Member-only story

Causal Discovery

Learning causation from data using Python

Shaw Talebi

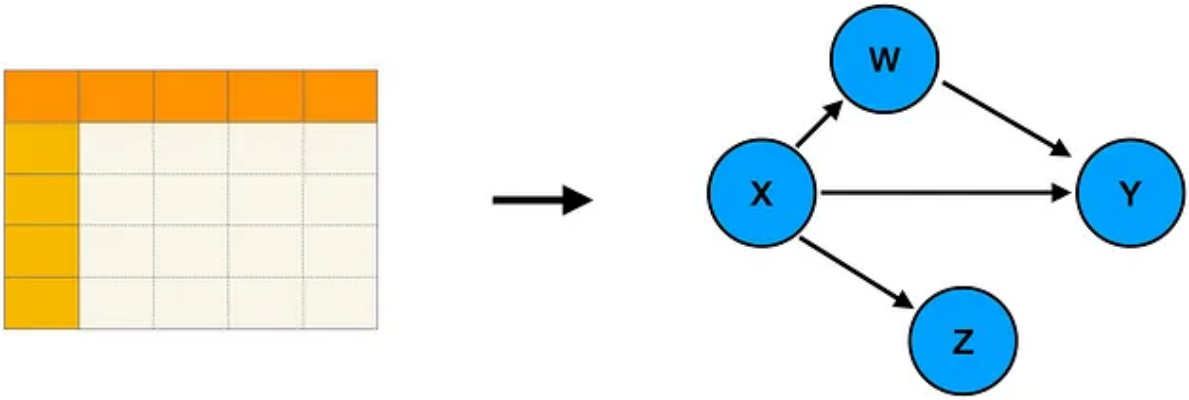
Published in Towards Data Science · 9 min read · Oct 26, 2021

This is the final post in a series of three on causality. In previous posts, the "new science" [1] of **causality** was introduced, and the topic of **causal inference** was discussed. The focus of this article is a related idea, **causal discovery**. I will start with a description of causal discovery, sketch how it works, and conclude with a concrete example using Python.

What is Causal Discovery?

In the **previous post**, I discussed **causal inference**, which aims to **answer questions involving cause and effect.** Although causal inference is a powerful tool, it *requires* a key. Namely, it requires a causal model. Often, in the real world, however, we may not be sure about which variables cause each other. This is where causal discovery can help.

Causal discovery aims to **infer causal structure from data.** In other words, given a dataset, *derive* a causal model that describes it.



Big picture goal of causal discovery: translate data into a causal model. Image by author.

How does it work?

There is no standard causal discovery method out there. In fact, there is a wide assortment of techniques available, where any two may have little resemblance to each other. My goal here is to sketch common themes across different algorithms and give a flavor of some key ideas.

As I mentioned in the **first post** of the series, causal discovery is an example of an **inverse problem.** This is like predicting the shape of an ice cube based on the puddle it left on the kitchen counter [10].

Clearly, this is a hard problem since any number of shapes could generate the same puddle. Connecting this to causality, the puddle of water is like statistical associations embedded in data, and the ice cube is the like underlying causal model.

4 Common Assumptions of Causal Discovery

The usual approach to solving inverse problems is to make assumptions about what you are trying to uncover. This narrows down the possible

solutions and hopefully makes the problem solvable.

There are four common assumptions made across causal discovery algorithms. A nice discussion of these assumptions can be found in reference [3].

1. **Acyclicity** — causal structure can be represented by DAG (G)
2. **Markov Property** — all nodes are independent of their non-descendants when conditioned on their parents
3. **Faithfulness** — all conditional independences in true underlying distribution *p* are represented in G
4. **Sufficiency** — any pair of nodes in G has no common external cause

Although these assumptions help narrow down the number of possible models, they do not fully solve the problem. This is where a few tricks for causal discovery are helpful.

3 Tricks for Causal Discovery

As mentioned earlier, no single method for causal discovery dominates all others. Although most methods use the assumptions above (perhaps even more), the details of different algorithms can vary tremendously.

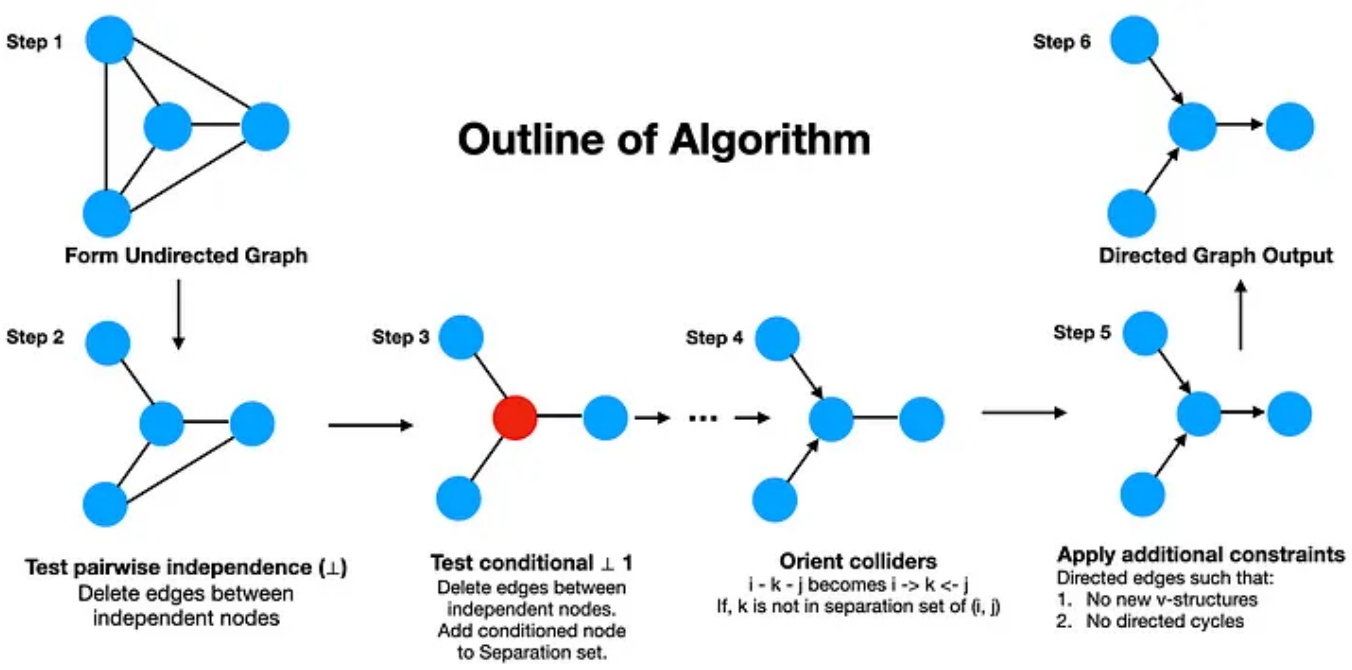
A taxonomy of algorithms based on the following tricks is given in the figure below. By no means is this an exhaustive overview. Rather, this is a collection of algorithms that I have come across from the cited references [2, 4, 5].

Trick	Algorithm
Conditional Independence Testing (i.e. Constraint-Based)	PC Fast Causal Inference (FCI) Inductive Causation (IC)
Greedy Search of DAG Space (i.e. Score-Based)	Greedy Equivalence Search (GES) Greedy Interventional Equivalence Search (GIES) Concave Penalized Coordinate Descent with Reparametrization (CCDr)
Exploiting Asymmetry	Linear Non-Gaussian Acyclic Model (LiNGAM) Nonlinear Additive Noise Model Post-nonlinear Causal Model (PNL) Granger Causality
Hybrid	Structural Agnostic Modeling (SAM) Causal Additive Model (CAM) Causal Generative Neural Network (CGNN)

A narrow taxonomy of causal discovery methods based on [2, 4, 5]. Image by author.

Trick 1: Conditional Independence Testing

One of these earliest causal discovery algorithms is the **PC algorithm** named after its authors Peter Spirtes and Clark Glymour. This algorithm (and others like it) use the idea that **two statistically independent variables are not causally linked.** The PC algorithm is illustrative of this first trick. An outline of the algorithm is given in the figure below. For more details on the PC algorithm, I'd recommend the discussions in references [2] & [6].



Outline of PC algorithm [2, 6]. Image by author.

The first step is to form a fully connected, undirected graph using every variable in the dataset. Next, edges are deleted if the corresponding variables are independent. Then, connected edges undergo conditional independence testing e.g. independence test of the bottom and far right node conditioned on the middle node in the figure above (step 2).

If conditioning on a variable kills the dependence, that variable is added to the Separation set for those two variables. Depending on the size of the graph, conditional independence testing will continue (i.e. condition on more variables) until there are no more candidates for testing.

Next, colliders (i.e. $X \rightarrow Y \leftarrow Z$) are oriented based on the Separation set of node pairs. Finally, the remaining edges are directed based on 2 constraints: 1) no new v-structures, and 2) no directed cycles can be formed.

Trick 2: Greedy Search of Graph Space

There are three main elements to this trick: a **graph**, a **graph space**, and a **greedy search**.

A **graph** is a mathematical construction consisting of vertices (circles) that are connected by edges (lines). Causal models typically use a special kind of graph called a directed acyclic graph (DAG). Graphs and DAGs were discussed in the **first post**. Briefly, a DAG is a kind of graph where edges between vertices have arrowheads and no cycles exist.

A **graph space** is **a collection of graphs**. This is just a fancy way to formalize that there are many possible graphs for a given number of vertices and edges. For example, a DAG with 2 vertices and 1 edge could take the forms: A → B or B → A. Notice, this is another take of the ice cube inverse problem from before. However, instead of multiple ice cubes possibly generating the same puddle of water, multiple DAGs can have the same number of nodes.

Finally, a **greedy search** is a way to navigate a space such that you always **move in a direction that *seems* most beneficial based on the local surroundings**. It's like being lost in a forest and trying to get out by only moving toward open areas. The problem with this type of approach is you are not guaranteed to find the best path through the space, as can be seen in the figure below.

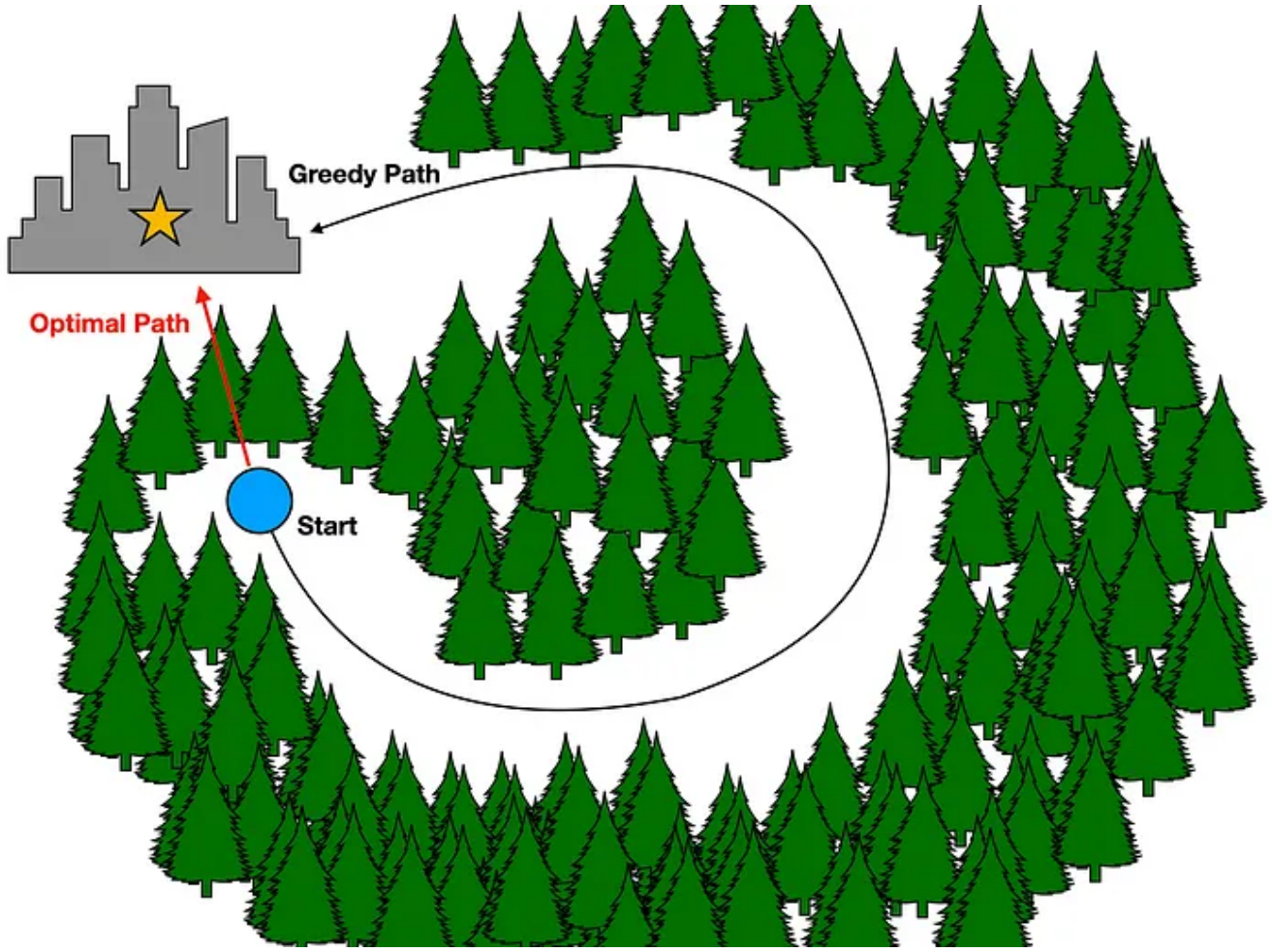


Illustration of how a greedy search can go wrong. Here, a greedy search of moving toward open areas when trying to get out of a forest will lead to a sub-optimal path. Image by author.

Although greedy searches cannot guarantee an optimal solution, for most problems, the space of possible DAGs is so big that finding a *true* optimal solution is intractable.

The **Greedy Equivalence Search (GES)** algorithm uses this trick. GES starts with an empty graph and iteratively adds directed edges to maximize the improvement in a model fitness measure (i.e. score). An example score is the Bayesian Information Criterion (BIC) [2].

Trick 3: Exploiting Asymmetries

As stated in the **first post**, asymmetry is a fundamental property of causality. A could cause B, but B may not cause A. There is a large space of algorithms

that leverage this idea to select between causal model candidates. I will categorize these asymmetries into **three common flavors: time, complexity, and functional**. Before describing these asymmetries, it is important to note that they are not mutually exclusive. Meaning algorithms can mix and match different flavors.

Time asymmetry is natural to us. It's the idea that **causes happen before effects**. This sits at the core of ideas such as **Granger causality**. Although Granger causality is not sufficient to claim causality, it leverages the idea that causes precede effects. It does this in the two variable case (e.g. X and Y), by quantifying the gain in predicting Y given past information of Y and X, as opposed to past information of Y alone [5].

Complexity asymmetry follows Occam's razor principle that **simpler models are better**. In other words, if you have a handful of candidate models, this idea says to pick the simplest one. One way of quantifying simplicity (or complexity) is the Kolmogorov Complexity.

Functional asymmetry assumes **models that better fit** a relationship **are better candidates**. For example, given two variables X and Y, the **nonlinear additive noise model (NANM)** performs a nonlinear regression between X and Y, e.g. $y = f(x) + n$, where n = noise/residual, in both directions. The model (i.e. causation) is then accepted if the potential cause (e.g. x) is independent of the noise term (e.g. n). More details on NANM can be found in reference [7].

Example: Causal Discovery with Census Data

We now turn away from theory and toward a concrete example. The example uses the Causal Discovery Toolbox, a Python library for causal discovery [4]. In this example, we look at census data from the same source as before [8].

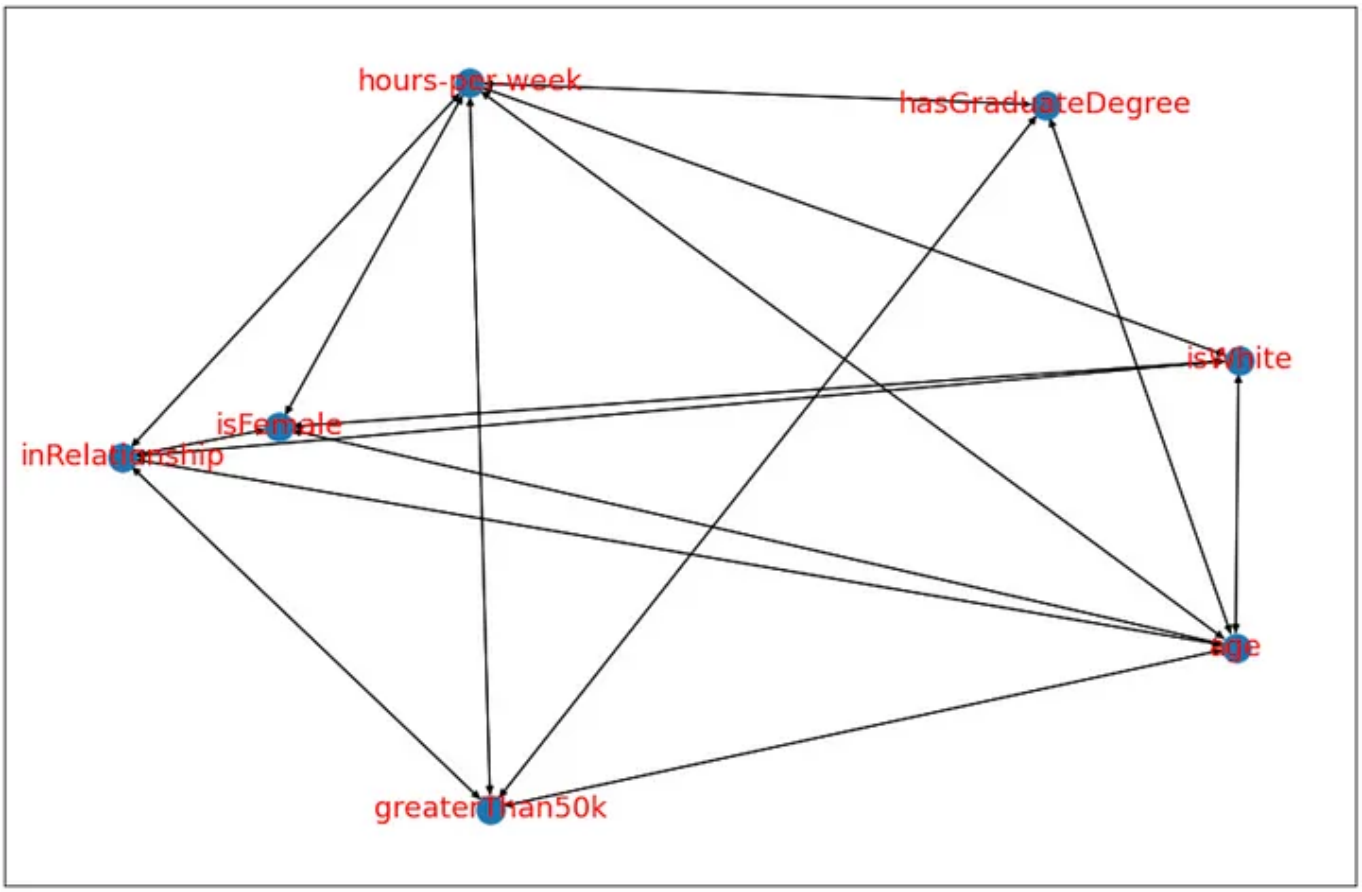
In the previous example, we assumed that income has two causes: age and education. However, *how can we be sure this assumption holds water?* In this example, we explore alternative causal models that both include more variables and make use of a data-driven approach: causal discovery. Example code and data can be found in the GitHub repo.

First, we load the necessary libraries and data. If you do not have the libraries, check out the requirements.txt in the repo. Data is also in the repo.

```
# import libraries
import pickle
import cdt
import networkx as nx
import matplotlib.pyplot as plt

# load data
df = pickle.load( open( "df_causal_discovery.p", "rb") )
```

Next, we need to construct a skeleton for our causal models. This is basically step 2 of the PC algorithm discussed earlier. In other words, pairwise independence tests are performed, and variables that are deemed statistically dependent receive a bi-directed edge.

```
# Get skeleton graph
# initialize graph lasso
glasso = cdt.independence.graph.Glasso()

# apply graph lasso to data
skeleton = glasso.predict(df)

# visualize network
fig = plt.figure(figsize=(15,10))
nx.draw_networkx(skeleton, font_size=18, font_color='r')
```



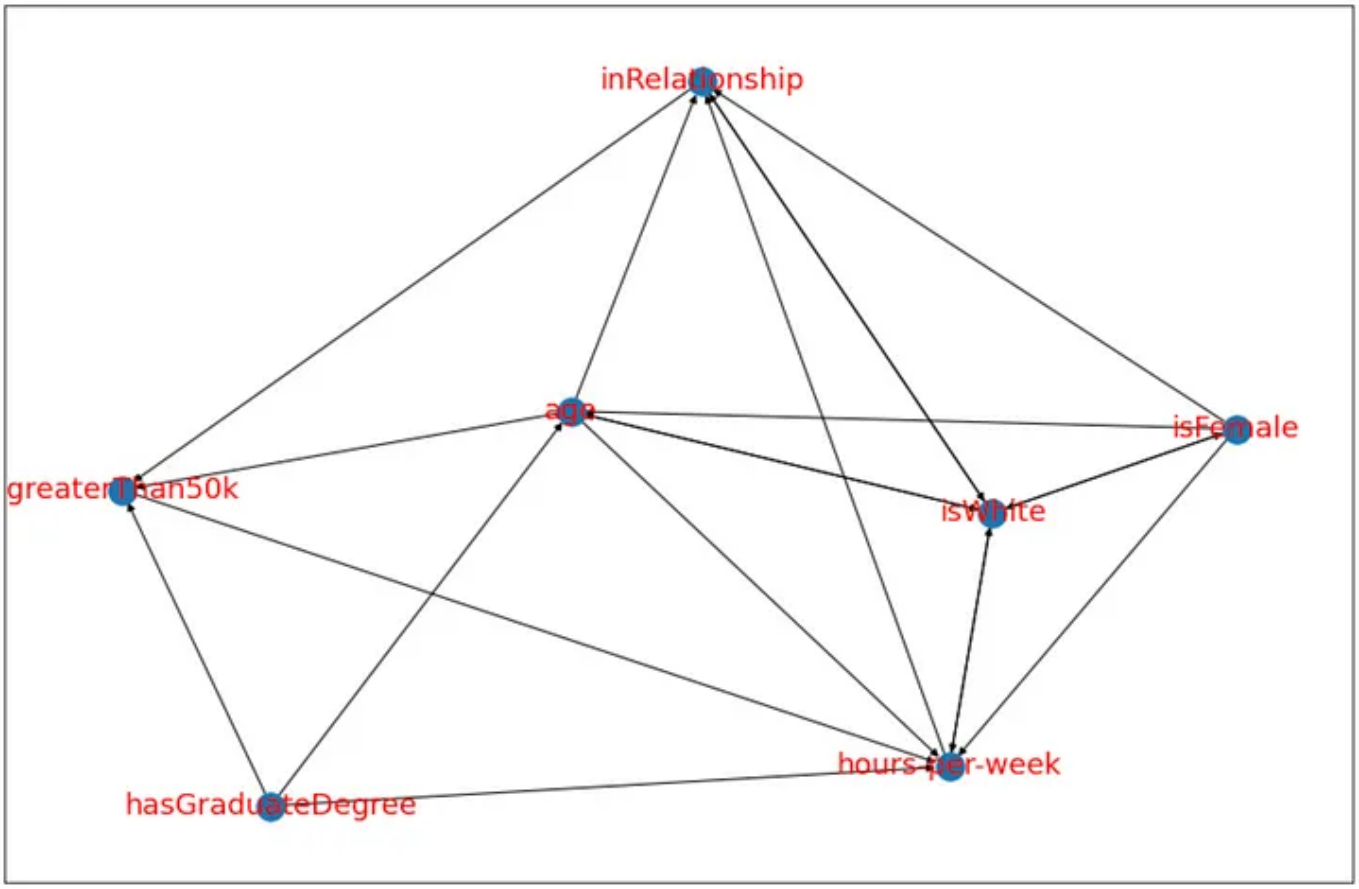
Skeleton graph. Image by author.

Notice we have 7 different variables. Before, we only considered age, education (hasGraduateDegree), and income (greaterThan50k). From the skeleton alone we can see statistical dependencies existing between more than just these 3 variables.

Now it's time for causal discovery. Here we try out 3 different algorithms: PC, GES, and LiNGAM.

```
# Use causal discovery to get causal models

# PC algorithm
model_pc = cdt.causality.graph.PC()
graph_pc = model_pc.predict(df, skeleton)

# visualize network
```

```
fig=plt.figure(figsize=(15,10))
nx.draw_networkx(graph_pc, font_size=18, font_color='r')
```

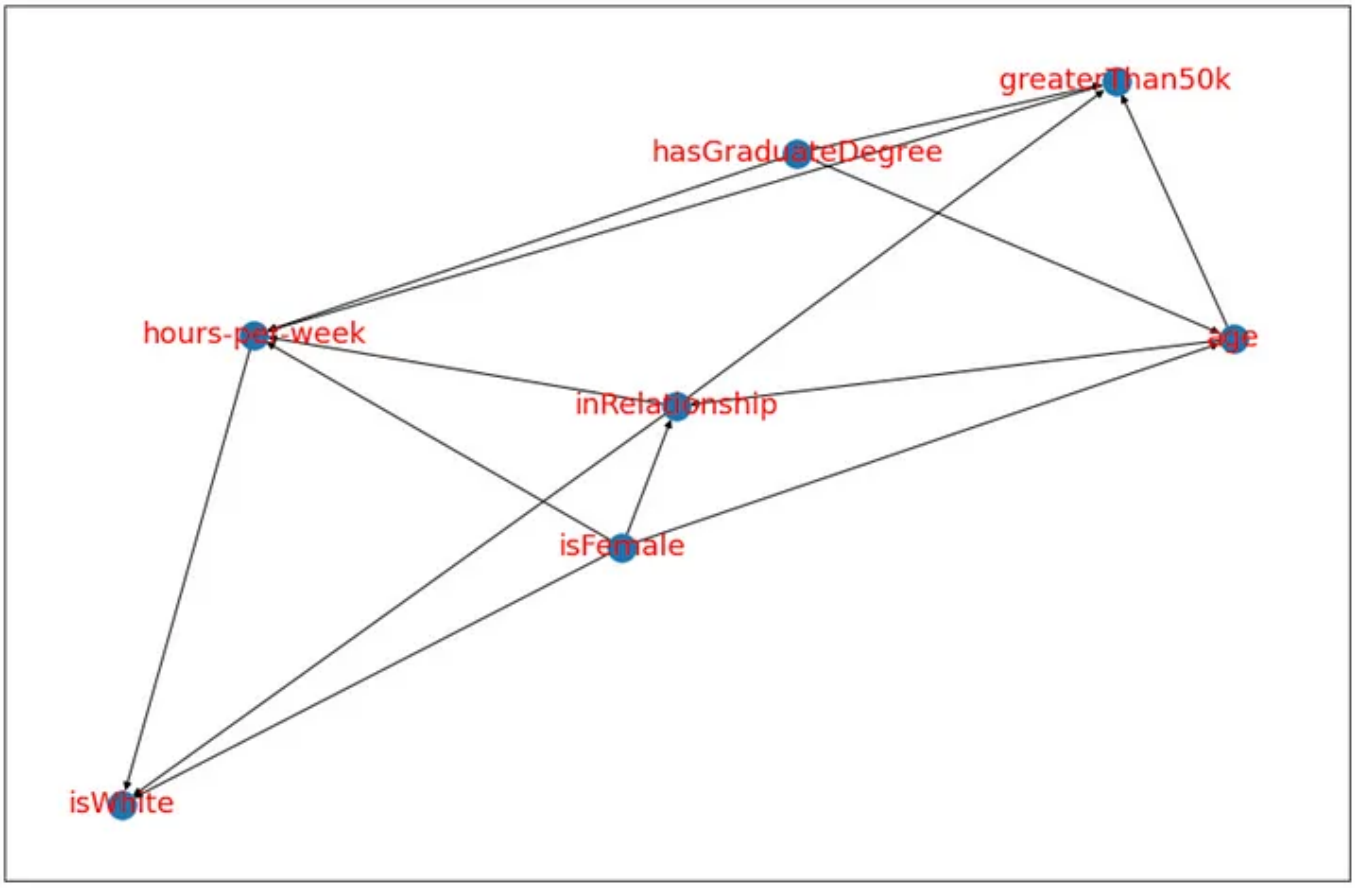


Output causal graph from PC algorithm. Image by author.

PC gives a somewhat sensible result. The simple 3 variable DAG we used in the last example is *almost* embedded in this graph. The arrow between age and hasGraduateDegree is flipped, which is questionable since this suggests having a graduate degree has a causal effect on one's age!

It also suggests additional causal relationships. Namely, hasGraduateDegree, greaterThan50k, age, and isFemale all have one-way causal impacts on hours-per-week, which itself has a causal effect on inRelationship. Next, we look at GES.

```
# GES algorithm
model_ges = cdt.causality.graph.GES()
graph_ges = model_ges.predict(df, skeleton)

# visualize network
fig=plt.figure(figsize=(15,10))
nx.draw_networkx(graph_ges, font_size=18, font_color='r')
```

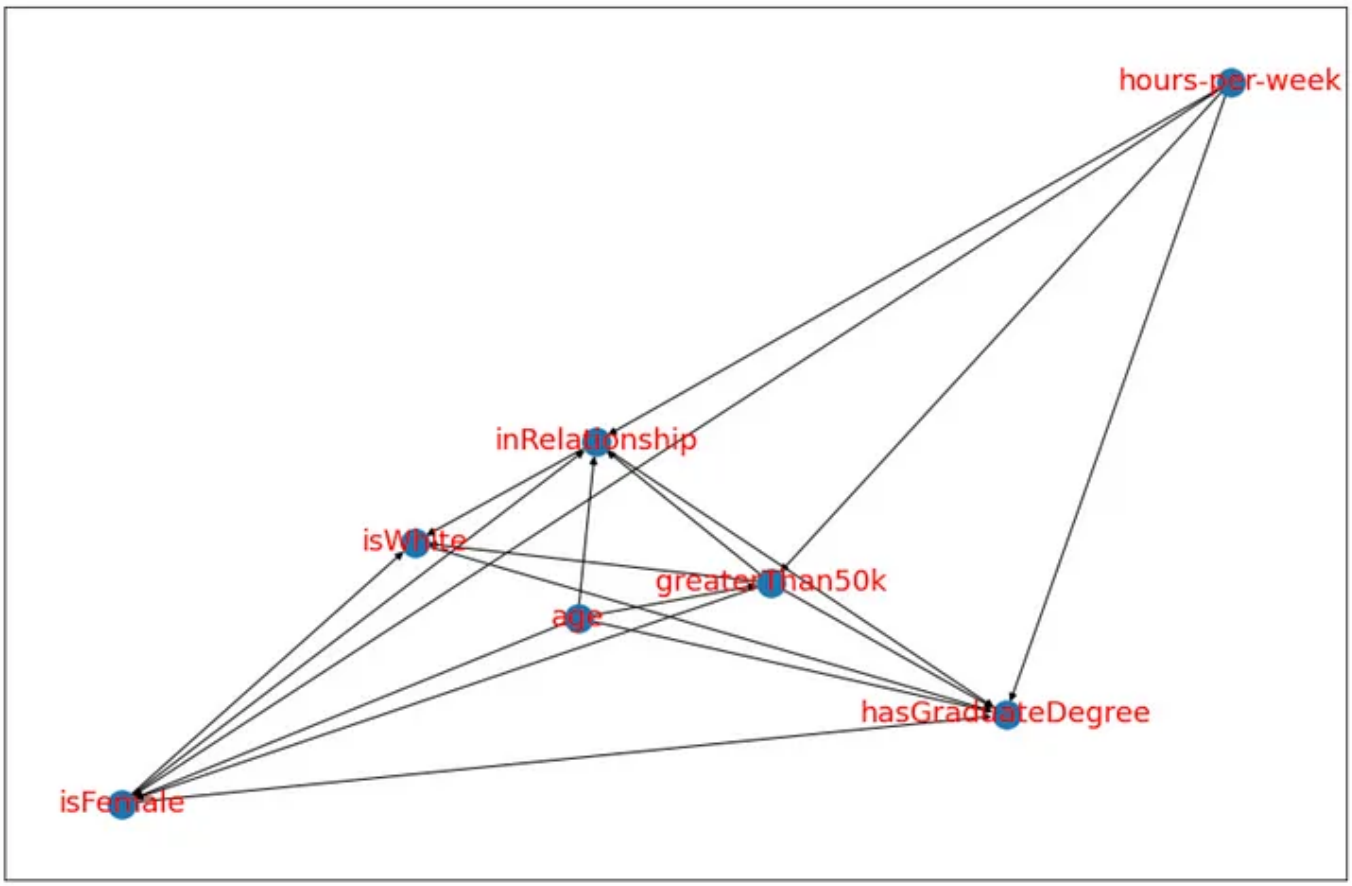


Output causal graph from GES algorithm. Image by author.

This graph seems to mostly agree with the PC result. Notable differences are the previously bi-directed edges stemming from isWhite are now uni-directed, which makes sense recalling how GES works.

Finally, we apply LiNGAM to the data, which uses what I called functional asymmetry between linear models to infer causation [9].

```
# LiNGAM Algorithm
model_lingam = cdt.causality.graph.LiNGAM()
graph_lingam = model_lingam.predict(df)

# visualize network
fig=plt.figure(figsize=(15,10))
nx.draw_networkx(graph_lingam, font_size=18, font_color='r')
```



Output causal graph from LiNGAM algorithm. Image by author.

This result differs significantly from the previous two. Many of the edge directions have been flipped, such as the one between **greaterThan50k** and **hasGraduateDegree**. The poor performance is likely due to a violation of the algorithm's assumption of linear effects. Most variables here are binary, thus linear structural equations are likely inappropriate.

Conclusion

comprehensive review of causal discovery in a short blog post, I have tried to curate a list of good references in the **Resources** section below. Despite being young, causal discovery is a promising field that may help bridge the gap between machine and human *knowledge*.

👉 **More on Causality:** Causal Effects Overview | Causality: Intro | Causal Inference | Causal Discovery

Resources

Connect: My website | Book a call | Ask me anything

Socials: YouTube 📽️ | LinkedIn | Twitter

Support: Buy me a coffee ☕

[1] The Book of Why: The New Science of Cause and Effect by Judea Pearl

[2] Review of Causal Discovery Methods Based on Graphical Models. https://doi.org/10.3389/fgene.2019.00524

[3] ML based causal discovery: arXiv:1803.04929 **[stat.ML]** (good primer)

[4] Causal Discovery Toolbox Documentation: https://fentechsolutions.github.io/CausalDiscoveryToolbox/html/index.html

[5] Granger causality: https://doi.org/10.1016/0165-1889(88)90055-3

[6] Causal Inference with the R Package pcalg: http://dx.doi.org/10.18637/jss.v047.i11 (good primer)

[7] Nonlinear additive noise models paper

[8] UCI Machine Learning Repository: data source

[9] LiNGAM paper

[10] Example taken from Antifragile by Nassim Nicholas Taleb (paid link)