

Causal DAG Extraction from 3 Short Stories and 3 Movie Scripts

Robert R. Tucci
tucci@ar-tiste.com

April 24, 2023

Abstract

We improve a previously proposed algorithm for doing causal DEFT (DAG Extraction from Text), and then we apply the new algorithm to 2 usecases: 3 short stories by P.G. Wodehouse and 3 movie scripts by Pixar/Disney. The software used to accomplish this endeavor is called “Mappa Mundi” and is available as open source at [GitHub](#).

“If humans were so good at causal inference, religion would not exist.”

- Yann LeCun, *Ref.[4]*

“How much of human knowledge is captured in all the text ever written? To which my answer is: not much. ”

- Yann LeCun, *Ref.[3]*

1 Introduction

In this paper, I improve an algorithm for doing causal DEFT (DAG Extraction from Text) that was first proposed in Ref.[7] I then apply the new algorithm to 2 usecases:

1. 3 short stories by P.G. Wodehouse (the text from these was obtained from the Project Gutenberg website [2])
 - Bill the Bloodhound
 - Extricating Young Gussie
 - Wilton’s Holiday
2. 3 movie scripts by Pixar/Disney. (the text for these was obtained from the IMSDb website Ref.[1])¹
 - Toy Story
 - Up
 - WALL-E

The Python software that was used to accomplish this endeavor is called Mappa Mundi. It is open source and available at Github (Ref.[8]).

So what is Mappa Mundi good for? The goal of DEFT in general and Mappa Mundi in particular is to create a directory of DAGs (“DAG atlas”). Conjecturing a DAG is always the first step in doing Judea Pearl’s causal inference (CI).² Once a DAG is available, one can use it to do Pearl’s 3 rungs of CI. For example, my free open source software SCuMpy (Ref.[9]) can be used to do all 3 rungs of CI with linear

¹The Mappa Mundi repo at GitHub contains a Python script called `downloading.py` that uses the BeautifulSoup Python package to scrape all the 1100+ movie scripts available (about 230 MB) at the IMSDb website. My original intention was to apply my algorithm to all of those movie scripts. However, due to lack of hardware resources, I had to settle for just 3 movie scripts.

²Pearlian CI is described in detail in Pearl’s *The Book of Why* (Ref.[5]) and in my free, open source book *Bayesuvius* (Ref. [6]).

SCM (Structural Causal Model, a type of DAG)³, including SCM with feedback loops and hidden variables.

As I explained in my previous paper Ref.[7], the scientific method (SM) looks for causation, not correlation. Pearl CI is the gold standard theory for distinguishing between correlation and causation. Hence, the SM and CI are closely related. CI can be viewed as an application of SM wherein the DAG is the hypothesis part of the SM, what we want to prove or disprove. DEFT provides DAG hypotheses. For example, DEFT could be used to discover causal DAGs that indicate pathways to diseases.

At the end of this paper, I describe possible ways of improving Mappa Mundi using LLMs (Large Language Models such as ChatGPT).

2 Algorithm Overview

The algorithm proposed in this paper can be applied to a broad range of texts. The only constraint is that those texts do “story-telling” in a chronological order. That is why I decided to use movie scripts, because movie-scripts usually do story-telling in a chronological order (except when they do flashbacks or time travel, but that is fairly uncommon in movies.)

For example, algorithm would not work well if applied the corpus of science papers at arXiv, because those don’t do chronological story-telling. On the other hand, it might work well if applied to a corpus of (time stamped) lab notebooks by one or more experimental scientists. It might also work well on a corpus of time-stamped logs maintained by an individual trying to figure out the cause of a disease that ails him/her.

Note also that this algorithm would work well on a corpus of videos or movies that do chronological story telling. This would require that some human or AI narrated the movie/video as the action happens. Such “in-time” movie and video narration, often referred to as AD (audio description) (Ref.[10]), and also closed captioning, are becoming increasingly widespread in the movie, TV and internet video streaming industries. In certain cases, they are mandated by laws (like the CVAA, Twenty-First Century Communications and Video Accessibility Act of 2010) that address the needs of the visually impaired.

The essence of my algorithm is simple to describe. Given a set of N movie scripts (or short stories), it compares all possible pairs movies. Hence, it makes $(N^2 - N)/2$ comparisons between 2 movies.

Before comparing movies, each movie script is simplified as follows. Each sentence is divided into clauses, and each clause is simplified by removing “stop-words” and other excess baggage from it. Then we define a node of a DAG for each of the simplified clauses. We define a DAG, possibly not connected, for each movie.

To compare two movies 1 and 2, we compare every node `node1` in movie 1

³DAG = Directed Acyclic Graph, SCM = Structural Causal Model

with every node **node2** in movie 2. To compare two nodes (**node1**, **node2**), we use what is called, in the NLP (Natural Language Processing) community, a **similarity between two sentences**. If the similarity between **node1** and **node2** exceeds a certain threshold that we call **SIMI_THRESHOLD**, then we store that node pair and its similarity in a dictionary with a node pair as key and its similarity as value. Call this dictionary **nd1_nd2_bridges**. We say that there is a **bridge** between **node1** and **node2** if (**node1**, **node2**) is contained in the keys of **nd1_nd2_bridges**.

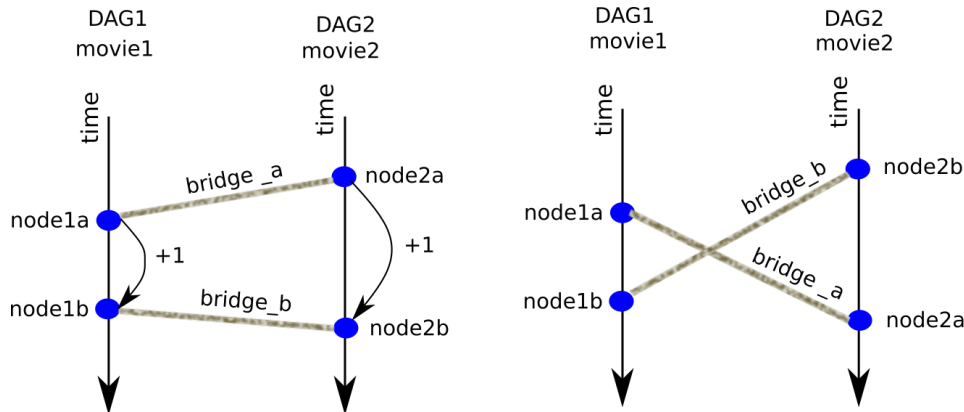


Figure 1: Bridges span two DAGs (i.e., movies). We consider 2 possibilities: bridges *a* and *b* cross, or they don't.

Next we consider every pair $\{a, b\}$ of bridges. Suppose bridge *a* connects node **node1a** in movie 1 to node **node2a** in movie 2. Likewise, let bridge *b* connect **node1b** in movie 1 to node **node2b** in movie 2. Let **node1a.time** be the time at which **node1a** occurs and define **node1b.time**, **node2a.time**, and **node2b.time** similarly. Assume that **node1a.time** < **node1b.time**. Then there are two possibilities that we wish to consider. These 2 possibilities are illustrated in Fig.1. Either the bridges don't cross (i.e. **node2a.time** < **node2b.time**) or they cross (i.e. **node2b.time** < **node2a.time**).⁴ Let N_{rep} be the **number of repetitions of an arrow**. If bridges *a* and *b* cross, we do nothing.⁵ If they don't cross, we do the following for both DAG1 and DAG2. If an arrow between the earlier and latter of the two nodes doesn't already exist, we add one with $N_{rep} = 1$. If such an arrow already exists, we increase its N_{rep} by one.

That's basically the whole algorithm. At the end of it, we will have generated DAG1 for movie 1 and DAG2 for movie 2.

When drawing one of those DAGs with Mappa Mundi, one specifies a number **reps_threshold**. Only the arrows with N_{reps} larger than **reps_threshold** are drawn. The number N_{reps} for each arrow is drawn in the middle of the arrow.

Here is a simple argument for why this algorithm should work. Consider Fig.2. The figure depicts a DAG that expresses the fact that both shark attacks and

⁴We ignore cases where the 2 nodes in movie 1 or the 2 nodes in movie 2 occur at the same time.

⁵It's also possible to assign a penalty when the bridges cross, but we don't explore that option in this paper

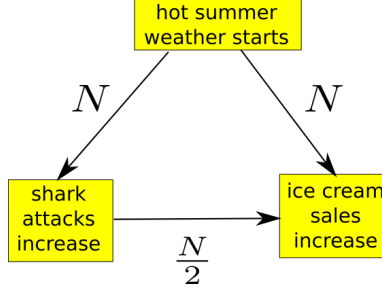


Figure 2: DAG that expresses the fact that both shark attacks and ice cream sales increase during the summer, because both are caused by hot summer weather.

ice cream sales increase during the summer, because both are caused by hot summer weather. Let

H = hot summer weather starts,

Sh = shark attacks increase,

IC = icecream sales increase.

If we compare this DAG to N other DAGs that contain these 3 nodes, then, since it is always true that summer precedes shark attack increases and ice cream sales increases, the two arrows emanating from the H node will have $N_{reps} = N$. On the other hand, we expect that half of the time, the Sh node will occur before the IC node, and half of the time it will happen after. Hence, $N_{reps} = \frac{N}{2}$ for the arrow $Sh \rightarrow IC$. If, when we draw this DAG, we set `reps_threshold` to be greater than $\frac{N}{2}$, then only the two causal arrows will be visible. The difference between N_{reps} for the causal and non-causal arrows (call it the **causal-noncausal repetition gap**) will grow ⁶ as $\frac{N}{2}$.

It’s important to point out that I expect this algorithm to produce good DAGs only after a large number N of DAGs (i.e., movie scripts or short stories) are compared. That’s because the repetition gap grows relatively slowly, just linearly in N . Since, due to lack of hardware resources, this paper only compares a minuscule $N = 3$, one can’t expect very dramatic causal revelations from this paper.

It’s also important to point out that every time a new movie script is added the list of the N already analyzed movie scripts, that new movie script must be compared to the preceding N movie scripts. If we imagine a robot watching a movie daily, then his daily dream time, if used solely for movie-DAG comparison, would grow linearly in time. At some point, it would take more than a day of dream time to compare today’s movie to all movies in its past. To keep his dream time constant, at 8 hours each day, the robot would have to compare today’s movie to a fixed number, say the most recent 365, in its past.

⁶This gap can probably be made to increase even faster if we penalize crossing bridges, but we won’t consider that in this paper.

3 Software Description

The full process for the 3 short stories is documented in the jupyter notebook

`jupyter_notebooks/navigating_short_stories.ipynb`

The full process for the 3 movie scripts is documented in the jupyter notebook

`jupyter_notebooks/navigating_m_scripts.ipynb`

For a detailed description, encompassing every method and variable used in the Mappa Mundi software, please consult the software’s Python “docstrings”. This section of the paper merely presents a brief overview of what you will find in those jupyter notebooks and docstrings.

The full process from raw data to DAG atlas is broken down in Mappa Mundi to performing the following steps. Most of my time programming was spent on the scripts that do pre-processing of the data (i.e., steps 2, 3 and 4 below). Pre-processing data is hard!, and not doing it is fatal to this algorithm. When evaluating the similarity between two clauses, even small misspellings can change that value substantially.

1. Data scraping using methods in `downloading_imsdb.py`.

This python script was only used for the movie scripts, not for the short stories. It scrapes all the movie scripts from the IMSDb website using the Python Package Beautiful Soup.

2. Cleaning using methods in `cleaning.py`

Here I remove contractions like “didn’t”, and replace unicode only symbols like curly quotes by straight ascii quotes, their closest ascii approximation.

For the case of movie scripts (but not for short stories), I also try to distinguish between dialog lines, and narration lines. The dialog lines are often indented with respect to the narration lines, but not always. In the case of Pixar/Disney, they don’t indent dialog. In cases where the movie script indents, the software gives the option to throw away all the dialog and keep only the narration.

Here I also use the software SpaCy⁷ to separate the sentences and return a file with only one sentence per line.

3. Spell-checking using methods in `spell_checking.py`

I discovered, to my chagrin, that spell-checking without any input from a human user, is very error prone, unless one takes context into consideration. LLMs can

⁷In the Python world, there are 2 general, dominant NLP libraries, SpaCy and NLTK (Natural Language Tool Kit). Mappa Mundi uses both. There is much overlap between the 2. In case of overlap, I tried to use the one that was fastest.

do contextual spell-checking, but this project did not use LLMs, since I don't have access to them. Instead of a contextual spell-checker, I used the non-contextual spell-checker `pyspellchecker` that tries to replace infrequent words by more frequent ones (a very risky and error prone approach). To diminish the risk, I constrained it in various ways so that it only makes very conservative corrections.

4. Simplifying using methods in `simplifying.py`

At this point, the file has only one sentence per line. Here, I use SpaCy to break each sentence into clauses. Then I simplify the clauses by removing stop-words, punctuation marks and other extra baggage. Then I replace each clause by its simplified version. Different clauses from the same sentence are put in the same line, separated by an asterisk. Some sentences are diminished to nothing after the simplification. Those sentences are replaced by a single asterisk.

5. Creating DAG atlas using methods in class `DagAtlas`

Everything up to this point has just been pre-processing of the data. Here, I finally implement the algorithm described in Section 2. The bottleneck and rate-determining-step for full process is calculating the similarity between 2 nodes. If the DAG1 for movie script (or short story) 1 has k_1 nodes, and DAG2 has k_2 nodes, then $k_1 k_2$ similarity calculations have to be made. For the short stories I considered, $k_1 k_2$ is typically on the order of $0.1M$. For the movies, it is typically $1.5M$. My computer is slow and old (I refuse to give specs so as not to embarrass myself). It can do about a million similarity calculations per hour. It can compare the 3 short stories in 20 minutes. I have't tried it, but I calculate that the 3 movie comparison would take $15(\frac{1}{3}) = 5$ hours.

For the similarity measure between nodes (i.e., sentences or clauses), I tried one method that uses NLTK+Wordnet, and another method that uses SpaCy+WordVec. The NLTK+WordNet method is, in my opinion, more logical, and it gives more reasonable distances. I also found it to be faster than the SpaCy+WordVec method.⁸

6. Visualizing DAGs using methods in class `Dag`.

This part is easy. I use `graphviz` to do it.

⁸The NLTK+WordNet method calculates the distance between synsets of nouns in node1 and node2. It also calculates the distance between synsets of verbs, adjectives and adverbs. (Synsets, i.e., synonym sets, are a sort of equivalence class of words.) Then it calculates from all those synset distances, a distance between the two nodes (i.e., clauses or sentences). The SpaCy+WordVec approach turns each word for the clause in node1 into a vector and then calculates the center of mass (CM) of those word vectors. It likewise calculates a CM vector for node2. Then it calculates the cosine of the angle between those two CM vectors. I personally don't think a CM vector is a good representation of a list of words.

4 Possible Improvements using LLMs

I believe the results of this study would improve dramatically if this were done with LLMs. LLMs could perform better the 3 data preprocessing steps (2-cleaning, 3-spell-checking and 4-simplifying). Cleaning and spell-checking could be done by the LLM taking context into consideration, and the output would be in the flawless English that ChatGPT answers questions with. As to step 4-simplifying, rather than using the crude method of breaking each sentence into clauses and calling each clause a node, one could, for example, ask the LLM to summarize each sentence (or a few consecutive sentences) as a list of simple declarative sentences. To demonstrate that LLMs can already performed this task admirably, did the following test. I gave ChatGPT the beginning narrative paragraph of the movie WALL-E, and I asked ChatGPT to summarize the paragraph as a list of simple declarative sentences. Here is the result:

PROMPT:

Consider the following quote: "A range of mountains takes form in the haze. Moving closer. The mountains are piles of TRASH. The entire surface is nothing but waste.

'..We're gonna find adventure in the evening air...'

A silhouetted city in the distance. What looks like skyscrapers turns into trash. Thousands of neatly stacked CUBES OF TRASH, stories high. Rows and rows of stacked cubes, like city avenues. They go on for miles."

Give an enumerated list summarizing this quote. Each list item must be a simple declarative sentence consisting of a subject phrase and a verb phrase. The list items must be in chronological order of occurrence in the quote.

REPLY:

1. A range of mountains appears in the haze.
2. The narrator moves closer.
3. The mountains are revealed to be piles of trash.
4. The entire surface is described as nothing but waste.
5. The narrator speaks of finding adventure in the evening air.
6. A silhouetted city appears in the distance.
7. The skyscrapers turn out to be made of trash.
8. Thousands of neatly stacked cubes of trash are seen, stories high.
9. Rows and rows of stacked cubes are seen, resembling city avenues.
10. The expanse of trash goes on for miles.

Each of these 10 sentences would make a superb node in a causal DAG. With nodes defined with such clarity, one could avoid much noise in the calculation of the similarity of 2 nodes.

References

- [1] Internet Movie Script Database (IMSDb). <https://imsdb.com/>.
- [2] Project Gutenberg website. <https://www.gutenberg.org>.
- [3] Yann LeCun. Twitter, all world text quote. <https://twitter.com/ylecun/status/1562137291845521408>.
- [4] Yann LeCun. Twitter, causal inference and religion quote. <https://twitter.com/ylecun/status/1577128801620070400>.
- [5] Judea Pearl and Dana Mackenzie. *The book of why: the new science of cause and effect*. Basic books, 2018.
- [6] Robert R. Tucci. Bayesuvius (book). <https://github.com/rrtucci/Bayesuvius/raw/master/main.pdf>.
- [7] Robert R. Tucci. Causal dag extraction from a library of books or videos/movies. <https://arxiv.org/abs/2211.00486>.
- [8] Robert R. Tucci. Mappa Mundi at github. https://github.com/rrtucci/mappa_mundi.
- [9] Robert R. Tucci. SCuMpy at github. <https://github.com/rrtucci/scumpy>.
- [10] Wikipedia. Audio description. https://en.wikipedia.org/wiki/Audio_description.