Karlsruher Institut für Technologie

Institut für Telematik
Pervasive Computing Systems / TECO
Leiter: Prof. Dr.-Ing. Michael Beigl

Evaluation and Comparison of Causal Discovery Algorithms

Bachelor´s Thesis

by

Hui Gong

Department of Informatics

Responsible Supervisor:	Prof. Dr. Michael Beigl
Supervising Staff:	Ployplearn Ravivanpong
Project Period:	30.11.2022 - 30.03.2023

Abstract

Causal discovery algorithms are very important in many fields because of their ability to reveal the underlying causal structure from observational data. With more and more causal discovery algorithms being developed, it is hard for researchers to determine which algorithm to use for their specific domain of work. In this paper, several causal discovery algorithms are made to run across datasets with varying variable numbers and observation numbers, then their performance is evaluated using the same metric. The result shows a low varsortability, such as 0.21 is achievable with synthetic datasets. Algorithms that rely on the assumption of causal sufficiency may exhibit similar performance on datasets that either with or without hidden variables, provided that the number of such variables is minimal. Algorithms that assume acyclicity, on the other hand, may display instability in their performance when applied to datasets containing cycles. Regarding the overall performance of tested causal discovery algorithms, it has been observed that GES underperformed in comparison to other algorithms. Therefore, it may not be considered the optimal choice for causal discovery. In this paper, the performance of several causal discovery algorithms is evaluated. Future research will focus on gathering more algorithms and datasets to do more cross-comparison in order to better understand the practical characteristics of different causal discovery algorithms.

Contents

List of Figures

List of Tables

1. Introduction

The use of causal discovery methods to extract knowledge from observational data has gained increasing attention in recent years. With the increasing availability of large amounts of data, combined with advances in computational techniques, it is now possible to use causal discovery methods to understand complex systems and gain valuable insights into the underlying causes of various phenomena, especially in fields such as medicine, epidemiology, social science, and economics, where interventions are difficult or even impossible to conduct.

The main objective of causal discovery is to identify the relationships between variables in a data set by conducting a graphical representation, in which the nodes stand for the variables and edges indicate direct causality between them.

1.1 Motivation

Currently, there are numerous algorithms available for causal discovery, with new ones being developed and published frequently. Each method claims to be more precise or better suited for certain situations than others. However, evaluations and comparisons, where algorithms are tested on the same data sets and evaluated using the same metrics, are lacking. Therefore, it can be challenging for researchers to determine which method would be the most suitable for their specific data set and application area. They often have to rely on the claims made by the authors when choosing methods for their work.

1.2 Goal

In this paper, several datasets are collected and unified into the same data format. Then, causal discovery algorithms are made to run on the same datasets and evaluated using the same metrics. The goal is to perform evaluation and comparison of causal discovery algorithms under equal grounds across data sets, thus giving some insights into the practical characteristics of each technique.

1.3 Related work

Before conducting the evaluation, several related papers are taken into consideration for inspiration and guidance.

Evaluation of causal structure learning methods on mixed data Types [1] evaluated several widely used causal structure learning methods on mixed data types under different parameter settings and sample sizes. PC and its variants, Greedy Equivalence Search(GES) were selected in this paper to represent two types of approaches in causal discovery: constraint-based and score-based. Though it provided certain guidelines as to which method to choose in a given context, its main focus was on the mixed data type. In this paper, the evaluation is performed on causal structure learning methods across several data sets, with a different number of variables and observations, as supplementation to help researchers make decisions on which method to choose for their work.

A survey of bayesian network structure learning [2] can be used as a comprehensive and detailed reference material for nearly all well-established and state-of-the-art algorithms used to learn the graphical structures of Bayesian Networks from data. It covered 24 constraint-based, 22 score-based, and 15 hybrid learning algorithms, and highlighted the similarities as well as the differences between algorithms using a consistent set of terms. However, the conclusions in this paper are mainly drawn from information provided in other papers, instead of running the evaluation themselves.

In comparison with the survey paper [2] mentioned above, Review of causal discovery methods based on graphical models [3] focused on the characteristics of causal discovery methods, which means, more details about how these algorithms work and the fundamental theories they are based on. Constraint-based, score-based, and functional causal models were discussed in this paper. Several illustrations and applications were also provided to help build a better understanding of these causal discovery methods. However, the evaluation and comparison of these algorithms were not covered in this paper.

Instead of focusing on the algorithms that learn causal model from data, The case for Evaluating causal models using interventional measures and empirical data [4] emphasized the evaluation techniques, more precisely, They advocated for greater use of evaluative techniques that focus on interventional measures instead of structural or observational measures, and to evaluate those measures using actual data rather than synthetic data. An Example of the evaluation was also provided, however, evaluating the performance of algorithms using their advocated techniques was not the center of this paper.

Benchpress[5] is a Snakemake[1] workflow made for causal discovery. It provides a "all-in-one" environment for benchmarking structure learning algorithms for graphical models. From causal discovery algorithms to datasets and evaluation metrics, they are all included in this framework. However, it needs the support of snakemake and apptainer[2] to be installed properly. It works at its best in a Linux environ-

[1] https://snakemake.readthedocs.io/en/stable/
[2] https://apptainer.org/

ment in a cluster, otherwise, docker[3] is needed. Due to the burdensome installation requirements of its dependencies, Benchpress is not utilized in this paper.

1.4 Structure of paper

In this paper, the background, including basic concepts of causal discovery, integrated causal discovery algorithms, datasets, and evaluation metrics used in the experiment are introduced. Then, details of the experiment setup are provided. In the evaluation part, firstly, several research questions are introduced, and then the corresponding result of each research question is presented. In the end, the conclusions drawn from this experiment and possible further works are discussed. The specific contributions of this paper are the following:

- Evaluation of several causal discovery algorithms on the same datasets of varying observation and variable sizes with the same metrics.
- Comparison of causal discovery algorithms against each other regarding their performance.

[3]https://www.docker.com/

2. Background

In this chapter, the causal discovery algorithms, datasets, and evaluation metrics used in this paper are introduced.

2.1 Algorithms for causal structure learning

2.1.1 The main approaches

According to [1], there are two main automated approaches when it comes to finding the causal structure from observational data: constraint-based and score-based. In general, the constraint-based approaches start with a fully connected undirected graph and then remove edges based on the result of conditional independence tests. After that, the graph is directed by a set of rules. Score-based approaches use a "goodness-of-fit" score of the model to the data while imposing a sparsity penalty to prevent overfitting. Both of them are proven successful in the past. Based on them, hybrid approaches have also been proposed. Nowadays, with the development of machine learning technology, several causal discovery algorithms based on continuous optimization have also been proposed.

In this paper, constraint-based algorithms and score-based algorithms are included. Algorithms that fall into the category of continuous optimization are also included.

Category	Algorithms
constraint-based	PC
function-based	DirectLiNGAM, ICALiNGAM
score-based	GES
gradient-based	Notears, NotearsLowRank

Table 2.1: List of the algorithms integrated into this project categorized by approach.

2.1.2 Assumptions

There are several important assumptions in the field of causal discovery algorithms. In this paper, the following assumptions are introduced:

- **Causal sufficiency**: Causal sufficiency, according to [6], means that there are no hidden or latent variables. Two typical approaches to modeling hidden variables are available: either by explicitly modeling them as nodes in structural equations or by their manifestation as a dependence among noise terms $(\epsilon_1, ..., \epsilon_p)$, which are presumed to be independent in the absence of latent confounding.

- **Causal faithfulness**: Causal faithfulness means all conditional independences in true underlying distribution p are presented in the true graph. According to [7], causal faithfulness can be expressed by this formula: Here is a direct quote "P_X is faithful to the DAG G if $\mathbf{A} \perp\!\!\!\perp \mathbf{B}|\mathbf{C} \Rightarrow \mathbf{A} \perp\!\!\!\perp_G \mathbf{B}|\mathbf{C}$ for all disjoint vertex sets $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$."

- **Acyclicity**: Acyclicity means the causal structure can be represented by a directed acyclic graph. According to [6], if cycles exist, the following aspects will be affected:

  1. "Existence of a unique equilibrium solution of $X \leftarrow BX + \epsilon$ where B is a $p \times p$ matrix and the distribution of X is determined by the choice of B and the distribution of ϵ."
  2. "Convergence to a stable equilibrium."
  3. "Existence of a stable equilibrium under do-interventions."

  With DAG, all three of the above aspects can be fulfilled.

In section 2.1.3, the algorithms integrated into this project and their assumptions are listed.

2.1.3 Table of algorithms

Algorithms integrated into this project and their assumptions are listed in table 2.2. For acyclicity, there is a concept called CPDAG, full name: completed partially directed acyclic graph. It is an equivalence class of DAGs because it contains edges with no explicit direction. For example, an undirected edge between A and B means it accepts both A to B and B to A. So far, the algorithms are from gCastle[1].

PC is one of the most famous constraint-based algorithms. It has the advantage of being super-fast and having relatively good performance. GES is representative of score-based algorithms. It also has a relatively fast run time. The downside of both PC and GES is that they have CPDAG as output thus the result is an equivalent class of DAGs, not one unique DAG.

ICALiNGAM and DirectLiNGAM are from the LiNGAM family. Their advantage is that the estimated true graph is weighted thus they provide more information on the causal relationship. Please note that in this paper, the weighted true graph is

[1] https://github.com/huawei-noah/trustworthyAI/tree/master/gcastle

Name	Sufficiency	Faithfulness	Acyclicity	Library
PC	Yes	Yes	Yes (CPDAG)	gCastle
GES	Yes	Yes	Yes (CPDAG)	gCastle
ICALiNGAM	Yes	No	Yes (DAG)	gCastle
DirectLiNGAM	Yes	No	Yes (DAG)	gCastle
Notears	-	-	Yes (DAG)	gCastle
NotearsLowRank	-	-	Yes (DAG)	gCastle

Table 2.2: Table of the algorithms used in this project. The "-" sign means no explicit information was found.

converted into a true graph with only zeros and ones where ones indicate there's a causal relationship, and zeros mean there's no causal relationship.

Notears and NotearsLowRank are both gradient-based. They greatly benefited from the development of machine learning technology and the downside is their run time is relatively longer than other algorithms like PC or GES. To achieve the optimized result, hyperparameter tuning is needed.

2.2 Datasets

The datasets used in this paper are collected from multiple sources and so far, a total of 1795 datasets are collected. It includes real, semi-synthetic, and synthetic datasets. Real datasets are from the real world, synthetic datasets are artificially generated and semi-synthetic datasets are generated based on real data. Datasets with and without hidden variables are also included. All of the datasets are unified to `numpy.ndarray`[2] data type when loaded into the experiment. The varsortability of each dataset has also been calculated.

In the following section, all the datasets that are integrated into this project are listed. (See Table: 2.3, 2.4, 2.5). Information about them can also be found in the correspondent GitHub repository[3] of this paper.

2.2.1 Real datasets

Table 2.3 lists the real datasets. Source of jdk, networking and postgres can be found here[4]. The other four real datasets can be found here[5]. In the case of the dataset "networking", dummy variables are used to represent categorical variables such as "server", which has 7 levels. Please note that the sachs dataset is different from the version on bn-learn[6]. auto_mpg, cites, and yacht doesn't have an explicit true graph but a ground truth knowledge. It contains information on specific forbidden and required edges and is represented in the following format:

[2]https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html
[3]https://github.com/ravivanpong/CausalBench/tree/main/causalbench/data
[4]https://groups.cs.umass.edu/kdl/causal-eval-data/
[5]https://github.com/cmu-phil/example-causal-datasets/tree/main/real
[6]https://www.bnlearn.com/bnrepository/

Name	Variable Number	Observation Number	Remark
jdk	15	37840	
networking	17	415840	
postgres	21	1506270	
auto_mpg	8	392	
cites	7	7	
sachs	11	7466	Original version. Different from the version on bn-learn
yacht	7	308	

Table 2.3: Real Datasets

```
/knowledge
addtemporal

1*   x y z
2   w r

forbiddirect
w z

requiredirect
a b
x b

...
```

The true graph is then converted from the ground truth knowledge based on the following steps:

1. Start with a fully connected graph.
2. Remove edges from later tiers to earlier tiers.
3. If a tier number is followed by an asterisk, remove all edges within that tier.
4. Remove edges specified in the "forbiddirect" section.

2.2.2 Synthetic datasets

Table 2.4 lists the synthetic datasets. Datasets with and without hidden variables can be found in this URL[7]. The others in the table can be found in this URL[8]. It contains datasets with and without hidden variables and datasets with a relatively large number of variants. Here the "variant" means that the datasets all have the same true graph. They are generated under the same condition using the same method.

[7]https://dataverse.harvard.edu/dataset.xhtml?persistentId=doi:10.7910/DVN/UZMB69
[8]https://github.com/cmu-phil/example-causal-datasets/tree/main/simulated/feedbacks

Name	Variable Number	Observation Number	Remark
with_confounders	$17 \sim 21$	500	20 datasets with 3 hidden variables.
without_confounders	$20 \sim 24$	500	20 datasets without hidden variables.
Network1_amp	5	500	60 variants
Network2_amp	5	500	60 variants
Network3_amp	5	500	60 variants
Network4_amp	10	500	60 variants
Network5_amp	5	500	60 variants
Network5_cont	5	500	60 variants
Network5_cont_p3n7	5	500	60 variants
Network5_cont_p7n3	5	500	60 variants
Network6_amp	8	500	59 variants. Variant number 25 is missing.
Network6_cont	8	500	60 variants
Network7_amp	6	500	60 variants
Network7_cont	6	500	60 variants
Network8_amp_amp	8	500	60 variants
Network8_amp_cont	8	500	60 variants
Network8_cont_amp	8	500	60 variants
Network9_amp_amp	9	500	60 variants
Network9_amp_cont	9	500	60 variants
Network9_cont_amp	9	500	60 variants

Table 2.4: Synthetic Datasets

2.2.3 Semi-synthetic datasets

Semi-synthetic Datasets are listed in Table 2.5. The source can be found in this URL[9]. Except for the "Dream4" datasets, the number appended to the name indicates how the data is generated. For example, "Alarm" has no number appended, meaning the data is generated from the original network. "Alarm3" means the data is generated from tiling 3 of the original networks. "Alarm5" means the data is generated from tiling 5 of the original networks, etc.

2.3 Evaluation metrics

There are several evaluation metrics available for this project. For example, true positive rate, precision, recall, F1-Score, etc. All the available metrics in this project are listed in section 2.3.2. In this section, several fundamental building components for calculating the value of metrics are introduced. Then, the metrics are explained in detail with formulas.

[9]https://pages.mtu.edu/ lebrown/supplements/mmhc_paper/mmhc_index.html

Name	Variable Number	Observation Number	Remark
Alarm	37	500, 1000, 5000	10 variants for each observation number
Alarm3	111	500, 1000, 5000	Same as above
Alarm5	185	500, 1000, 5000	Same as above
Alarm10	370	500, 1000, 5000	Same as above
Barley	48	500, 1000, 5000	Same as above
Child	20	500, 1000, 5000	Same as above
Child3	60	500, 1000, 5000	Same as above
Child5	100	500, 1000, 5000	Same as above
Child10	200	500, 1000, 5000	Same as above
Hailfinder	56	500, 1000, 5000	Same as above
Hailfinder3	168	500, 1000, 5000	Same as above
Hailfinder5	280	500, 1000, 5000	Same as above
Hailfinder10	560	500, 1000, 5000	Same as above
Insurance	27	500, 1000, 5000	Same as above
Insurance3	81	500, 1000, 5000	Same as above
Insurance5	135	500, 1000, 5000	Same as above
Insurance10	270	500, 1000, 5000	Same as above
mildew	35	500, 1000, 5000	Same as above
munin1	189	500, 1000, 5000	Same as above
pigs	441	500, 1000, 5000	Same as above
gene	801	500, 1000, 5000	Same as above
link	724	500, 1000, 5000	Same as above
Dream4_1	100	100	
Dream4_2	100	100	
Dream4_3	100	100	
Dream4_4	100	100	

Table 2.5: Semi-synthetic Datasets

2.3.1 Building components of metrics

Metrics that concentrate on evaluating the relationship between two graphical structures can be built by the following basic components:

- True Positives(TP), corresponding to the number of true edges present in the learned graph.
- False Positives(FP), corresponding to the number of false edges present in the learned graph.
- True Negatives(TN), corresponding to the number of true absent edges in the learned graph.
- False Negatives(FN), corresponding to the number of false absent edges in the learned graph.
- Reverse, corresponding to the number of edges with reversed direction in the learned graph.

2.3.2 List of metrics

These metrics are the higher the better:

- Number of Nonzero Elements(NNZ)
  - $$NNZ = TP + FP$$
  - NNZ shows the number of edges that are estimated by the algorithm, including those not estimated correctly.
- Precision
  - $$Precision = \frac{TP}{TP + FP}$$
  - Precision measures the ratio of correct edges from all edges in the estimated true graph.
- Recall
  - $$Recall = \frac{TP}{TP + FN}$$
  - Recall measures the ratio of the correctly estimated edges from all the edges in the actual true graph.
- F1
  - $$F1 = \frac{2 * Recall * Precision}{Recall + Precision}$$
  - The F1 score can be interpreted as a harmonic mean of the precision and recall, where an F1 score reaches its best value at 1 and worst score at 0.
- G-Score
  - $$G\text{-}Score = \max(0, \frac{TP - FP}{TP + FN})$$
  - Alternative of F1-Score but more strict.
- Normalized_SHD
  - $$Normalized\_SHD = 1 - \frac{SHD}{2 * ARC\_NUM}$$

  - Normalized_SHD is used to enable cross-comparison. Its value is between 0 to 1 with 1 indicating that the estimated graph is the same as the actual true graph. ARC_NUM is the total number of edges in the actual true graph.

These metrics are the lower the better:

- False Discovery Rate(FDR)
  -
$$FDR = \frac{Reverse + FP}{TP + FP}$$
  - FDR measures the ratio of incorrect edges from all the edges in the estimated true graph.
- False Positive Rate(FPR)
  -
$$FPR = \frac{Reverse + FP}{TN + FP}$$
  - FPR can be interpreted as "false alarm rate", it measures the ratio of edges that are incorrectly included in the estimated true graph from the edges that are not included in the actual true graph.
- Structural Hamming Distance(SHD)
  -
$$SHD = undirected\_extra + undirected\_missing + Reverse$$
  - SHD is the number of edge deletions, insertions, or flips in order to transform one graph into another graph.
- Runtime in Seconds
  -
$$Runtime = Finish\_timestamp - Start\_timestamp$$
  - Time measured in seconds. It indicates how long it takes for an algorithm to estimate the causal graph.

Structural Intervention Distance (SID) and Total Variation Distance(TVD) are not included as evaluation metrics in this experiment. SID is not included because there is no implementation ready to be used for this experiment and the original implementation is meant for intervention while some of the datasets in this experiment have no intervention. The reason for SID is similar. The original implementation is outdated and no longer maintained by the author. It's also quite tricky to run R code in a python codebase.

3. Experiment design

For the experiment design, the main goal is to run each algorithm on 1795 datasets. In this chapter, the workflow of the experiment and details of the implementation are introduced. Then how the parameters are chosen and the reasons behind them are explained. After that, it goes to the details of the runtime environment, including both hardware and software. In the end, the time limitation is introduced as a method to control the pace and energy consumption of the experiment.

3.1 Workflow

Figure 3.1 shows the workflow of the experiment. It starts with a JSON file as input, including information on selected algorithms and their parameters, datasets the algorithms will run on, and the name of the output file. Then, the JSON file will be parsed and a task list will be generated by combining given algorithms with given datasets. After that, functions provided by `concurrent.futures`[1] are used to run the tasks in parallel. At last, the experiment will be ended either by hitting the time limit or by finishing normally.

3.2 Experiment implementation

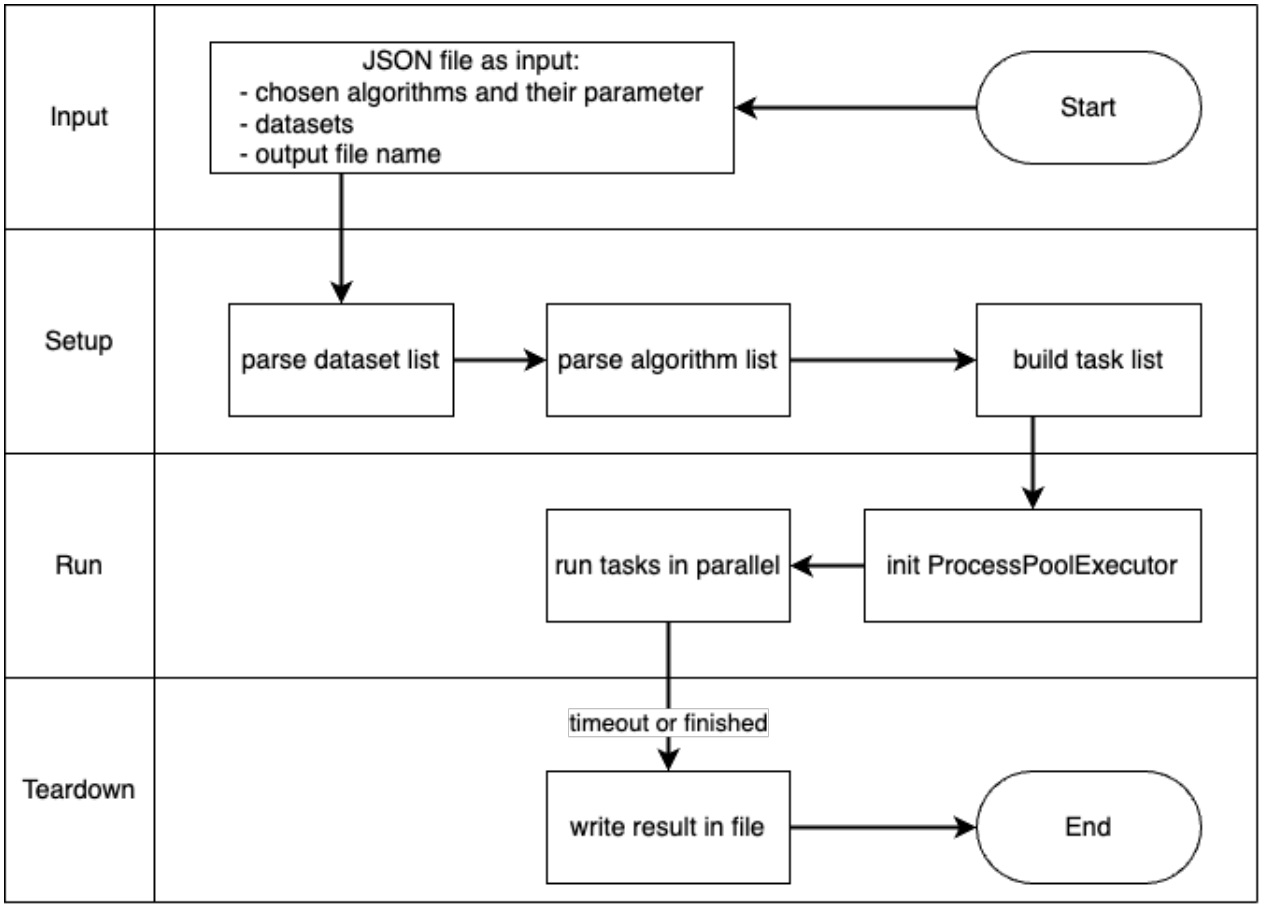
Implementation of the experiment is hosted on GitHub[2].

3.2.1 Command-line interface

`argparse`[3] is a popular tool to write user-friendly command-line interfaces. Here it is used to define the arguments the experiment requires. The arguments can be configured in a JSON file and the experiment takes the relative path to the JSON file as argument. To use it, the user needs to first navigate to the directory of the experiment python file, then use the command line interface to run it. Here is the general form of the command:

[1] https://docs.python.org/3/library/concurrent.futures.html
[2] https://github.com/ravivanpong/CausalBench
[3] https://docs.python.org/3/library/argparse.html

Figure 3.1: Workflow of experiment




```
$ experiment.py -src relative/path/to/json/file
```

For example, if the path to the `experiment.py` file is `username/Causalbench/experiment/castle/experiment.py`, and the path of the `input.json` file is `username/input.json`, it should look like this:

```
$ experiment.py -src ../../../input.json
```

3.2.2 JSON as input

The JSON file configures the algorithms and their parameters, as well as the datasets they will be run on. The output file name is also configured here. Here is an example of the general form of the JSON file:

```
{
    "datasets": [
        {
            "name": "alarm",
            "kwargs": {
                "index": 3,
                "sample_num": 500,
                "version": 1
            }
        },
```

```
        {
            "name": "dream4",
            "kwargs": {
                "version": 1
            }
        },
        {
            "name": "insurance",
            "kwargs": {
                "index": 1,
                "sample_num": 500,
                "version": 1
            }
        }
    ],
    "algorithms": [
        {
            "gcastle": [
                {
                    "name": "pc",
                    "kwargs": {
                        "variant": "stable",
                        "alpha": 0.05,
                        "ci_test": "fisherz",
                        "priori_knowledge": null
                    }
                },
                {
                    "name": "ges",
                    "kwargs": {
                        "criterion": "bic",
                        "method": "scatter",
                        "k": 0.001,
                        "N": 10
                    }
                }
            ]
        }
    ],
    "OUTPUT_FILE_NAME": "my_output"
}
```

The content of this JSON file can also be customized to run the same experiment with different algorithms and datasets. It can be stored anywhere locally with any name.

3.2.3 Data loader

Each dataset has a dedicated `dataset_loader.py` to provide easy access to the datasets. For example, if the input file is as defined in 3.2.2, the corresponding data loader will be called like this:

```
load_alarm(3, 500, 1)
load_dream4(1)
```

The loader function will return a dictionary object that contains:

- `true_matrix`: true graph of dataset in form of `numpy.ndarray`[4]
- `X`: dataset in form of `numpy.ndarray`.
- `var_num`: number of variables
- `sample_num`: number of samples
- `name`: full name of data set

3.2.4 Parallel computing

Because of the large number of datasets, parallel computing is necessary. The hardware environment also has multiple CPUs which makes parallel computing possible. In this experiment, `concurrent.futures`[5] was used to run the tasks in parallel. The process is:

1. Submit tasks to the executor
2. store the Future objects in an array
3. as soon as `as_completed()` is called, write the result in the predefined output file.

3.3 Parameter

In this experiment, default parameters are used because normally it is considered optimal by the author and it is the first choice if the user has no experience in tuning hyperparameters. And generally, tuning hyperparameters is hard due to a lack of ground truth. Another reason is that the computing resource is limited so in this experiment, all the algorithms are tested with their default parameters. There are some exceptions:

- By default, PC version is "original" in gCastle. In this experiment, version "stable" is used, because version "original" is obsolete.
- All algorithms that have a GPU version is running with GPU. The default is CPU.

[4]https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html
[5]https://docs.python.org/3/library/concurrent.futures.html

3.4 Run time environment

The hardware runtime environment is on BWUniCluster2 with 10 CPU, 90 GB RAM, and 1 GPU. The software environment is python 3.8.13, future 0.18.2, gcastle 1.0.3, lingam 1.7.0, scikit-learn 1.1.1, numpy 1.22.3, pandas 1.4.3, networkx 2.8.5, pytorch 1.13.0.dev20220721, and scipy 1.8.1

3.5 Time limit

Despite running in parallel, several time limits are set up, because the computing resource is limited and it is not practical and not green to run a single task for more than 3 days. Table 3.1 lists out the time limit for datasets with different variable numbers. For variable number between 3 and 100, the time limit is 1 day, between 100 and 300 is 2 days, and a maximum of 3 days for variable number between 300 and 1000.

Variable Number N	Time limit
$3 <= N < 100$	24h
$100 <= N < 300$	48h
$300 <= N < 1000$	72h

Table 3.1: Time limits

4. Results

In this chapter, research questions and their corresponding result are presented. Firstly, the research questions are introduced. Then, it goes to the results where the data used to draw conclusions from are presented and step for step making approach to the conclusions.

4.1 Research questions

There are 4 research questions that need to be answered:

- Do real, semi-synthetic, and synthetic datasets have similar varsortability?
- What is the impact of hidden variables on algorithms that assume causal sufficiency?
- How sensitive are the algorithms that assume acyclicity when there's a violation of it?
- How is the overall performance and what conclusion can be drawn from the results?

4.2 Varsortability

4.2.1 Significance of varsortability

Varsortability is proposed as a measure of agreement between the order of increasing marginal variance and the causal order in [8]. According to this paper, the data-generating process may leave information about the causal order in the data scale, so that varsortability is introduced as a measure of such information. It states that varsortability is high, for example, above 0.94 in simulated graphs and above 0.71 in non-linear additive noise models. The higher the varsortability, the easier to identify the causal structure. The value range of varsortability is from 0 to 1 with 1 indicating that the causal structure is identifiable.

4.2.2 Experiment configuration

The experiment is set up to calculate the varsortability of all integrated 1795 datasets. The implementation of varsortability is provided by [8] and is hosted on GitHub[1]. The results are written into a CSV file for further usage.

4.2.3 Result data

The summary data is exhibited in Table 4.1

	Real	Semi-synthetic	Synthetic
count	6	1124	664
mean	0.56	0.52	0.54
std	0.23	0.17	0.21
min	0.12	0.17	0.21
50%	0.62	0.52	0.50
max	0.75	0.85	0.87

Table 4.1: Summary of varsortability group by dataset type

One of the real datasets, "Networking", is not included in Table 4.1 and Figure 4.1 because the current implementation of varsortability failed to calculate its varsortability by giving "nan" as result.

It can be easily seen that real, semi-synthetic, and synthetic datasets have similar mean values of varsortability. All three of them have an average varsortability of around 0.5 and it is much lower than the empirical average varsortability from [8], which stated an average varsortability of above 0.94 in simulated graphs. However, conclusions like "The varsortability of real, semi-synthetic, and synthetic datasets are similar with each other." can not be drawn yet because the means value could be misleading sometimes and statistical tests should be done to tell if there is any statistical significance.

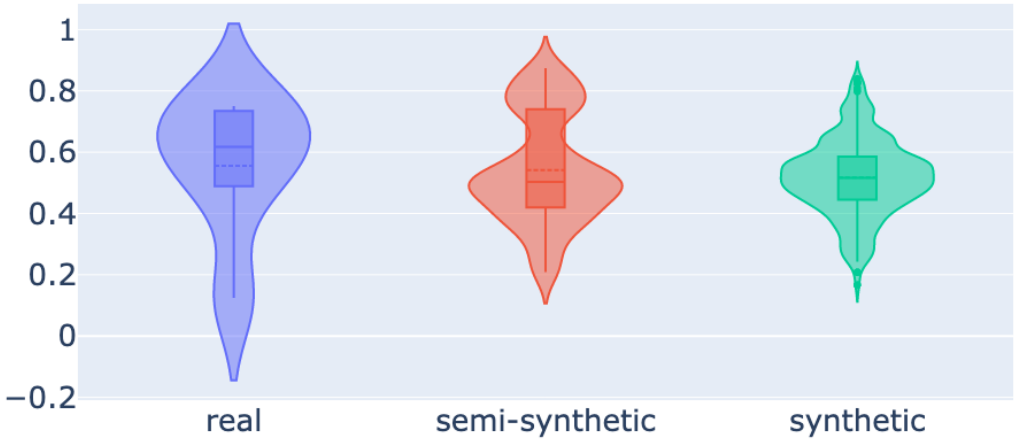
4.2.4 Statistical significance

Statistical tests are used to test if a relationship observed in the data occurred only by chance. The process includes the following steps:

1. Apply normality tests to see if the data is normally distributed.
2. If it is normally distributed, apply parametric statistical significance tests.
3. If not, apply nonparametric statistical significance tests for more complex distributions.

[1] https://github.com/Scriddie/Varsortability

Figure 4.1: Violin plot of varsortability



Here the real datasets are not included in the statistical tests because the number of them is too small to do a test (See Table: 4.1).

For the normality test, `scipy.stats.normaltest`[2] is used. Based on the result of the normality test, `scipy.stats.mannwhitneyu`[3] or `scipy.stats.ttest_ind`[4] is used to test if there is a statistical significance.

The result of test 4.2 shows that the synthetic datasets have a normal distribution of varsortability while it is unlikely that the semi-synthetic datasets have a normal distribution. After the Mann-Whitney U test, a p-value of 0.4 and a statistic of 364379.5 are given as result. Since 0.4 is smaller than the threshold of 0.5, the result indicates that the similar mean values between synthetic and semi-synthetic datasets could be a total coincidence and there is no relationship between them.

	statistic	p-value	result
synthetic	0.80	0.67	normal
semi-synthetic	64.64	9.17e-15	not normal

Table 4.2: Normality test on varsortability of synthetic and semi-synthetic datasets

4.3 Sufficiency and hidden variables

4.3.1 Experiment configuration

PC (stable), GES, ICALiNGAM, DirectLiNGAM are chosen for this experiment because they all assume causal sufficiency. There are two sets of datasets, one with

[2]https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.normaltest.html
[3]https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.mannwhitneyu.html
[4]https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.ttest_ind.html

20 datasets that have 3 hidden variables and another one with 20 datasets that have no hidden variable. The purpose is to evaluate the performance of those algorithms on these two sets of datasets, thus getting some insights into the impact of hidden variables on algorithms that assume causal sufficiency.

The performance of the algorithms is measured by F1-Score.

4.3.2 Result data

Here exhibit the violin graphs (see Figure 4.2) of the performance of those four algorithms on two sets of datasets: The left violin graph on each subfigure shows the distribution of the F1-Score of the corresponding algorithm on datasets with hidden variables. And the right one is on datasets without hidden variables. It is clear that the mean value of the F1-Score is higher when the datasets are coherent with the acyclic assumption. The detailed data can be seen in Table 4.3. However, the statistical test still needs to be done to check if it could be a mere coincidence.

4.3.3 Statistical significance

Table 4.4 shows that all of them have a normal distribution so a T-test is needed to check the statistical significance. Table 4.5 shows that except GES, the other algorithms have equal performance on datasets with 3 hidden variables and without hidden variables. So the result shows that GES indeed has better performance when the dataset is coherent with its causal sufficiency assumption and for the other three algorithms, the difference of their performance on datasets with 3 hidden variables and without hidden variables can be ignored.

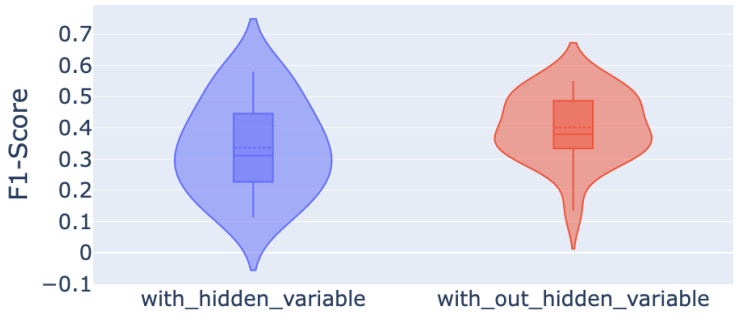
4.4 Acyclicity

This section concerns the sensitivity of algorithms that assume acyclicity on datasets that violate acyclicity. The approach is to run algorithms that assume acyclicity on groups of datasets that have one or more cycles in the true graph. Datasets that share the same true graph are in the same group. Each group contains 60 datasets that are generated under the same condition and share the same true graph. Ideally, if the algorithms are stable, the estimated true graph would be the same when they run on datasets within the same group.

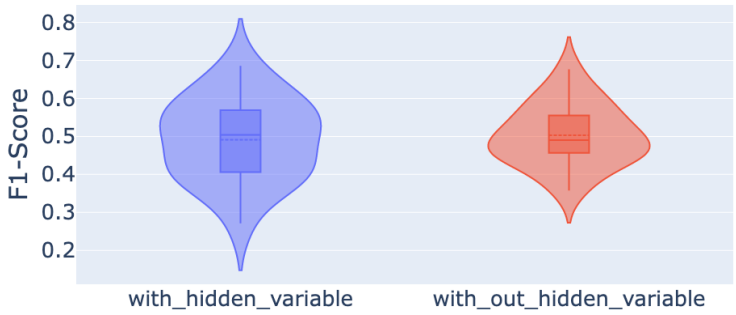
4.4.1 Experiment configuration

The experiment is setup with:

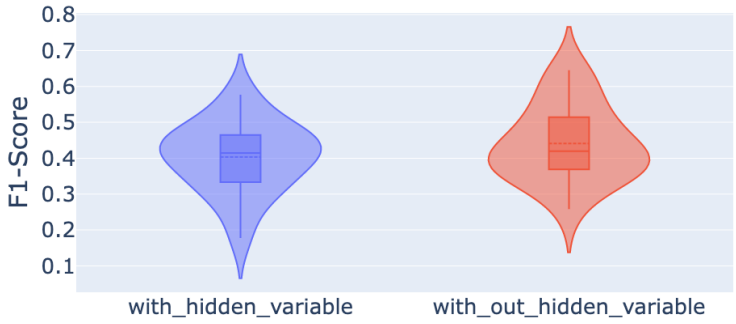
- 4 algorithms: PC (stable), GES, DirectLiNGAM, ICALiNGAM.
- 17 groups of datasets. (See Table 4.6)
- 60 variants in each group.



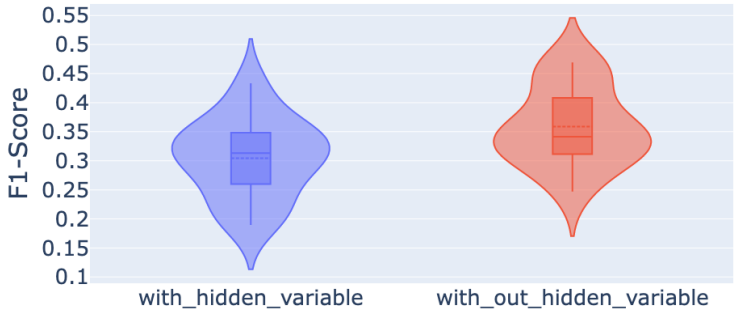
(a) DirectLiNGAM



(b) ICALiNGAM



(c) PC (stable)



(d) GES

Figure 4.2: PC, GES, ICALiNGAM, DirectLiNGAM on datasets with and without hidden variables

- Normalized_SHD as evaluation metric.

The reason that these datasets are chosen is that they have cycles in the true graph and the number of datasets in each group is relatively large so it is suitable for the sensitivity test. Netowrk6_amp is not included because it only has 59 variants. The other groups all have 60 variants.

4.4.2 Result data analyse

Figure 4.3 shows the violin graphs of the performance of those four algorithms on four of the selected group of datasets. The sub-figure names like "Network3_amp" are names of the group of the datasets. It can be seen that the violin plots all look thin and long, which means the normalized_SHD is scattered. And these four groups are already the best picked out from the tested groups. Ideally, if the algorithms are stable, the violin graph should be wide and short, which means the normalized_SHD is stable accross datasets within the same group. And thus, the estimated true graph would be the same or very similar to each other.

However, even if the violin graphs are all wide and short, it can not guarantee that the estimated true graphs are similar to each other. The estimated true graphs could have the same normalized_SHD but look different from each other. An example of PC (stable) on Network2_amp is shown in Figure 4.4. They have the same normalized_SHD but are different from each other.

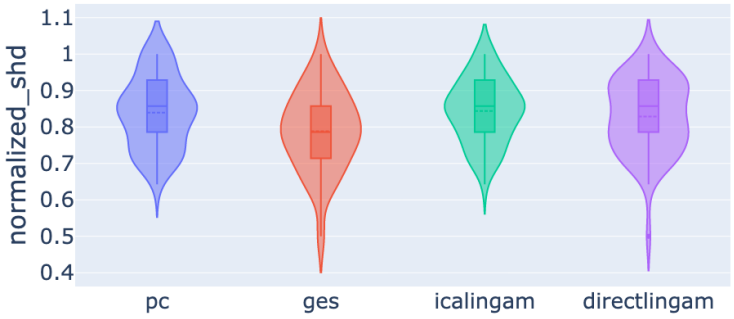
4.5 Overall performance

Due to limitations of time and computing resources, the author does not have the data of all the algorithms on all 1795 datasets. However, useful information can still be extracted from the existing available result data. In this section, the performance of 6 algorithms on 1079 datasets is evaluated by different metrics, and By doing this, insights into the practical characteristics of these algorithms can be provided.

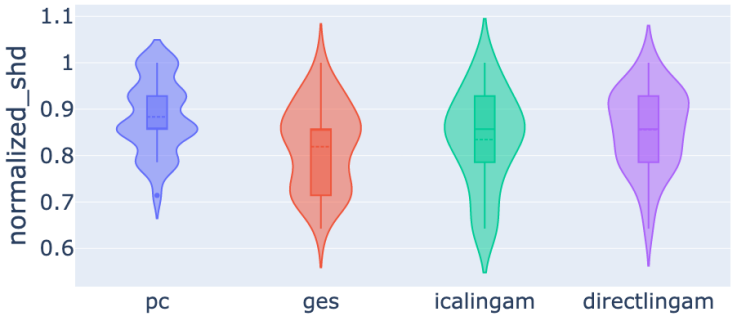
4.5.1 Experiment configuration

The experiment is setup with:

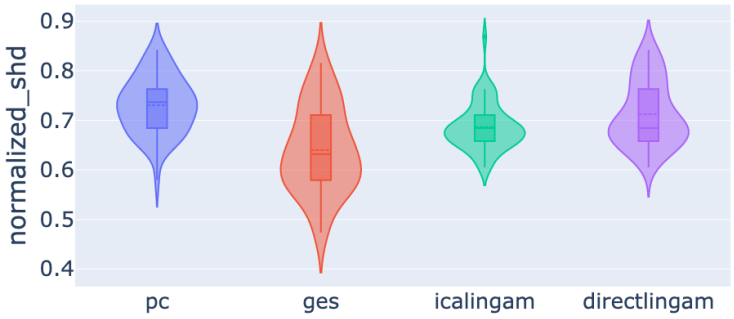
- 6 algorithms: PC (stable), GES, DirectLiNGAM, ICALiNGAM, Notears, Notearslowrank.
- 1795 datasets.
- Evaluated by:
  - Normalized_SHD
  - G-Score
  - F1
  - False Positive Rate(FPR)
  - Runtime in seconds



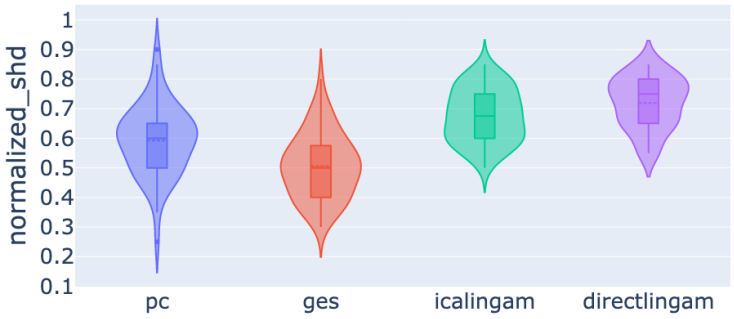
(a) Network2_amp



(b) Network3_amp

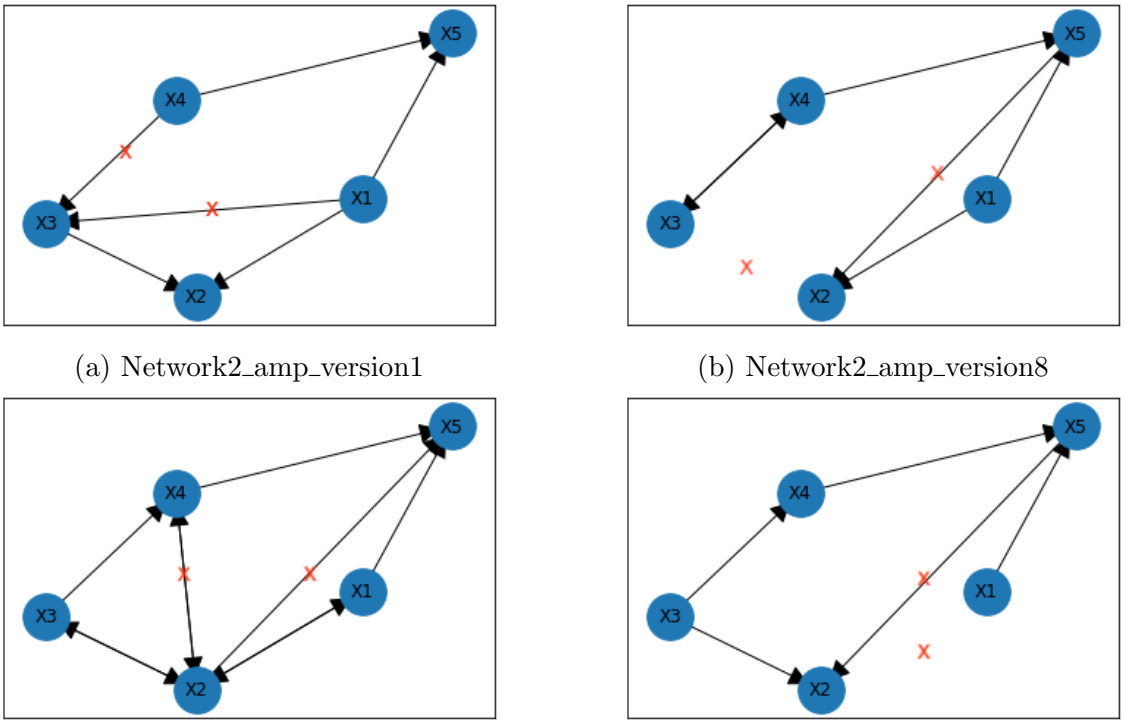


(c) Network4_amp



(d) Network9_amp_amp

Figure 4.3: PC, GES, ICALiNGAM, DirectLiNGAM on groups of datasets with cycles in their true graph.



(a) Network2_amp_version1

(b) Network2_amp_version8

(c) Network2_amp_version9

(d) Network2_amp_version18

Figure 4.4: Estimated true graphs of PC (stable) on 4 datasets in group "Network2_amp".

4.5.2 Result data analyse

Figure 4.5 shows the performance of PC (stable), GES, DirectLiNGAM, ICALiNGAM, Notears, Notearslowrank evaluated by Normalized_SHD, G-Score, and F1. With each value of varsortability, it shows the median of the metric value of each algorithm.

It is hard to find out a "winner" as the absolute "top" algorithm but ICALiNGAM seems always to have a place on the higher end of the metrics. DirectLiNGAM, an algorithm from the LiNGAM family, also has a relatively good performance against the others. According to this paper [9], there are some potential problems of ICALiNGAM. One of them is that ICALiNGAM may not converge in a finite number of steps. Another one is that the way ICALiNGAM works is not scale-invariant. It's possible that ICALiNGAM gives a different or even wrong ordering of variables depending on the scales of variables. These issues should be considered when researchers select a causal discovery algorithm for their research.

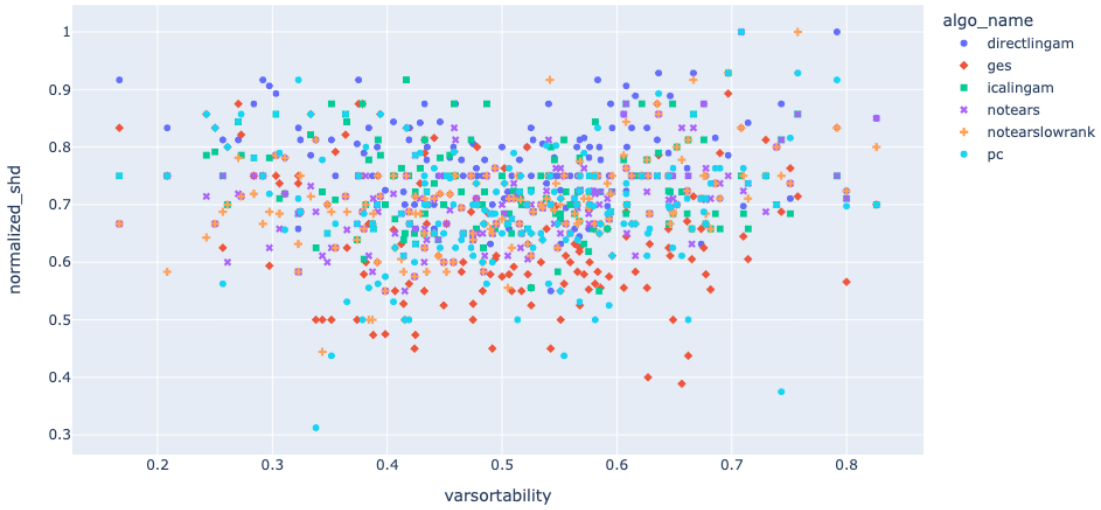
Except for ICALiNGAM, another algorithm also stands out from the others, in another way.

Table 4.7 shows the performance on normalized_SHD. It can be seen that GES has a slightly lower normalized_SHD compared with the others. Table 4.8 shows the performance on G-Score. It can be seen that GES also has a relatively lower value than the others. Table 4.9 shows performance measured by the False Positive Rate(FPR), it is clear that GES has a higher false alarm ratio. When we look at the runtime in seconds (see Table 4.10), GES is not bad compared with Notears or

Notearslowrank but it s still a lot worse than PC, ICALiNGAM, and DirectLiNGAM. Most importantly, GES failed to estimate the true graph on 8 datasets and the other 5 algorithms can run on them with no problem.

Based on the data mentioned above, it seems GES undeformed in comparison to the other algorithms tested in this experiment. However, it needs to be mentioned that this could be a potential implementation issue of gCastle [5] that has been used in this experiment.

[5]gCastle: https://github.com/huawei-noah/trustworthyAI/tree/master/gcastle



(a) Normalized_SHD

(b) G-Score

(c) F1

Figure 4.5: Performance in scatter plots

	3 hidden variables	No hidden variables
count	20	20
mean	0.34	0.40
std	0.15	0.11
min	0.11	0.14
50%	0.31	0.38
max	0.58	0.55

(a) DirectLiNGAM

	3 hidden variables	No hidden variables
count	20	20
mean	0.49	0.50
std	0.11	0.08
min	0.27	0.36
50%	0.50	0.49
max	0.69	0.68

(b) ICALiNGAM

	3 hidden variables	No hidden variables
count	20	20
mean	0.40	0.44
std	0.10	0.10
min	0.18	0.26
50%	0.41	0.42
max	0.58	0.65

(c) PC (stable)

	3 hidden variables	No hidden variables
count	20	20
mean	0.30	0.36
std	0.07	0.07
min	0.19	0.25
50%	0.31	0.34
max	0.43	0.47

(d) GES

Table 4.3: Performance of ICALiNGAM, DirectLiNGAM, GES, and PC (stable) on datasets with and without hidden variables.

	statistic	p-value	result
PC with hidden variables	1.36	0.51	normal
PC without hidden variables	1.13	0.57	normal
GES with hidden variables	0.09	0.96	normal
GES without hidden variables	1.21	0.55	normal
ICALiNGAM with hidden variables	0.23	0.89	normal
ICALiNGAM without hidden variables	0.52	0.77	normal
DirectLiNGAM with hidden variables	1.52	0.47	normal
DirectLiNGAM without hidden variables	1.75	0.42	normal

Table 4.4: Normality test on vICALiNGAM, DirectLiNGAM, GES, and PC (stable) on datasets with and without hidden variables

algorithm	p-value	result
PC	0.24	No difference on datasets with and without hidden variables
GES	0.01	Performance is better on datasets without hidden variables
ICALiNGAM	0.71	No difference on datasets with and without hidden variables
DirectLiNGAM	0.12	No difference on datasets with and without hidden variables

Table 4.5: T-test of ICALiNGAM, DirectLiNGAM, GES, and PC (stable) on datasets with and without hidden variables

Name	Variable Number	Observation Number	Remark
Network1_amp	5	500	60 variants
Network2_amp	5	500	60 variants
Network3_amp	5	500	60 variants
Network4_amp	10	500	60 variants
Network5_amp	5	500	60 variants
Network5_cont	5	500	60 variants
Network5_cont_p3n7	5	500	60 variants
Network5_cont_p7n3	5	500	60 variants
Network6_cont	8	500	60 variants
Network7_amp	6	500	60 variants
Network7_cont	6	500	60 variants
Network8_amp_amp	8	500	60 variants
Network8_amp_cont	8	500	60 variants
Network8_cont_amp	8	500	60 variants
Network9_amp_amp	9	500	60 variants
Network9_amp_cont	9	500	60 variants
Network9_cont_amp	9	500	60 variants

Table 4.6: Datasets used in sensitivity experiment

	PC (stable)	GES	ICALiNGAM	DirectLiNGAM	Notears	Notearslowrank
count	1071.00	1071.00	1071.00	1071.00	1071.00	1071.00
mean	0.69	0.67	0.73	0.78	0.70	0.70
std	0.14	0.15	0.11	0.10	0.10	0.11
min	0.19	0.25	0.39	0.42	0.40	0.35
50%	0.70	0.67	0.72	0.79	0.70	0.70
max	1.00	1.00	1.00	1.00	1.00	1.00

Table 4.7: Summary of performance on Normalized_SHD

	PC (stable)	GES	ICALiNGAM	DirectLiNGAM	Notears	Notearslowrank
count	1071.00	1071.00	1071.00	1071.00	1071.00	1071.00
mean	0.14	0.09	0.23	0.40	0.12	0.12
std	0.22	0.17	0.22	0.22	0.18	0.18
min	0.00	0.00	0.00	0.00	0.00	0.00
50%	0.00	0.00	0.20	0.40	0.00	0.00
max	1.00	0.86	0.88	0.88	0.83	0.88

Table 4.8: Summary of performance on G-Score

	PC (stable)	GES	ICALiNGAM	DirectLiNGAM	Notears	Notearslowrank
count	1071.00	1071.00	1071.00	1071.00	1071.00	1071.00
mean	0.39	0.46	0.17	0.08	0.25	0.29
std	0.23	0.29	0.19	0.16	0.23	0.25
min	0.00	0.00	0.00	0.00	0.00	0.00
50%	0.33	0.40	0.12	0.04	0.20	0.21
max	1.33	2.33	1.33	1.67	2.00	1.67

Table 4.9: Summary of performance on False Positive Rate

	PC (stable)	GES	ICALiNGAM	DirectLiNGAM	Notears	Notearslowrank
count	1071.00	1071.00	1071.00	1071.00	1071.00	1071.00
mean	0.51	1.29	0.50	0.19	36.81	112.62
std	0.72	1.31	0.44	0.13	45.12	117.91
min	0.01	0.09	0.02	0.04	5.76	10.29
50%	0.22	0.68	0.51	0.16	26.10	75.62
max	3.76	5.90	1.84	0.68	460.65	1194.83

Table 4.10: Summary of performance on runtime

5. Conclusion and future work

In this chapter, the conclusions drawn from the conducted experiments are presented. Following this, the challenges and critiques are discussed. Lastly, potential avenues for future research are suggested in order to further advance the topic at hand.

5.1 Conclusion

It has been demonstrated that low varsortability, such as 0.21, can be attained on synthetic datasets. In the context of synthetic dataset generation, it is recommended that varsortability be maintained at levels lower than 0.75, as indicated by the results presented in the associated table 4.1 since real datasets were found to exhibit a maximum varsortability of 0.75.

Algorithms that rely on the assumption of causal sufficiency may exhibit similar performance on datasets that either with or without hidden variables, provided that the number of such variables is minimal, for example, three or fewer.

Algorithms that assume acyclicity may display instability in their performance when applied to datasets containing cycles. Specifically, the estimated true graphs resulting from such algorithms may exhibit considerable variation, even among datasets that belong to the same group, which means that they were generated under the same condition, and share the same true graph.

With regards to the general performance of tested algorithms for causal discovery, it has been observed that GES underperformed significantly in comparison to other algorithms. As such, it may not be regarded as the optimal choice for causal discovery.

5.2 Challenges and critiques

The major causal discovery algorithm library used in this experiment, gCastle [1], has implementation issues. For example, GAE, this algorithm has both CPU and

[1] https://github.com/huawei-noah/trustworthyAI/tree/master/gcastle

GPU versions in gCastle. The CPU version works fine but for the GPU version, some variables are not migrated to the GPU properly thus it's unusable. Collecting and cleaning datasets for causal discovery is hard. Firstly, the sources of datasets are often scattered, thus efforts to locate and consolidate the necessary data are needed. Secondly, collected datasets frequently suffer from inconsistencies, requiring laborious manual cleaning. The biggest challenge is the limitation of computing resources. Some of the datasets have more than 500 or even 800 variables which require extra computing power to process.

As for the critiques, it needs to be mentioned that structural intervention distance(SID), an important evaluation metric in causal discovery, is not included in this experiment. Other than that, no R packages were used in this experiment which is a pity because a lot of useful libraries in the field of causal discovery were in R. As mentioned in the challenges, because of the limitation of computing resources, not all 1795 datasets were used in the experiment. Some of them have a large number of variables and it is challenging to run algorithms on them with such limited computing resources. The number of algorithms used in the experiment is also limited because almost half the integrated algorithms in gCastle are gradient-based so GPU is needed. However, GPU is not always available with limited computing resources.

5.3 Outlook

For future work, integrating more algorithms from diverse libraries, such as bnlearn[2], pcalg[3], and trilearn[4]. is necessary. The inclusion of a broader spectrum of algorithms, not solely limited to Python, but also R or Matlab, would enable the cross-comparison of algorithms implemented across diverse programming languages and libraries. Furthermore, it holds significant value to undertake hyperparameter tuning to enhance the performance of the algorithms. This approach would ensure that the algorithms are not merely running on their default parameters, but rather, are optimized through fine-tuning to attain improved performance. In addition, generating own synthetic datasets can serve as a valuable supplement to existing datasets for causal discovery.

[2]https://www.bnlearn.com/
[3]https://cran.r-project.org/web/packages/pcalg/index.html
[4]https://github.com/felixleopoldo/trilearn

Bibliography

[1] Vineet K. Raghu, Allen Poon, and Panayiotis V. Benos. *Evaluation of Causal Structure Learning Methods on Mixed Data Types*. https://proceedings.mlr.press/v92/raghu18a.html.

[2] Neville Kenneth Kitson et al. *A survey of Bayesian Network structure learning*. https://arxiv.org/ftp/arxiv/papers/2109/2109.11415.pdf.

[3] Clark Glymour, Kun Zhang, and Peter Spirtes. *Review of causal discovery methods based on graphical models*. https://www.frontiersin.org/articles/10.3389/fgene.2019.00524/full.

[4] Amanda Gentzel, Dan Garant, and David Jensen. *The case for Evaluating causal models using intervential measures and empirical data*. https://arxiv.org/pdf/1910.05387.pdf.

[5] Felix L. Rios, Giusi Moffa, and Jack Kuipers. *Benchpress: a scalable and versatile workflow for benchmarking structure learning algorithms for graphical models*. 2021. arXiv: 2107.03863 [stat.ML].

[6] Christina Heinze-Deml, Marloes H. Maathuis, and Nicolai Meinshausen. *Causal Structure Learning*. https://https://arxiv.org/abs/1706.09141.

[7] Christopher Bishop et al. *Elements of Causal Inference*. http://web.math.ku.dk/~peters/jonas_files/ElementsOfCausalInference.pdf.

[8] Alexander G. Reisach, Christof Seiler, and Sebastian Weichwald. "Beware of the Simulated DAG! Causal Discovery Benchmarks May Be Easy to Game". In: *Advances in Neural Information Processing Systems* 34 (2021).

[9] Shohei Shimizu et al. *DirectLiNGAM: A Direct Method for Learning a Linear Non-Gaussian Structural Equation Model*. https://www.jmlr.org/papers/volume12/shimizu11a/shimizu11a.pdf.