output: G, $SepSet$
1: $G =$ fully connected graph
2: $SepSet = \emptyset$
3: $\ell = 0$
4: $A_G =$ adjacency matrix of graph G
5: **repeat**
6: **if** $(\ell == 0)$ **then**
7: GPU: execute level zero
8: **else**
9: GPU: compact A_G into A'_G
10: GPU: execute level ℓ
11: **end if**
12: $\ell = \ell + 1$
13: **until** (max degree $-1 \geq \ell$)

Figure 2: Structure of first step in cuPC

possible that, during the searching process, we can combine two subset to create a larger latent cover. Hence, we could combine latent cover that where the intersection of pure children is not empty. If we did this, we need to reset k back to 1. If no latent cover is found at a certain k, we just increase the size of k. This procedure is repeated until no more clusters are found. We assign edges to the graph in the end following the independence relationship we get from the rank deficiency test.

2.1.2 Phase II: Refining Clusters

The above phase I algorithm may not help us correctly identify the casual structure because the existence of bond set. Fortunately, the bonded set could be removed and we use its children to connect with the siblings and grandparents. Since it is no way that we could identify a bond set beforehand without knowing the true graph, we decide a root node and perform a breadth-first search from the root variable. The algorithm will ends after refining every latent cover in G.

2.1.3 Phase III: Refining Edges

Then last phase is somewhat similar to the PC algorithm. We start with a fully connected latent covers. Then, we perform so called Cross Cover Test on every pair of latent cover. It there is rank deficiency, it means there is a d-separation so we will remove the edge between two latent covers. Then, we need to decide the v-structure. Note that we won't have a v-structure among a latent cover with its children and parents as this would generate any rank deficiency. Finally, we will apply Meek's rule to furnish more certain directions.

2.2 Parallel PC Algorithm

As described above, the latent hierarchical discovery algorithm screens through clusters of variables recursively for their relationships, which is structurally similar to PC algorithm. Thus, we believe the algorithm can be potentially improved by borrowing the idea of parallel PC algorithm. Madsen, et al. [3] offers two parallelizations for PC algorithms: a vertical parallelization which is based on Balanced Incomplete Block (BIB) designs for marginal independence testing; a horizontal parallelization which assigns the same number of cases to each thread. Similarly, Zarebavani, et al. [4] introduces a CUDA-based CPU acceleration for PC-algorithm, cuPC, and its two variants, cuPC-E and cuPC-S. These variants can significantly increase the efficiency with the potential to speedup by up to 1300 times on average compared to sequential implementation. In this report, we will mainly focus on the idea of cuPC algorithm.

Before going into details about parallel PC, it is nice to briefly talk about PC algorithm. As an algorithm in Causal Discovery Learning, PC algorithm means to restore the Markov equivalent class based on data generated by those variables. PC algorithm has mainly two steps: 1. Skeleton search by conducting Conditional Independence tests on every pair of variables, where a pseudo-code is shown in Figure 1. 2. Find v-structure and apply Meek's Rule to determine the direction of edges.

According to previous work, the second step of PC algorithm is comparatively fast, so cuPC algorithm is designed to accelerate the first step. Zarebavani et al. utilized CUDA, a parallel programming API for Nvidia CPU, and managed to implement the first step of PC algorithm with three parallel functions on GPU 2. This implementation of three parallel functions up speed the runtime for 500 times, compared to the classic loops.

3 Methodology

Inspired by the parallel implementation of PC algorithms, in this section, we will present our promoted implementation of Latent Hierarchical Causal Structure Discovery Algorithm with Rank Constraints with a python API, Numba.

A short introduction to Numba is presented in Section 3.1. We will then discuss the implementation of algorithm with Numba in Section 3.2.

3.1 Numba

Numba is a just-in-time compiler for Python that translates Python functions, Numpy, and loops into optimized machine code that promotes the efficiency of the functions. By adding appropriate decorators in front of functions with large logical computations and loops, Numba automatically handles the acceleration process [1].

`@jit` is a frequently used Numba decorator. `@jit` has two modes: `nopython` and `Object`. `nopython` is executed when the Python C API is not accessed, and this mode has the best efficiency. When a`@jit` decorated function is called, Numba first compile it with `nopython` mode. If `nopython` mode is not suitable, Numba will turn to `Object` mode, a more general form with the expense of efficiency.

3.2 Implementation of Algorithm with Numba

To achieve our goal of enhancing efficiency, we integrate Numba into the original implementation of Latent Hierarchical Causal Structure Discovery Algorithm with Rank Constraints. Because Numba is most effective in optimizing functions with loops and heavy mathematical and logical operators, we aim to extract loops from the original implementation to facilitate the compilation of Numba.

For instance, figure 3 is a piece of function that implements the Phase I in the algorithm, introduced in Section 2.1.1. We extract the for-loop within this function and decorate the function with `@jit` and auto-parallel option. As shown in Figure 4, the decorated function consists of solely loops and logical operators. Thus, this implementation with Numba should be theoretically effective in leveraging the performance of the original function. Similarly, we utilize this approach in other functions with Numba. We further discussed our outcome of this implementation in the next Section.

```python
def _findClusters(self, G: LatentGroups):
    """
    Called by findClusters and refineClusters.
    """
    assert isinstance(G, LatentGroups), "G must be a LatentGroups object."

    prevCovers = set(G.latentDict.keys())  # Record current latent Covers

    k = 1
    while True:
        LOGGER.info(f"{'-'*15} Test Cardinality now k={k} {'-'*15}")
        LPowerSet = self.generateLatentPowerset(G)
        activeSetCopy = deepcopy(G.activeSet)

        # Select a combination of latents, and replace their place in
        # the activeSet with their children for the search
        for Ls in LPowerSet:
            Vprime = deepcopy(activeSetCopy)
            for L in Ls:
                children = G.findChildren(L)

                # If children of L is just one variable, do not replace
                if len(children) > 1:
                    Vprime = Vprime - set([L])
                    Vprime |= children
            G.activeSet = Vprime
```

Figure 3: Piece of Original Implementation

```python
@jit(parallel=True)
def loop_powerset(self, G, LPowerSet,activeSetCopy,k):
    for Ls in LPowerSet:
        Vprime = deepcopy(activeSetCopy)
        for L in Ls:
            children = G.findChildren(L)

            # If children of L is just one variable, do not replace
            if len(children) > 1:
                Vprime = Vprime - set([L])
                Vprime |= children
        G.activeSet = Vprime
```

Figure 4: Implementation with Numba

4 Experimental Results

In this section, we conducted experiments to evaluate the performance of the proposed algorithm by Huang[2]. Before adopting the existing algorithm with the cuPC method, we first reran the original script and tested the performance of the algorithm as described in the paper.

4.1 Experimental Setup

We applied the proposed algorithm to synthetic data in order to learn different types of latent hierarchical causal graphs. Specifically, we considered two types of general Irreducible Linear Latent Hierarchical Graphs (IL2H), a tree structure, and one type of measurement model.

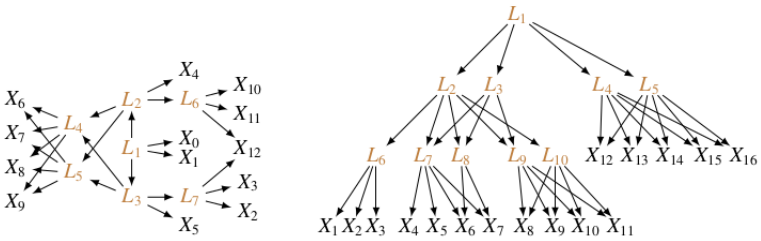

Figure 5: IL2H model

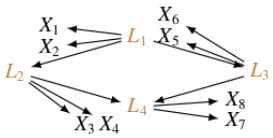

Figure 6: Measurement model

To assess the algorithm's performance, we ran the script multiple times to calculate the average runtime and enable comparison with an improved model. Additionally, we tested the algorithm using different sample sizes (1k, 2k, 5k, and 10k) to determine if there was a correlation between runtime and sample size. Consistent with the paper, the causal strength was generated uniformly from $[-5, -0.5] \cup [0.5, 5]$, and the noise term followed either a Gaussian distribution or a uniform distribution.

4.2 Evaluation of Causal Graphs

We examined the resulting causal graphs in detail to evaluate the algorithm's performance.

For the first type of general IL2H graph, with a sample size parameter set to 10k, we compared the ground truth graph (left side) with the generated result (right side). The generated result included three different phases corresponding to the three steps mentioned earlier. The third phase result represented the final result.

We manually checked the clusters generated by the algorithm and confirmed that it successfully recovered the clusters of observed variables. We used different colors to represent each cluster, allowing us to verify that all clusters from the ground truth were generated.

From the ground truth, {X6, X7, X8, X9}, {X0, X1}, {X10, X11, X12}, and {X2, X3} were measured variable clusters. Notably, {X6, X7, X8, X9} shared the same latent parent nodes.

Furthermore, we observed that X4 and X5 were individual nodes not belonging to any variable clusters. Specifically, for X4, we found that it shared the same parent cluster as {X6, X7, X8, X9}, {X0, X1}, and {X10, X11, X12}. Similarly, X5 shared the same parent cluster as {X2, X3}, {X0, X1}, and {X6, X7, X8, X9}. Hence, our results aligned with the ground truth, demonstrating that the model accurately recovered the causal relationships and clusters.

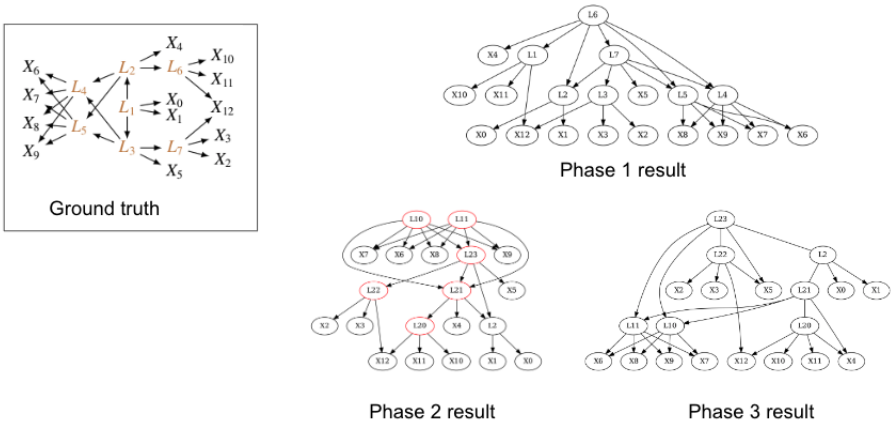

Figure 7: IL2H1 Result

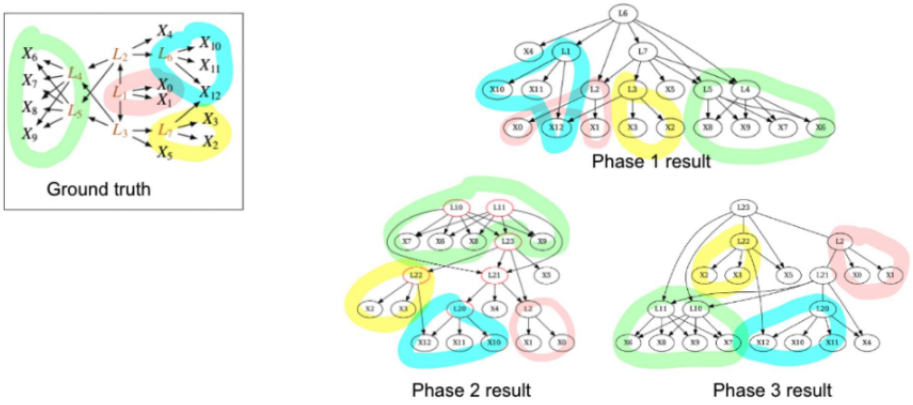

Figure 8: IL2H1 Result

Next, we present the ground truth and our results for the other type of IL2H structure, with the sample size parameter set to 10k. Similar to the first IL2H structure, we examined the clusters and causal relationships between latent variables generated by the algorithm. We found that the model perfectly recovered the graph when the sample size was 10k.

We also tested the algorithm with a different scenario where the input was a measurement model. On the right side, we show the generated result for this scenario. Surprisingly, we noticed some differences compared to the ground truth, as some causal edges were not generated by the model. We hypothesize that the model performs better when the input is an IL2H graph. Upon further consideration, we realized that the paper aimed to compare the performance of the proposed algorithm with the FOFC algorithm [Kummerfeld and Ramsey, 2016] and GIN Xie et al. [2020], which may explain the focus on IL2H graphs in the evaluation.

4.3 Runtime Analysis

We calculated the average runtime for each of the three causal graphs and different sample sizes. We conducted 10 runs for each experiment and observed that there was no linear advancement in runtime across the runs. To gather more comprehensive results, we increased the number of runs in subsequent experiments, yet the average runtime remained almost unchanged.

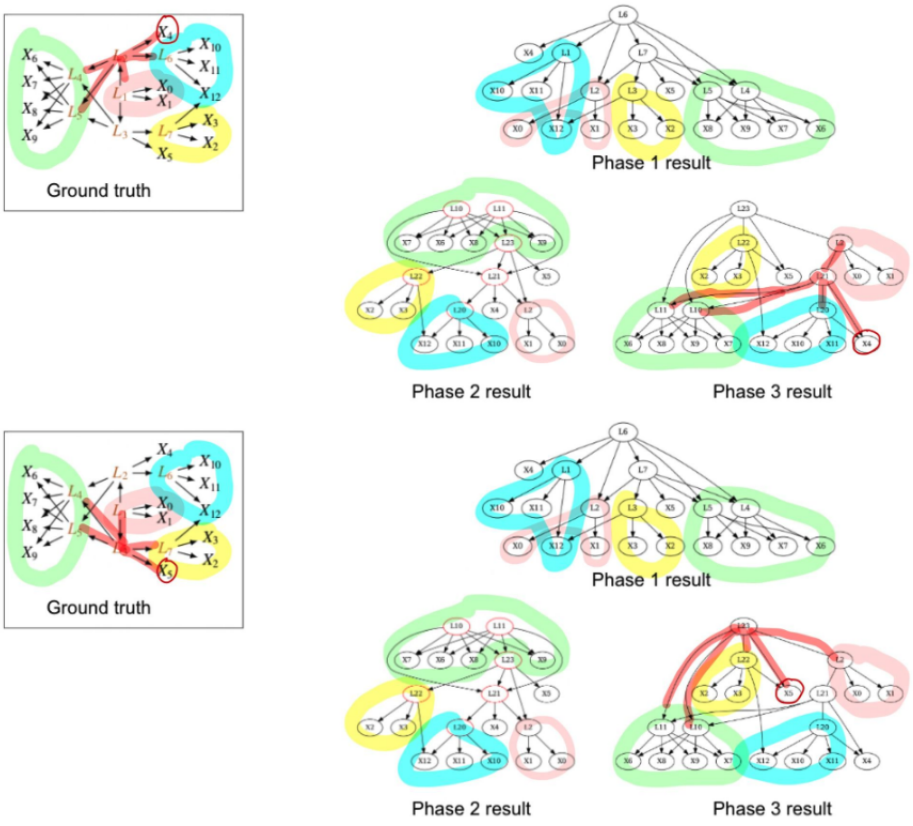

Figure 9: IL2H1 Result

4.4 Evaluation Metrics

We also assessed the performance by calculating the recovery rate over the measured variables and all variables. We used the following two metrics for our evaluation.

Given that all variable clusters appeared to be recovered perfectly by the model, and without finding any alternative definitions for the equation, we can conclude that the recovery rate is 100 percent. However, we acknowledge that the scenario used in the paper might be more complex and contain additional variables that could impact the recovery rate.

Overall, our experiments demonstrated the effectiveness of the proposed algorithm in recovering latent hierarchical causal graphs, particularly for the IL2H structure. The algorithm accurately identified causal relationships and clusters of observed variables. However, when tested with a measurement model, there were some deviations from the ground truth, indicating potential limitations of the algorithm in this specific scenario. The runtime analysis showed consistent performance across multiple runs, and the recovery rate metric indicated a high level of success in capturing the variable clusters.

4.5 Comparison

Now, we compare two methods using the Scenario 1a. We run 10 times on each number of sample and take the average the run time. It can be seen that our algorithm generally run faster than the original algorithm 13. In order to have a more general result, we can try more experiments on more complicated graph and run more times so that our result would have a better convergence.

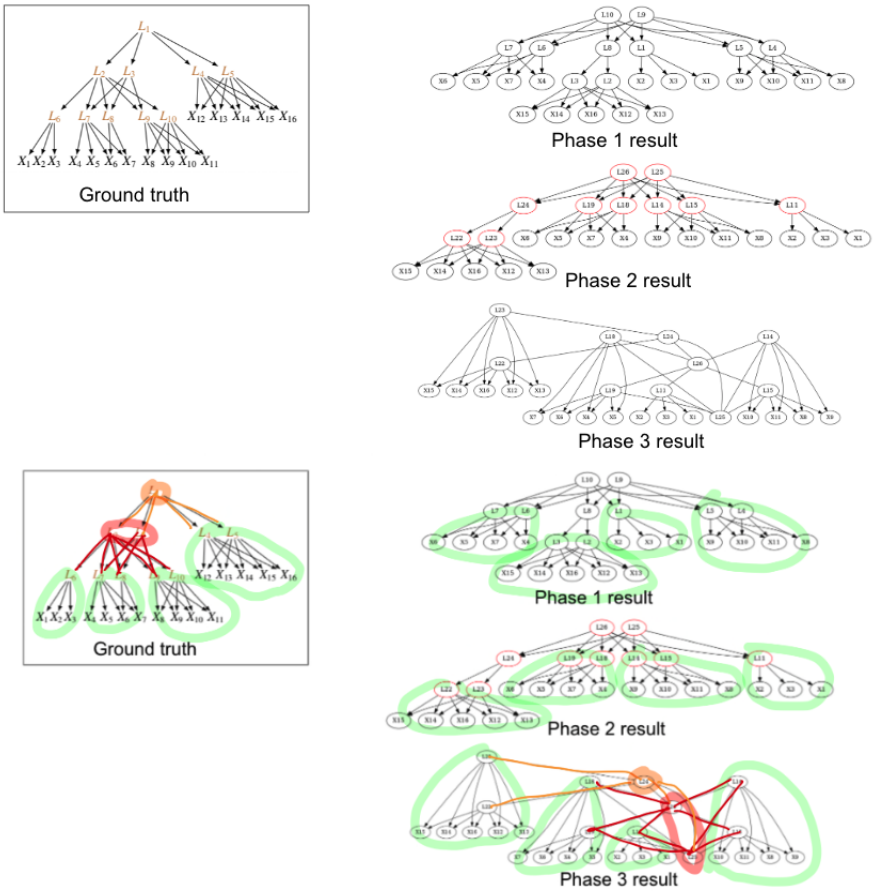

Figure 10: IL2H2 Result

5 Conclusions and Future Work

5.1 Acknowledgement

We would like to express our sincere gratitude to all those who have contributed to the successful completion of this study and the development of the proposed algorithm.

First and foremost, we would like to thank the authors of the paper[2] for their insightful work, which laid the foundation for our research. Their algorithm served as an essential reference point and source of inspiration for our own improvements.

We extend our appreciation to our research advisor, Biwei Huang, for their invaluable guidance, mentorship, and support throughout this project. Their expertise and insights were instrumental in shaping the direction of our research and enhancing the efficiency of the algorithm.

5.2 Algorithm Constraints

Two main constraints for this algorithm and project. These are the same as the constraints discussed in the original paper [2]. The first one is *linear relationships*. Our algorithm focuses exclusively on linear relationships for causal inference. By doing so, we simplify the interpretation of causal effects within a linear framework. This constraint allows us to easily understand the direction and magnitude of the causal effects discovered by the algorithm. Second, we have *variable independence*. We impose a constraint that prevents measured variables from causing latent variables. This constraint plays a crucial role in preserving the integrity of our causal estimates. By avoiding spurious relationships, we ensure that the algorithm provides reliable and meaningful causal effects.

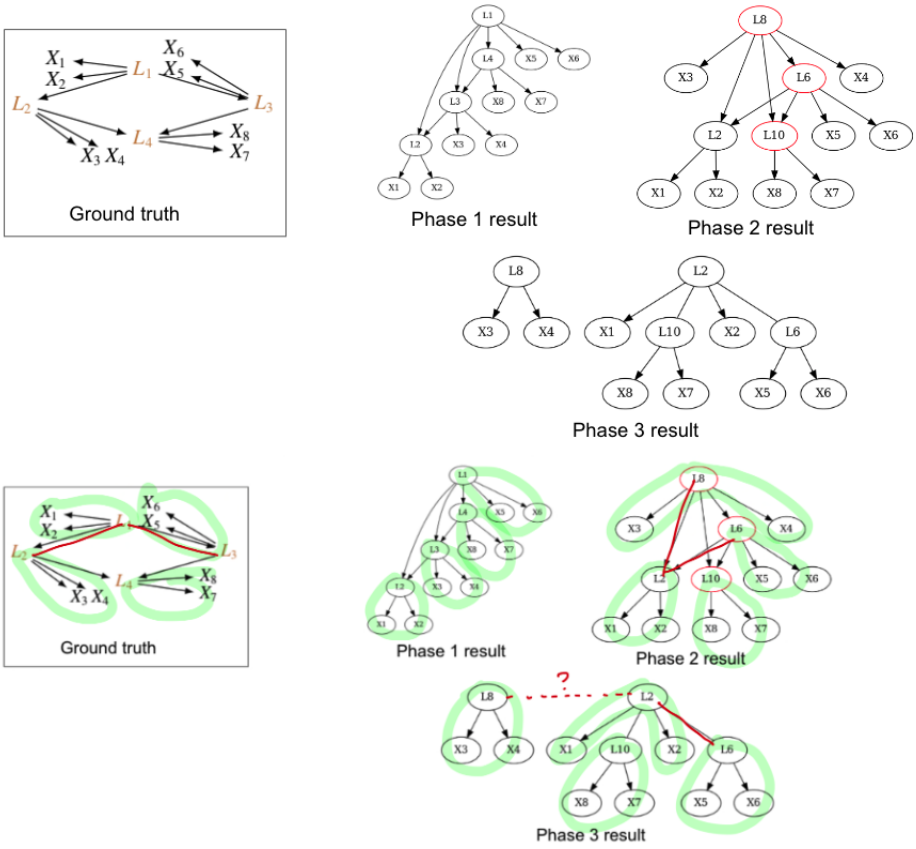

Figure 11: Measurement Model Result

	1k	2k	5k	10k
IL²H-1a	11.320s	11.417s	11.324s	11.451s
IL²H-1b	9.223s	9.033s	9.090s	9.025s
Measurement model	4.237s	4.221s	4.225s	4.280s

Figure 12: Runtime

5.3 Future Work

In this project, our primary focus is to improve the runtime of the algorithm proposed in the paper. We aim to optimize the existing algorithm by implementing parallelization techniques and leveraging computational resources to expedite the identification of the causal structure. By employing techniques such as horizontal parallelization, utilizing Numba and CUDA, we anticipate significant reductions in runtime, enabling more efficient analysis of large datasets.

In addition to improving runtime, our future work will also involve exploring non-linear relationships within the causal structure. While the current algorithm is constrained to linear relationships, we recognize the importance of capturing non-linear dependencies in real-world systems. Therefore, we plan to extend the algorithm's capabilities to handle non-linear relationships, allowing for a more comprehensive understanding of complex causal effects.

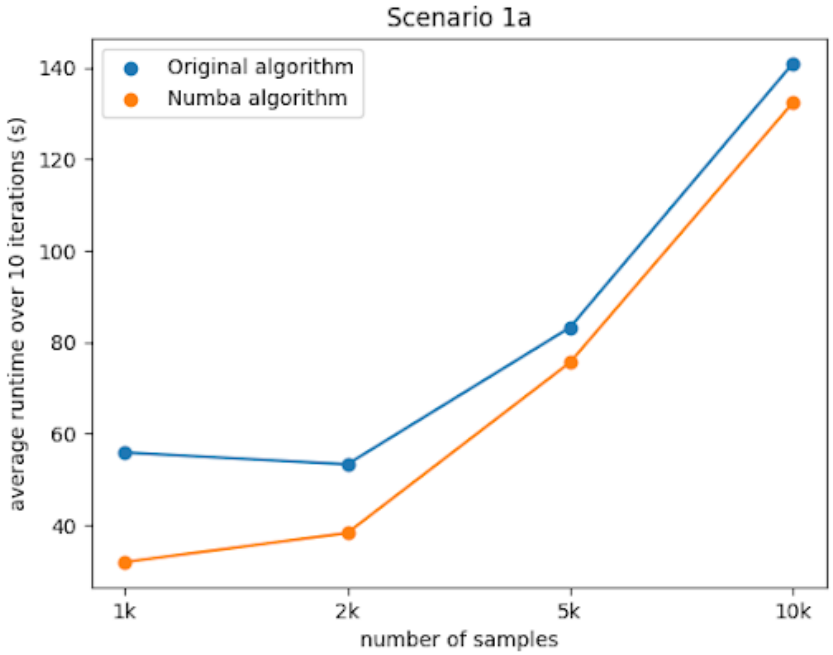

Figure 13: Comparasion of two method

Furthermore, we will investigate the possibility of relaxing the constraint on variable independence. This will enable us to explore scenarios where measured variables can cause latent variables, providing insights into more intricate causal relationships. By relaxing this constraint in a controlled manner, we aim to uncover additional causal dynamics and enhance the algorithm's applicability to a broader range of causal inference scenarios.

Through these future research directions, we anticipate significant advancements in the runtime efficiency, flexibility, and scope of the algorithm, ultimately improving our ability to uncover and interpret causal relationships in complex systems.

References

[1] Numba Documentation. `https://numba.readthedocs.io/en/stable/user/5minguide.html`. Accessed on June 16, 2023.

[2] Biwei Huang, Charles Low, Feng Xie, Clark Glymour1, and Kun Zhang. Latent hierarchical causal structure discovery with rank constraints. 2022.

[3] Anders L. Madsen, Frank Jensen, Antonio Salmerón, Helge Langseth, and Thomas D. Nielsen. A parallel algorithm for bayesian network structure learning from large data sets. *Knowledge-Based Systems*, 117:46–55, 2017. Volume, Variety and Velocity in Data Science.

[4] Behrooz Zarebavani, Foad Jafarinejad, Matin Hashemi, and Saber Salehkaleybar. cupc: Cuda-based parallel pc algorithm for causal structure learning on gpu. *IEEE Transactions on Parallel and Distributed Systems*, 31(3):530–542, 2020.