Enchanted to Learn You: Structure Learning in Bayesian Networks

Aryan Raj Dhawan

December 2024

1 Introduction

"Are you ready for it?"

This paper will involve an analysis of different structural learning algorithms for Bayesian Networks. Given a dataset with attributes $V = v_1, ..., v_n$, we aim to learn a structure for a Bayesian network that best represents the relationships between these attributes. The metrics with which such a "best fit" structure is determined varies according to the type of algorithm employed for structure learning. In general, there are 3 primary types of structure learning algorithms:

1. Constraint-Based
2. Score-Based
3. Hybrid

The aim of this paper is two-fold. First, the theoretical and mathematical foundations of a few key algorithms from each category will be explored - highlighting both the differences between each category and between algorithms within each category as well. Second, the algorithms that are explored in the first section will be applied to a real-world problem of building the basis for a music recommender system.

2 Constraint-Based Algorithms

2.1 A broad overview

"Your faithless love's the only hoax I believe in"

The broad idea of constraint-based algorithms is to find a **minimal I-map** for a distribution that satisfies a set of dependency (or independency) relationships. Prior to the discussion of the PC algorithm, a few key definitions must be introduced.

Definition 1. Given a distribution of random variables, an **I-Map** implies a subset of the conditional independence satisfied by the distribution (Huang). Recall that a **P-Map** represents the independency relationships exactly. A **minimal I-Map** is one such that the removal of any edge would cease its property of being an I-Map.

Definition 2. The **Markov Condition** holds for a directed-acylic graph $G = (V, E)$ and a probability distribution over the vertices $P(V)$, if every variable $v \in V$ is conditionally independent of its (graphical) non-descendents, given its graphical parents. Formally,

$$\forall X, Y \in V, \forall Z \subseteq V \setminus \{X, Y\} : (X \perp\!\!\!\perp_d Y \mid Z \implies X \perp\!\!\!\perp_p Y \mid Z).$$

Definition 3. The **faithfulness** assumption extends this idea, and states that there are no additional independencies in the probability distribution $P(V)$ other than those dictated by the graph G. Formally, we get the condition below.

$$\forall X, Y \in V, \forall Z \subseteq V \setminus \{X, Y\} : (X \not\perp\!\!\!\perp_d Y \mid Z \implies X \not\perp\!\!\!\perp_p Y \mid Z).$$

Hence, if G implies no separation (graphically) between variables X and Y, there is no statistical independence between X and Y in $P(V)$. Together, the faithfulness and Markov conditions holding implies that

$$\forall X, Y \in V, \forall Z \subseteq V \setminus \{X, Y\} : (X \perp\!\!\!\perp_d Y \mid Z \iff X \perp\!\!\!\perp_p Y \mid Z).$$

This signifies that the dependency relationships in the data precisely match those in the graph G. Thus, faithfulness is assumed to "prove" that the learned graph is correct (Huang)

2.2 The Peter Clark Algorithm

"Darling, I'm a nightmare dressed like a daydream"

The Peter-Clark algorithm consists of 2 main parts - (1) skeleton recovery and (2) v-structure recovery.

The PC algorithm starts by assuming a fully connected graph H. During the **skeleton recovery** phase, the algorithm iteratively tests conditional independence relationships between pairs of nodes given subsets of other nodes. When $u \perp\!\!\!\perp_p v \mid S$ is found for a pair (u, v) and subset S, the edge $u - v$ is removed, and S is stored as the separation set S_{uv}. This process continues until the graph's structure "stabilizes". Note that to determine whether there is a set separating u and v, we might search all 2^{n-2} subsets of $V/\{u, v\}$. Hence, the complexity for investigating each possible edge in the skeleton is $O(2^n)$ and hence the complexity for constructing the skeleton is $O(n^2 * 2^n)$, where $n = |V|$.

In the **v-structure recovery** phase, the algorithm identifies and orients v-structures using the separation sets. A v-structure is oriented if u and v are not conditionally independent given w but share a common neighbor w. More formally, for all separation sets S_{uv} we consider $w \in V/(\{A, D\} \cup S_{AD})$ and if such a w exists, orient $u - w - v$ as $u \to w \leftarrow v$. Note that in many cases

such a w may not exist, and thus the PC algorithm may return a **PDAG** - a partially directed acyclic graph - which contains some directed edges with the rest undirected (as they couldn't be oriented).

Thus, the overall time complexity of the PC algorithm is $O(n^{k+2})$, where $n = |V|$ and $k = \Delta G$ (the maximal degree of any vertex). This is owing to the number of conditional independence tests required by the algorithm (Jamshidi).

Algorithm 1 The PC Algorithm for Learning DAGs

"All the king's horses, all the king's men couldn't put me back together again"

Input: A set V of nodes and a probability distribution P faithful to an unknown DAG G
Output: DAG pattern H
Initialize H as a complete undirected graph over V ▷ Skeleton Recovery
for $i \leftarrow 0$ to $|V| - 2$ **do**
    while possible **do**
        Select any ordered pair of nodes u and v in H such that $u \in \text{adj}_H(v)$ and $|\text{adj}_H(u) \setminus \{v\}| \geq i$
        for each subset $S \subset \text{adj}_H(u) \setminus \{v\}$ with $|S| = i$ **do**
            if $u \perp\!\!\!\perp_p v \mid S$ **then**
                Set $S_{uv} = S$
                Remove edge $u - v$ from H
            end if
        end for
    end while
end for ▷ v-structure Recovery
for each separator S_{uv} **do**
    if $u - w - v$ exists in H and $w \notin S_{uv}$ **then**
        Orient $u \rightarrow w \leftarrow v$ to form a v-structure
    end if
end for
Return H

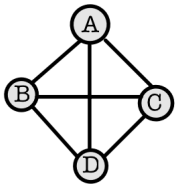
2.3 Simulating the PC algorithm

"Step by step, from town to town"

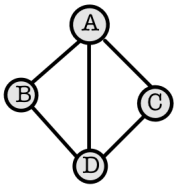
Example 1. Consider a dataset with variables A, B, C, and D. Let the following conditional independence tests be satisfied (calculated from the data):

1. $B \perp\!\!\!\perp_p C \mid A$
2. $A \perp\!\!\!\perp_p D \mid \{B, C\}$

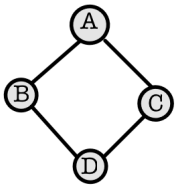
We initialize a graph H, which is the complete graph K_4.



After the first iteration ($i = 1$), the algorithm removes the edge $B - C$ due to $B \perp\!\!\!\perp_p C \mid A$. We set $S_{BC} = \{A\}$.



In the next iteration ($i = 2$), the edge $A - D$ is removed due to $A \perp\!\!\!\perp_p D \mid \{B, C\}$. We set $S_{AD} = \{B, C\}$ This yields the final skeleton.

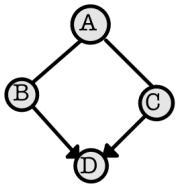


In the v-structure recovery phase, the algorithm orients edges to resolve directed relationships based on separation sets. We iterate over all the separation sets: S_{AD} and S_{BC}.

- S_{AD}: We first try to find a common neighbor for A, D that is not in the separation set S_{AD}. No such neighbor exists as $V/(\{A, D\} \cup S_{AD}) = \emptyset$

- S_{BC}: Observe that D neighbors B, C and $D \in V/(\{B, C\} \cup S_{BC})$ Thus, we orient $B - D - C$ as $B \rightarrow D \leftarrow C$

After resolving all v-structures and ensuring the graph remains acyclic, the resulting partially directed acyclic graph (PDAG) is:



This result also begs the question of how to convert the PDAG returned by PC, also called a *pattern* (Koller and Friedmann), to a DAG. This will be explored in the coming section.

2.4 Dealing with undirected edges

"This thing was a masterpiece 'til you tore it all up"

To convert a PDAG into a DAG, we must ensure that the edges are oriented in such a way that the resulting structure is cyclical. One could think of this as a search over the DAG space wherein we assign each edge a direction, and "undo" our action if it produces a cycle.

Granted, however, that a random DAG arising from a PDAG may not be the mot optimal. A better approach would be to quantify the quality of a given structure, using some score metric that we define. Then, we can undertake a local/greedy search over the DAG space in order to maximize the score metric (which thereby becomes the "objective function" in the case of local search). Commonly used score metrics to quantify the structure of a Bayesian Network, such as BIC and AIC, are briefly discussed in section 3.1.

3 Score-Based Algorithms

The score-based approach first defines a criterion to evaluate how well the Bayesian network fits the data, then searches over the space of DAGs for a structure achieving the maximal score. The score-based approach is essentially a search problem that consists of two parts: the definition of a score metric and the search algorithm.

3.1 Score Metrics

"It's a love story, baby, just say yes"

$$Score(G;D) = LL(G;D) - \phi(|D|)||G||$$

with D = sample size, and $||G||$ = number of parameters in graph G (Ermon Group).

Here, $LL(G;D)$ refers to the log-likelihood of the data under graph G. $\phi(|D|)|G|$ is a regularization term, which penalizes more complex models - favouring simpler ones. The log-likelihood term is computing by implicitly parameter learning using **Maximimum Likelihood Estimation**, which sets the value of the parameters to that which maximize the likelihood of the data.

$$\theta* = MLE(G;D,\theta) = \underset{\theta}{argmax}\ LL(G;D,\theta)$$

$$= \underset{\theta}{argmax} \prod_{i=1}^{n} P(X_i|Parents(X_i))$$

$$= \underset{\theta}{argmax} \prod_{i=1}^{n} \frac{P(X_i, Parents(X_i))}{P(Parents(X_i))}$$

$$= \underset{\theta}{argmax} \prod_{i=1}^{n} \frac{N(X_i, Parents(X_i))}{N(Parents(X_i))}$$

which can be simply obtained from the data since it's merely the **empirical distribution**. Note that $N(X = x, E = e)$ represents the number of data points where $X = x$ and $E = e$, according to the definition of an empirical distribution.

Observe that if we set $\phi(t) = 1$, we obtain the **Akaike Information Criterion (AIC)**, wherein

$$Score(G;D) = LL(G;D) - ||G||$$

Similarly, if set $\phi(t) = log(t)/2$, we get the **Bayesian Information Criterion (BIC)** score function, given by

$$Score(G;D) = LL(G;D) - \frac{log(|D|)}{2}||G||$$

A greater sample size would yield a linearly growing log-likelihood, with a logarithmically growing regularization term, thus making the "penalty term" more insignificant.

Note that typically BIC is defined as $-Score(G;D)$, and thus a lower value of BIC corresponds to a higher implicit "score" - and thus a model with a lower BIC is preferred.

3.2 Hill Climbing

The Hill Climbing algorithm is a quintessential example of a score-based structure learning algorithm, which carries out **local search** on the DAG space. Recall that local search involves beginning at some random point, computing the value of some objective function for neighboring points, and setting the current point to the neighbor which maximizes the objective function. This repeats until all neighors have an objective function value lower than the current point. In the case of structure learning, this general algorithm works exactly the same, where the "initial point" is a random/pre-defined DAG over V, "neighbors" are DAGS obtained by making a finite number of changes ((Zheng et al.) of a fixed type to the currrent DAG, (which can be seen in the algorithm below), and the "objective function" is merely the score metric being used. Again, AIC and BIC are typically used as score metrics.

Algorithm 2 Hill Climb Search for Bayesian Networks

1: **Input:** A dataset D with variables (attributes) $V = \{v_1, \ldots, v_n\}$, scoring function $S(G; D)$.
2: **Output:** A DAG G.
3: Initialize a random or predefined initial structure G_0.
4: Set $G = G_O$
5: **repeat**
6: Compute the score $S(G; D)$ for the current structure G.
7: Generate a set of neighboring structures $N_G = \{G_1, \ldots, G_k\}$ by:
   1. Adding an edge between two variables.
   2. Removing an existing edge.
   3. Reversing the direction of an edge.
8: Set $G* = \underset{G}{argmax}\ S(G; D)$ where $G \in N_G$
9: Set $G = G^*$.
10: **until** No further improvement is possible or a termination criterion is met.
11: Return the best structure G found.

3.3 The Chow-Liu Algorithm

"We found wonderland, you and I got lost in it"

The Chow-Liu Algorithm is a specific type of score-based approach that learns a Bayesian network structure by finding the maximum-likelihood tree-structured graph. In this algorithm, each node has exactly one parent (except for the root node), forming a directed tree. It achieves this by maximizing the mutual information between pairs of variables.

The algorithm proceeds as follows:

1. Compute the **mutual information** (which is the "score" for the Chow-Liu Algorithm) for all pairs of variables X, U, which quantifies the amount of shared information between them:

$$MI(X,U) = \sum_{x,u} \hat{p}(x,u) \log \frac{\hat{p}(x,u)}{\hat{p}(x)\hat{p}(u)}.$$

   This measures how much information U provides about X.

   Mutual information captures how much two variables (e.g., songs in a playlist) depend on each other. Variables with higher mutual information are more likely to have strong connections. For instance, if X and U are independent,

$$\forall x, u \; \hat{p}(x,u) = \hat{p}(x)\hat{p}(u)$$

$$\implies MI(X,U) = \sum_{x,u} \hat{p}(x,u) \log \frac{\hat{p}(x)\hat{p}(u)}{\hat{p}(x)\hat{p}(u)}. = \sum_{x,u} \hat{p}(x,u) \log 1 = 0$$

   Also observe that, $\hat{p}(x,u)$ represents the empirical distribution over x and u,

$$\hat{p}(x,u) = \frac{N(x,u)}{N}$$

   with $N(x,u)$ representing the number of data points with $X = x$ and $U = u$, and N being the total number of data points. Then,

$$Score(G;D) = LL(G;D) = |D| \sum_{(X,U)\in E} MI(X,U)$$

   a proof of which is the appendix.

2. Construct a **complete graph** where each edge is weighted by the mutual information computed in Step 1.

3. Compute the **maximum-weight spanning tree (MST)** from the complete graph using a greedy algorithm such as Kruskal's or Prim's algorithm. The MST ensures that the resulting structure maximizes dependencies while avoiding cycles.

Recall that a **spanning tree** of a graph G refers to a connected, undirected subgraph that contains all the vertices in G. A **MST** is simply the one with the maximum total weight among all spanning trees (where total weight is the sum of all edge weights). The weights in this scenario are given by the mutual information values calculated in step 1.

4. Select any node to be the **root variable**, and assign directions to the edges such that all arrows "radiate outward" from the root. This step transforms the undirected tree into a directed tree structure. The final step of assigning directions ensures a valid Bayesian network, where the tree structure naturally satisfies the Markov condition.

It can be observed that the Chow-Liu algorithm is more computationally efficient that PC, with a complexity of:

- $O(n^2)$ to compute mutual information for all pairs of n variables.
- $O(nlog(n))$ to compute the MST (e.g., using Kruskal's or Prim's algorithm).

Thus, the overall complexity is $O(n^2)$, which is significantly faster than algorithms that search the entire DAG space, as well as constraint-based algorithms like PC which compute a large amount of conditional independence tests.

3.4 Optimality of the Chow-Liu Algorithm

"You're the best thing that's ever been mine"

Theorem 2. The Chow-Liu algorithm is optimal.

Proof. The Chow-Liu algorithm works because it finds the tree structure that maximizes the likelihood of the data. Specifically, the likelihood score decomposes into mutual information and entropy terms, as follows:

$$\log P(\mathcal{D} \mid \theta^{ML}, G) = |\mathcal{D}| \sum_i MI_{\hat{p}}(X_i, X_{\mathrm{pa}(i)}) - |\mathcal{D}| \sum_i H_{\hat{p}}(X_i)$$

where:

- $MI_{\hat{p}}(X_i, X_{\mathrm{pa}(i)})$ is the mutual information between X_i and its parent $X_{\mathrm{pa}(i)}$ in the graph G.
- $H_{\hat{p}}(X_i)$ is the entropy of X_i.

The proof is this is given in **Appendix, Theorem 1**. The goal of any score-based algorithm is to maximize the "score" (log-likelihood) which is equivalent to maximizing the sum of the mutual information terms, as the entropy terms are independent of the dependency relationships. Thus,

$$\arg\max_G \log P(\mathcal{D} \mid \theta^{ML}, G) = \arg\max_G \sum_i MI(X_i, X_{pa(i)}).$$

This shows that the optimal tree structure is the one that maximizes the sum of mutual information values along its edges - making it a valid and optimal score metric. By computing the MST based on mutual information, the Chow-Liu algorithm ensures this optimality.

3.5 Simulating the Chow-Liu Algorithm

"The leaves are all falling down like pieces into place"

Example 2. Consider a dataset with variables A, B, C, and D. The mutual information between each pair of variables is calculated from the dataset.

$$MI(A,B) = 0.32, \qquad MI(B,C) = 0.17, \qquad MI(C,D) = 0.02,$$
$$MI(A,C) = 0.32, \qquad MI(A,D) = 0.07, \qquad MI(B,D) = 0.32.$$

First, we construct a complete graph with edges weighted by mutual information.

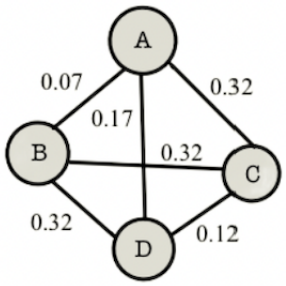


Figure 1: Fully Connected Graph with Mutual Information Edge Weights

Second, we use Prim/Kruskal's algorithms to find the Maximum Spanning Tree. Effectively, we choose the edges with the highest weights: $A - C$, $B - C$, and $B - D$, which add up to a total weight of 0.96.

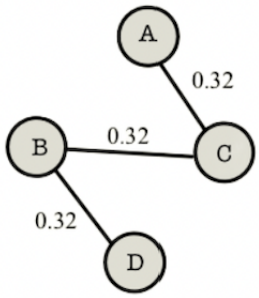


Figure 2: Maximal Spanning Tree arising from Figure 1

Lastly, we orient the tree edges by arbitrarily choosing a root and pointing the edges outwards.

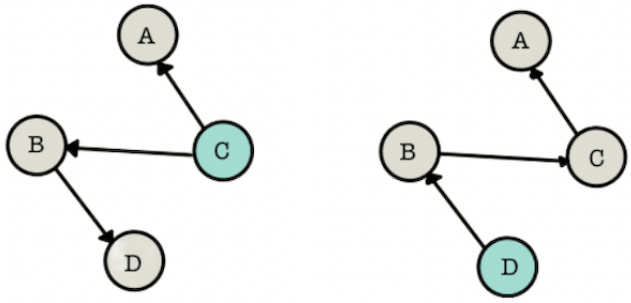


Figure 3: Choosing C as the root yields the graph on the left. Choosing D as the root yields the graph on the right.

As can be seen, the resulting graph is a tree that spans all the vertices and maximizes mutual information. Note that there are also more recent variants of the Chow-Liu algorithm which involve the potential creation of forests, as in the PC algorithm.

4 Applications with a Music Recommender System

This section will involve the application of the Peter-Clark and Chow-Liu algorithms on a dataset provided by Spotify (Spotify), in order to produce the foundations for a music recommender system. Indeed, building a complete system would necessitate parameter learning to completely characterize the Bayesian Network - which is beyond the scope of this paper. The focus of this section, therefore, will be completely on the structure learning aspect of the recommender system.

4.1 Understanding the dataset

"I once was poison ivy, but now I'm your daisy"

The entire dataset consists of 1 million playlists $P_1, .., P_N$, each of which contains a number of music tracks. Each track is identified with a unique resource identifier (uri), and metadata such as the track name, artist's name, etcetera. For the purpose of this analysis, I have significantly reduced the size of the dataset. First, we'll consider results across 1000 playlists, and then across 10000.

Further, 20 songs will be analyzed. 10 of these correspond to the top 10 most frequently appearing tracks across playlists. The other 10 are Taylor Swift songs which are included in the dataset. The expectation is for there to be a strong

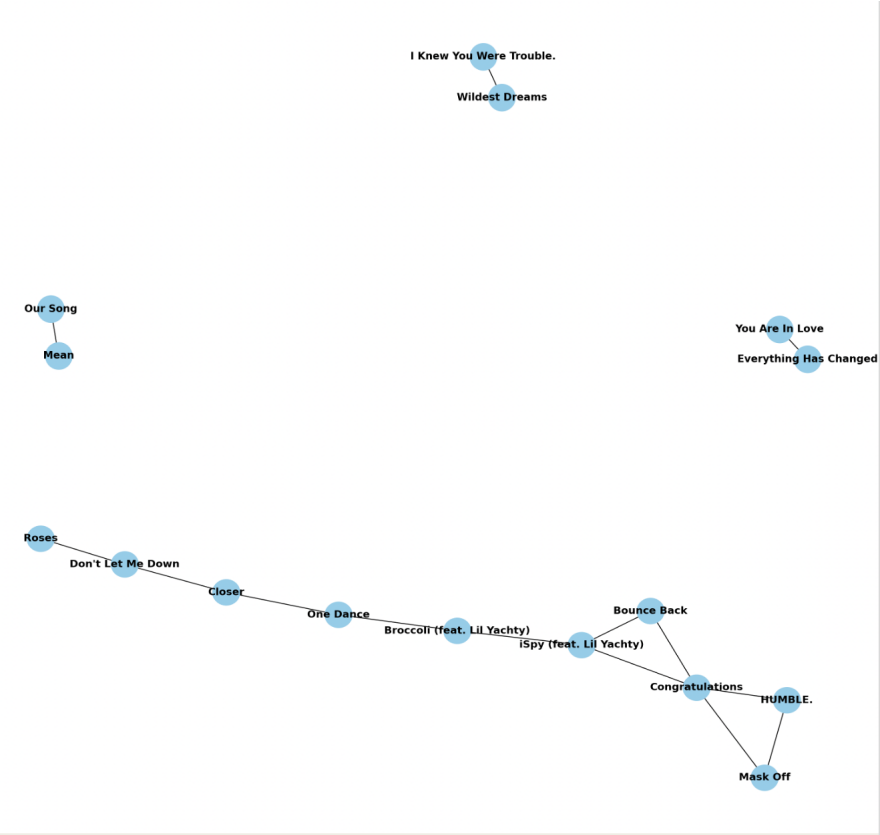
dependency relationship among Taylor Swift songs - after all, once you listen to your first Taylor Swift song, can you ever stop? (if you couldn't tell already, I love Taylor Swift) One would even expect some correlation among the top hits. To convert the creation of the recommender system to the general framework of a structure learning problem, we treat the songs $S_1, ..., S_{20}$ to be our boolean random variables (or attributes). S_i represents whether or not song i is in the given playlist, and thus $S_i \in \{0, 1\}$. Our data consists of the playlists (rows) $P_1, ..., P_N$ for $N = 1000$ and then $N = 10000$.

4.2 Results for $N = 1000$

"I once believed love would be burning red, but it's golden, like daylight"

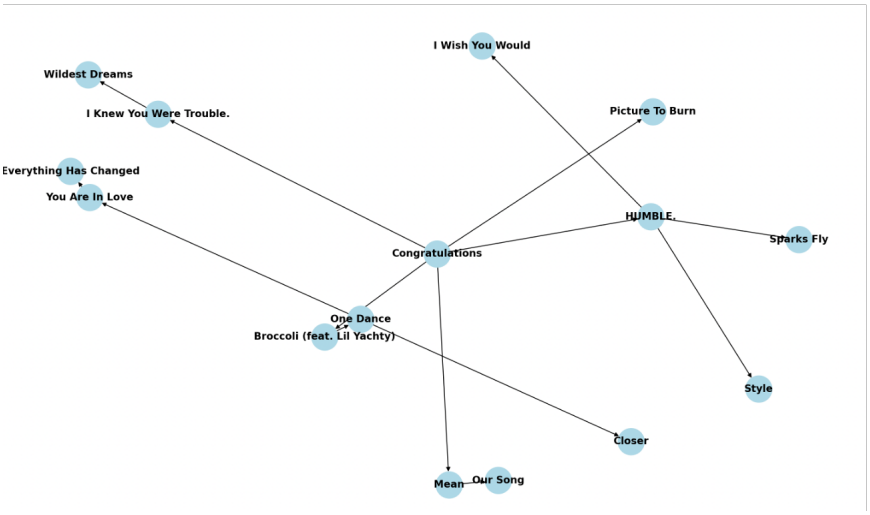
The pgmpy library in Python (pgmpy) proved to be incredibly useful for applying the PC and Chow-Liu algorithms. The code for applying the algorithm is provided in the appendix, and simply applies the required algorithm on a dataframe with columns as attributes (S_i) and rows as data points (P_i).

The resulting graph for the Peter-Clark algorithm is below. Note that most of the edges are undirected (PDAG) and according to this representation, all isolated nodes are not represented.



As expected, the PC algorithm produces a forest with disjoint components of the graph. It is able to recognize a connection between Taylor Swift songs, however it doesn't cluster all her tracks in a strongly connected component. All "hit songs" seem to be strongly dependent on one another, which is an intuitive conclusion.

The graph yielded by the Chow-Liu algorithm follows.



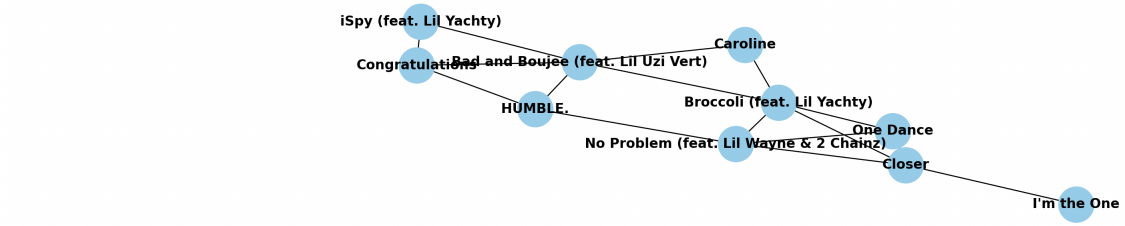
As expected, this particular implementation of the algorithm results in a tree structure (with each "song node" having only parent). Further, ignoring the edge directions, it is a **connected graph** (each node being "reachable" by any other). This is in stark contrast to the forest produced by Peter-Clark. Further, it also yields strong connections between Taylor Swift songs and similar to the Peter-Clark algorithm, it does so in pairs: "Wildest Dreams" and "I know you were Trouble" being a pair, "Everything Has Changed" and "You are In Love" being another, among others. A run time analysis yields the following results (with Chow-Liu being faster), but a bigger sample size is needed to extrapolate more general conclusions.

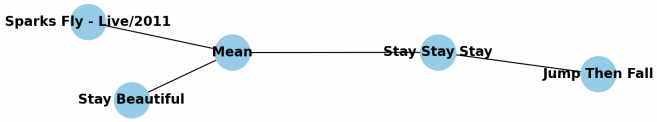
```
PC: Time taken is 0.536 seconds
Chow-Liu: Time taken is 0.341 seconds
```

4.3 Results for N = 10000

"Now I see daylight, I only see daylight"

The resulting graph for the Peter-Clark algorithm is below.





For $N = 10000$,

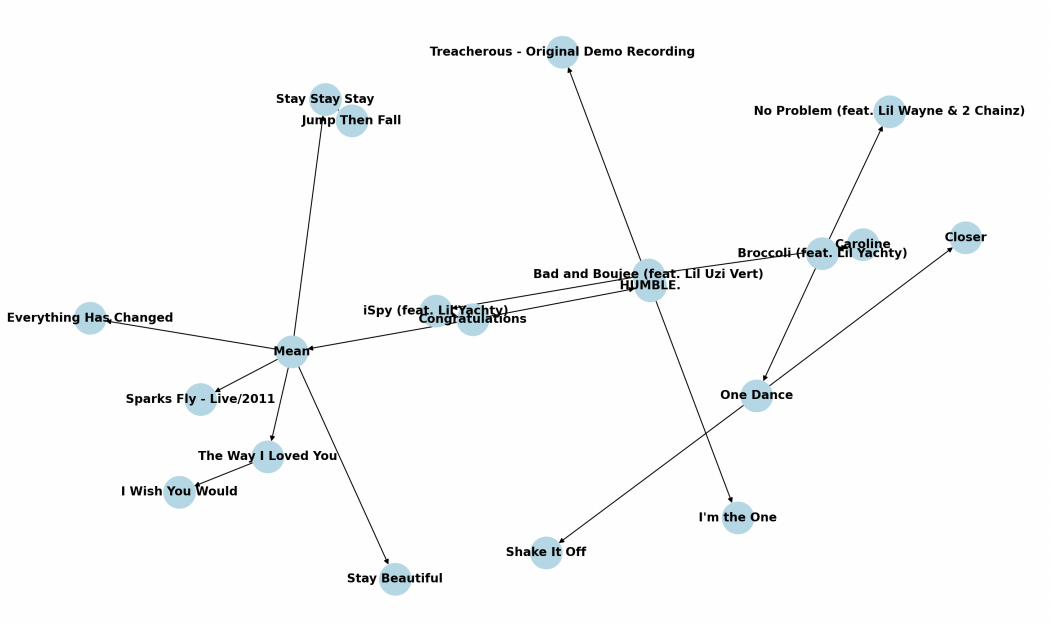
Again, a forest is observed. The results this time are clear - the PC algorithm has recognized the patterns and dependency relationships almost perfectly! All the Taylor Swift songs are in a connected component on the lower left, and all the "hit songs" are in their own connected component on the upper right. The actual results meet the expected ones, however there are two main caveats.

- The aforementioned issue of PC generating a PDAG in place of a DAG, although a possible solution for this was discussed in *section 2.4*
- The runtime greatly exceeds that of Chow-Liu - an expected result of the increasing number of calculations that must be performed to determine conditional independence.

The runtime comparison with the larger sample size is below. PC's runtime is more than 7 times greater than that of Chow-Liu.

```
PC: Time taken is 15.06 seconds
Chow-Liu: Time taken is 2.03 seconds
```

Chow-Liu's accuracy in determining dependency relationships may not be as sureshot, however.



Visually, "accuracy" in dependency detection would entail connections between all Taylor Swift songs (as these maximize mutual information). However, while a majority of her songs are indeed connected, some (like "Shake It Off") are further away from the bulk of the songs. These outliers indicate that it may not have been able to extrapolate these relationships as well as PC. Yet again, its relatively quicker runtime and edge directions provide more information about the structure of the data.

5 Conclusions and Final Results

"This love is good, this love is bad"

The results so far have been the raw output of the algorithms. If we take the output of the PDAG returned by PC, and turn it into a DAG, we can obtain more interesting empirical results. We can then use maximum likelihood estimation to estimate the CPT parameters and compute actual scores.

For $N = 1000$, we obtain the following $K2/BIC$ scores. PC produces a lower BIC score and a lower K2 score, thus producing better results.

```
PC Algorithm K2/BIC: {'K2Score': -1668.2034966414358, 'BIC': -1674.476428045211}
Chow-Liu K2/BIC: {'K2Score': -1652.3505692942008, 'BIC': -1643.9912504058075}
PC produces lower BIC! (Tested 10 times)
```

For $N = 10000$, we obtain the following $K2/BIC$ scores. Chow-Liu is the clear winner here in both regards, producing a lower BIC score and a lower K2 score.

```
PC Algorithm K2/BIC: {'K2Score': -12139.695804694114, 'BIC': -12196.837475380904}
Chow-Liu K2/BIC: {'K2Score': -12799.753725526258, 'BIC': -12793.242933382617}
Chow-Liu produces lower BIC! (Tested 10 times)
```

Hence, whereas Chow-Liu is superior in terms of of time complexity (in addition to PC having the overhead of converting the PDAG to a DAG), it also produces a better BIC/K2 score for a large sample size (on average). However, it would surely be reductionist to claim that one algorithm is inherently better than the other. It's also important to note that PC has the potential to model more complicated relationships - as at the end of the day, Chow-Liu is greatly constricted by its forced "tree" structure.

Future work would yield more insight into this, and also involve a deep dive into hybrid algorithms (which are a mix of both ideas). One could claim that the PC algorithm we considered is also "hybrid" in the sense that after the constraint-based algorithm is finished, we must search the space of DAGs to direct unorientable edges.

Appendix

Theorem For the Chow-Liu Algorithm, maximizing the score metric (the sum of all mutual information terms in the tree structure) is equivalent to minimizing KL divergence.

Proof. In a tree-structured Bayesian network G, each variable X_i depends on a single parent $X_{\mathrm{pa}(i)}$, except for the root, which has no parent. The joint probability distribution over the variables $X_1, X_2, \ldots, X_n$ is:

$$P_t = P(X_1, X_2, \ldots, X_n \mid G) = \prod_{i=1}^{n} P(X_i \mid X_{\mathrm{pa}(i)}).$$

where $P(X_i|X_{pa(i)} = P(X_i)$ for the root. Further, recall that KL divergence of distributions P and P_t is given by

$$KL(P, P_t) = \sum_x P(x) log\left(\frac{P(x)}{P_t(x)}\right)$$

$$= \sum_x P(x) log(P(x)) - \sum_x P(x) log(P_t(x))$$

Observe the first term is simply negative entropy $-H_p(x)$ of $P(x)$. The second term can be further decomposed,

$$\sum_x P(x) log(P_t(x)) = \sum_x P(x) log\left(\prod_{i=1}^{n} P(X_i \mid X_{\mathrm{pa}(i)}\right)$$

$$= \sum_x P(x) \sum_{i=1}^{n} log\left(P(X_i|X_{\mathrm{pa}(i)})\right) = \sum_x P(x) \sum_{i=1}^{n} log\left(\frac{P(X_i, X_{pa(i)}}{P(X_{pa(i)})}\right)$$

using the definition of conditional probability. For $i = root$: $P(X_i|X_{pa(i)} = P(X_i)$ and we further set $P(X_{pa(i)}) = 1$ to obtain the correct corresponding term within the sum. Note that we can rewrite the resulting expression above as,

$$\sum_x P(x) \sum_{i=1}^{n} log\left(\frac{P(X_i, X_{pa(i)})}{P(X_{pa(i)})}\right) = \sum_x P(x) \sum_{i=1}^{n} log\left(\frac{P(X_i, X_{pa(i)})}{P(X_i)P(X_{pa(i)})} P(X_i)\right)$$

$$= \sum_x P(x) \left(\sum_{i=1}^{n} log\left(\frac{P(X_i, X_{pa(i)})}{P(X_i)P(X_{pa(i)})}\right) + \sum_{i=1}^{n} log(P(X_i))\right)$$

$$= \sum_x P(x) \sum_{i=1}^{n} log\left(\frac{P(X_i, X_{pa(i)})}{P(X_i)P(X_{pa(i)})}\right) + \sum_x P(x) \sum_{i=1}^{n} log(P(X_i))$$

$$= \sum_{X_i, X_{pa(i)}} P(X_i, X_{pa(i)}) \sum_{i=1}^{n} log\left(\frac{P(X_i, X_{pa(i)})}{P(X_i)P(X_{pa(i)})}\right) + \sum_{i=1}^{n} \sum_{X_i} P(X_i) log(P(X_i))$$

$$= \sum_{i=1}^{n} MI(X_i, X_{pa(i)}) - \sum_{i=1}^{n} H_p(X_i)$$

Thefore, we can express KL divergence as,

$$KL(P, P_t) = \sum_x P(x) log(P(x)) - \sum_x P(x) log(P_t(x))$$

$$= -H_p(X) - \left(\sum_{i=1}^{n} MI(X_i, X_{pa(i)}) - \sum_{i=1}^{n} H_p(X_i)\right)$$

$$= -\sum_{i=1}^{n} MI(X_i, X_{pa(i)}) + \sum_{i=1}^{n} H_p(X_i) - H_p(X)$$

Observe that the last 2 entropy terms are independent of the dependence tree. Further, mutual information is always non-negative, meaning that the larger this value is, the smaller KL divergence is. Hence, we conclude that minimizing KL divergence is equivalent to maximizing the sum of the mutual information terms (edge weights).

References

Xiaowei, Huang. Probabilistic Graphical Models I: Representation Using Bayesian Networks. University of Liverpool, 2024, https://cgi.csc.liv.ac.uk/ xiaowei/ai_materials/24-PGM-I-Maps.pdf

Jamshidi, Pooyan. Bayesian Network Learning. University of South Carolina, https://pooyanjamshidi.github.io/csce580/lectures/CSCE580-GuestLecture–BNLearning.pdf.

Ermon Group. Structure Learning in Probabilistic Graphical Models. Stanford University, https://ermongroup.github.io/cs228-notes/learning/structure/.

Koller, Daphne, and Nir Friedman. Probabilistic Graphical Models: Principles and Techniques. MIT Press, 2009.

Zheng, Yichi, et al. "A Survey on Knowledge Graphs: Representation, Acquisition, and Applications." arXiv, 2021, https://arxiv.org/pdf/2109.11415.

Spotify. Spotify Million Playlist Dataset Challenge. AIcrowd, https://www.aicrowd.com/challenges/spotify-million-playlist-dataset-challenge/dataset_files.

pgmpy: Probabilistic Graphical Models Using Python. https://pgmpy.org.