Utrecht University
Utrecht, November 2020

Utrecht University

An Improved Algorithm Based on BBES for Learning Bayesian Network Structure from Data

Author: Siyang Qian

Supervisor: dr. M. van Ommen

Second examiner: dr. S. Renooij

MASTER THESIS

MSc COMPUTING SCIENCE

Utrecht University

Acknowledgements

Thank you to my supervisor, Dr. M. van Ommen, for providing guidance and feedback throughout this project. Thank you to my thesis examiner, Dr. S. Renooij, for helping me to write my thesis proposal and review my thesis. Thanks to my study adviser Yvonne Tromp, for always answering my questions with patience. Thanks also to my boyfriend Niels van der Plas, for always being here with me.

Utrecht University

Utrecht, November 2020

Abstract

In this paper we look at some approaches to improve the computational performance of BBES (van Ommen, 2018) algorithm, which aims to learn Bayesian network structure from data. We discuss about how to choose AIC (Akaike information criterion) and BIC (Bayesian information criterion) score in different situations. We give a new branching heuristic based on Graph Attention Network and evaluate it on both simulated continuous data and real-life data.

Keywords – Bayesian network, Graph attention network

Contents

List of Figures

List of Tables

1 Introduction

1.1 Correlation and causation

Imagine you are learning to play a shooting game, and your goal is to kill all your enemies. The first thing one learns is that right-click at some point can make your character move there. You can choose to attack an enemy, but in the meantime, you leave yourself dangerously exposed, which increases the probability that you get shot by his hiding allies. Every choice you made matters because each of them has consequences. These are examples of what we call cause-effect relationships.

Is correlation causation? Two or more things can be positively correlated, negatively correlated, or not correlated at all. If as one set of values increases, the other set tends to increase, then it is called a positive correlation. If as one set of values increases, the other set tends to decrease, then it is called a negative correlation. If the change in values of one set doesn't affect the values of the other, then the variables are said to have "no correlation". Causation between two or more things exists if the occurrence of the first causes the other. The first thing is called the cause and the second is called the effect. A correlation between two variables does not imply causation. On the other hand, if there is a causal relationship between two variables, they must be correlated. So correlation is not causation. But we often use correlation to infer causation.

1.2 Why Bayesian networks

Bayesian networks (BNs) are a type of probabilistic graphical model that represents a set of variables and their conditional dependencies by a directed acyclic graph. BNs are great models for casual relationship study. The reasons are various. First, BNs are graphical models, so that they can show the relationships clearly and intuitively. Second, BNs are directed graphs, thus being capable of representing cause-effect relationships. Third, Bayesian networks are good at dealing with uncertainty problems. They use conditional probability to express the probabilistic relation between various information elements and can learn under limited, incomplete, and uncertain information conditions. Finally, our goal is to find causal relationships from data. Even though there are a lot of types of

models that can be used to represent uncertainty, for example, Markov networks, neural networks, decision trees *etc.*–Bayesian networks are the only models that can learn direct causal relationships. Additionally, they can also represent indirect causation.

In Bayesian network structure learning (BNSL), the goal is to find such causation from observations in the dataset $\mathcal{D}$, and compile those causal relationships into a Bayesian network G. This G is called a causal model and can be used to describe how things work in dataset $\mathcal{D}$. We only care about the structure of the model, that is, the focus is to find the set of edges, so the distribution of values of variables in the model will not be discussed.

1.3 The research question

van Ommen (2018) designed a branch-and-bound equivalence search (BBES) algorithm to learn the structure of Bayesian networks. The algorithm starts with a state that is the set of all equivalence classes on n variables, and then using branching heuristics to branch states and BIC score to bound them (more details in section 2). Our research question is how to improve the computational efficiency of the BBES algorithm by exploring better score metrics and trying different branching heuristics?

1.4 The relevance of the research

Developing a Bayesian network structure is very useful for a variety of applications in general. For example, where there are masses of data available, and we want to understand what underlies the knowledge or which features can cause the others. In addition to providing a network that will allow us to predict behavior under conditions that we have not seen, the structure can also incorporate domain expert knowledge to provide more reliable suggestions.

BNSL has been a successful data analysis tool in many other areas of biology, such as cell signaling pathways, systems biology, genetic data analysis, and prediction-based classification of disease (Needham et al., 2007; Ramoni et al., 2009; Woolf et al., 2005). Schlosberg et al. (2011) applied BNSL to identify potential causal SNP (single-nucleotide polymorphism) associated with the affected phenotype. Ramsey et al. (2017) applied

their BNSL algorithm–fGES to a brain activity research project, trying to find out causal relationships among cortical voxels in a resting-state fMRI scan.

2 Background

There has been a long debate about if causation can be discovered given observational data. Over the last few decades, many people have tried to design algorithms to solve the problem. As a result, more and more evidence has shown that there is a possibility that the cause-effect relationships can be found in observational data. The algorithms below either provided the theoretical support for BBES or offered some improvement directions.

Spirtes et al. (2000) introduced the PC algorithm, which is a classic constraint-based BNSL method. The algorithm starts with a complete undirected graph G, then repeatedly removes edges from G by doing independence tests (Figure 1 shows how the procedure goes). For example, for a pair A ($A \in V$) and A's neighbor B ($B \subseteq Adj(A)$). If A and B are independent given $S \subseteq Adj(A) \setminus B$, then the edge between them can be removed. After that, undirected edges are replaced by directed ones using the previous results to form required v-structures. Finally, the algorithm orients any edges in G that we know the direction but are not a part of a v-structure. However, there are several limitations of the PC algorithm. First, the algorithm could be very inefficient when dealing with high dimensional data, since the runtime of the algorithm is exponential in the number of nodes. Second, if the order of variables changes, the results may change too (Le et al., 2016).

Meek (1997) designed the Greedy Equivalence Search (GES) algorithm, and it was further improved and studied by Chickering (2002a,b). Equivalence refers to Markov equivalence here. Two directed acyclic graphs (DAGs) are Markov equivalent if and only if they have the same d-separations. GES is a score-based algorithm for BNSL. It searches over equivalence classes of DAGs to find out the best class identified by a Bayesian scoring criterion (BIC score in Meek's and BDeu score in Chickering's). The algorithm has two phases. It starts with an equivalence class without any dependencies, and greedily adds a single edge that can be made to all DAGs in the current equivalence class until no more single-edge addition can improve the score (local maximum). The second phase is a backward process to the first one. It performs, at each step, all possible edge deletions that can be added to all DAGs in the current equivalence class until it reaches to a local maximum. As a result, the current equivalence class is returned. However, from

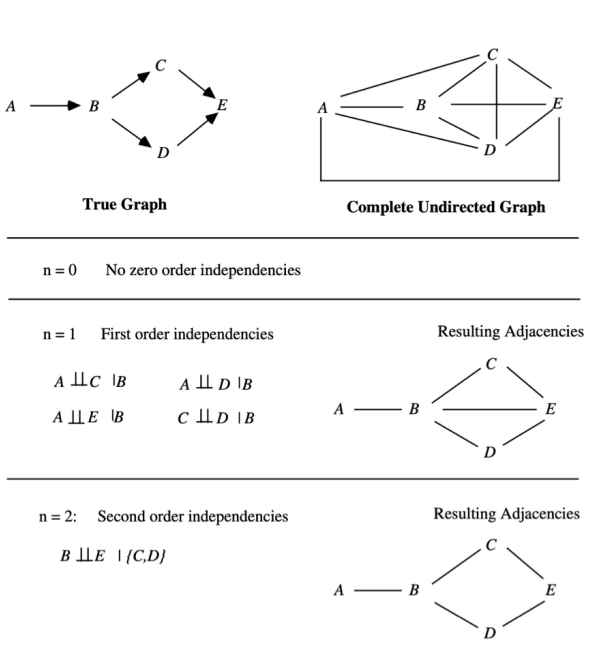

Figure 2.1: PC algorithm: initialization & skeleton search (Spirtes et al., 2000)

the practical view, the algorithm still suffers an expensive computation when applied to complex domains. That is, it is still inefficient when dealing with large variable sets.

Ramsey et al. (2017) introduced an algorithm named Fast Greedy Equivalence Search (fGES), which can be applied to learn a high dimensional Bayesian network structure efficiently. The algorithm uses a similar strategy to traditional GES with some different highlights. First, the updating of the score for a potential edge addition is done by caching scores of potential edge additions from previous steps, and where a new edge addition will not (for graphical reasons) alter the score of a fragment of the graph. In this way, the algorithm no longer needs to compute scores for the entire graph at every step, saving quite an amount of time. The trade-off is more memory will be required for caching scores. Second, each step of the fGES can be done in parallel. Third, they increased the penalty part of BIC score if it is used so that the sparser graph can be produced and the search will take less time. As a consequence, the risk of false negative results gets higher. Fourth, a limited version of the faithfulness assumption is used. That is, for high-dimensional problems, it is assumed that the edge between x and y does not exist, if x and y are uncorrelated. Then the edge $x \to y$ will never be accepted by any chance during the edge addition procedural. The faithfulness assumption is indeed a risky move, it may lead to incorrect graphs in some situations. For those high-dimensional problems, the

trade-off between speed and accuracy is worth trying. For those simple domains, since the speed is not a problem, so there is no need to take such a risk. This paper offered some approaches for improving the efficiency of a score-based BNSL algorithm. We may apply similar strategies in our research as well.

van Ommen (2018) introduced the score-based BBES algorithm to learn the structure of Bayesian networks (Figure 2 is the pseudo code of BBES). It uses conditional independence constraints as a way to represent the search space–equivalence classes. The algorithm starts with a single state containing all candidate solutions, then it chooses a state and split it into two new disjoint states, whose union is equal to the original one. Next, compute an upper bound for each state, and if there is only one element in the state, then the upper bound is its actual score. Finally, choose the item with the highest score, which will be returned by the algorithm. Some details need to be stressed here. Firstly, the algorithm uses an upper bound to score states, and this upper bound is represented by BIC score: an upper bound on the likelihood g and a lower bound on the penalty term h, in terms of $f([G]) = g([G]) - h([G])$. For each state, assume there is a greatest element [G] (for any other elements in the state $g([H]) < g([G])$ and $h([H]) < h([G])$), then $g[G]$ is the upper bound on g. As for the penalty part in BIC score, it is determined by the number of edges (in linear case). To get a lower bound on h, we want to know the smallest penalty among all classes in the state. Secondly, since the algorithm only focuses on the upper bound f([g]), it may take some time until the actual score shows. Thirdly, PDAG or completed PDAG (Chickering, 2002b) is applied to represent Markov equivalent classes.

Algorithm 1: BBES

Input: Data $\mathcal{D}$, number k
Output: A list Res consisting of the k equivalence classes with the highest scores on $\mathcal{D}$
Let S be the state consisting of all classes;
Add the greatest element of S to the list of results Res, and update threshold accordingly;
Create a priority queue Q containing only the state S;
while Q is not empty **do**
  Let S be the state in Q with the largest upper bound, and remove it from Q;
  If the upper bound of S is below the threshold, exit the while loop;
  if S contains two or more classes **then**
    Pick a conditional independence constraint constraint corresponding to a valid Delete operator on the greatest element of S and such that the class resulting from applying the operator is also in S;
    Create a new state T consisting of those classes in S that impose constraint;
    Consider the greatest element in T as a solution, updating Res and threshold;
    Let R be S \ T;
    Add T and R to the queue Q;
  end
end

Figure 2.2: BBES algorithm (van Ommen, 2018)

3 BIC vs AIC in BBES

Score metrics are used to evaluate how good a model fits with the dataset. AIC is an estimate of a constant plus the relative distance between the unknown true likelihood function of the data and the fitted likelihood function of the model (Akaike, 1974), so that a lower AIC means a model is considered to be closer to the truth. BIC is an estimate of a function of the posterior probability of a model being true (Schwarz et al., 1978), under a certain Bayesian setup, so that a lower BIC means that a model is considered to be more likely to be the true model. BBES uses BIC score to evaluate each model, but BIC may not always be the best option. Here are the formulas of BIC and AIC score:

$$BIC = -2\ln(\mathcal{L}) + p\ln(n) \tag{3.1}$$

$$AIC = -2\ln(\mathcal{L}) + 2p \tag{3.2}$$

where $\mathcal{L}$ is the likelihood function, p is the number of parameters in the model and n is the number of data points. Both criteria are based on various assumptions and asymptotic approximations. Despite various subtle theoretical differences, their only difference in practice is the size of the penalty; BIC penalizes model complexity more heavily (Bishop, 2006). Thus, AIC always has a chance of choosing a complex model, regardless of n. BIC has very little chance of choosing a big model if n is sufficient, but it has a larger chance than AIC, for any given n, of choosing a small model (Murphy, 2012). The only way they should disagree is when AIC chooses a smaller model than BIC.

In section 4, we will try to find a better branching heuristic and we used linear Gaussian Bayesian networks to generate the artificial data, so the true model is in the set of candidate models, thus it is better to use BIC in that case because BIC is strongly consistent (Nishii et al., 1988), which means that the true model tends to almost surely be selected. However, if we want to expand our method and use BBES to find the model behind some real-world data, it might be better to use AIC in that case.

4 Convolution on Graphs

BBES is an example of the branch-and-bound technique. It starts with a single state that is just the entire set of candidate equivalence classes. The branch operation takes a state and splits it into two new, disjoint states whose union is equal to the original state. Further, the algorithm computes an upper bound for the scores in any state. If a state consists of just one element, it will use its actual score as the upper bound. These operations are applied until we find an element whose score is larger than the upper bound of each other state. Then this element is the target. While the goal in branching process is to minimize the total time used before finding the target equivalence class. Obviously, branching heuristics have a significant influence on the speed of the algorithm (van Ommen, 2018).

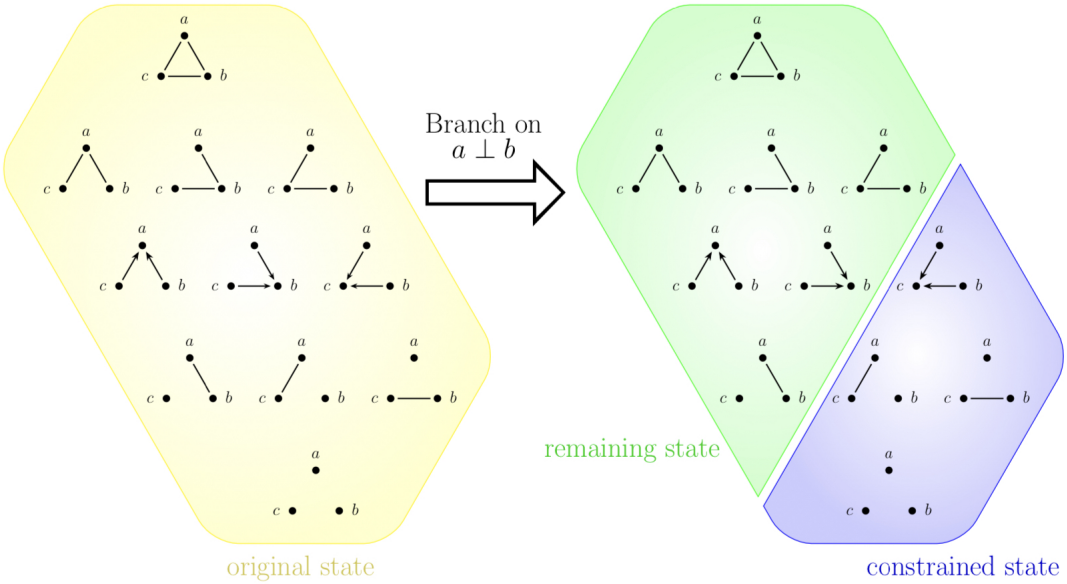

Figure 4.1: How the branch operation works to split a state in BBES (van Ommen, 2018)

Figure 4.1 shows how the branch operation works to split a state in BBES. The state we start with will be called original state, the resulting state where each equivalence class imposes the constraint will be called constrained state and the other state will be called remaining state. The algorithm will carefully choose branch operations to make sure that all states we encounter have a greatest element. So, given the greatest element and the constraints imposed by each equivalence classes, we can get the representation of a state.

In this part, we are going to use a neural network as the branching heuristic. More

specifically, taken a state and a branch operation as an input, the neural network will output a score by which BBES can choose the best branch operation to split the state.

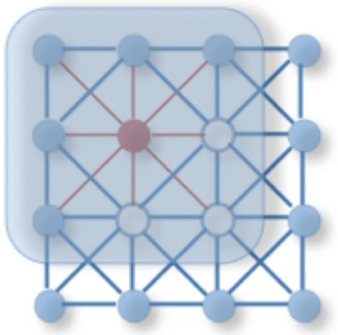

Figure 4.2: 2D convolution. Analogous to a graph, each pixel in an image is taken as a node where neighbors are determined by the filter size. The 2D convolution takes a weighted average of pixel values of the red node along with its neighbors. The neighbors of a node are ordered and have a fixed size (Wu et al., 2020)

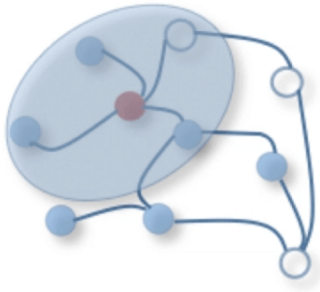

Figure 4.3: Graph Convolution. To get a hidden representation of the red node, one simple solution of a graph convolution operation takes the average value of node features of red node along with its neighbors. Different from image data, the neighbors of a node are un-ordered and variable in size (Wu et al., 2020)

In neural networks, convolutional neural network (CNN) is one of the main categories to solve images recognition and images classification problems (Yamashita et al., 2018). CNN uses a kernel to preserve the relationship between pixels by learning image features using small squares of input data. It is a mathematical operation that takes two inputs such as image matrix and a kernel. The kernel works fine on images since they can always be represented by Euclidean structures–matrices. However, it is hard to do so in graph structures because they are non-Euclidean structures. For example, in a graph G, there may be only one node B which is connected to node A (we call node B a neighbor of node A) while node C has multiple neighbors. Different graphs may have different node relationships, or in other words, different spatial structures. So we need a different neural network model to deal with graphical data.

4.1 Graph convolutional networks

Before we introduce Graph convolutional networks (GCN), we first present a brief introduction to spectral graph theory (Levie et al., 2018). Assume we have an undirected graph $G = (V, E)$, where V is the set of vertices with $|V| = n$, and E is the set of edges. The unnormalized graph Laplacian of G is defined as $L = D - A$, where A is the adjacency matrix and D is the degree matrix of the graph with diagonal entries $D_{ii} = \sum_j A_{ij}$. Then, the normalized Laplacian matrix of the graph is given as:

$$L^{sys} = D^{-1/2} L D^{-1/2} = I - D^{-1/2} A D^{-1/2} \tag{4.1}$$

where I is the identity matrix. The eigendecomposition of the L^{sys} yields $L^{sys} = U \Lambda U^{-1}$, where $U = (u_1, u_2, ..., u_n)$ are orthonormal eigenvectors of L^{sys}, and Λ is a diagonal matrix made of n corresponding eigenvalues, so we know $UU^T = E$. Given the definition of traditional Fourier transform:

$$F(\omega) = \mathcal{F}[f(t)] = \int f(t) e^{-i\omega t} dt \tag{4.2}$$

where ω is the frequency, its graphical version will be:

$$F(\lambda_l) = \hat{f}(\lambda_l) = \sum_{i=1}^{N} f(i) u_l(i) \tag{4.3}$$

where λ_l is the eigenvalue of the lth eigenvector. f is an n-dimensional vector on the graph, each node of the graph has its own $f(i)$, and $u_l(i)$ is the ith value of the lth eigenvector. If we apply matrix multiplication into the formula we can get: $\hat{f} = U^T f$, where U has the same definition as before. Similarly, the inverse Fourier transform of f on graphs in matrix form is $f = U\hat{f}$. Further, we can conduct convolution on graphs in the spectral domain also by analogy with convolution on discrete Euclidean spaces facilitated by Fourier transform. That is, spectral convolution of two signals f and g is defined as:

$$(f * h)_G = U((U^T f) \odot (U^T h)) \tag{4.4}$$

where $\odot$ indicates element-wise product between two vectors. Now we can introduce GCN. Suppose we have a graph $G = (V, E)$, where V is the set of vertices with $|V| = n$, and E is the set of edges. What each GCN layer does is the function below:

$$H^{(j+1)} = \sigma(H^{(j)}, A) \tag{4.5}$$

where $H^0 = X$ is the input of the first layer, $X \in \mathcal{R}^{N \times F}$, N is the number of nodes of the graph, F is the dimension of nodes' eigenvectors, and A is the adjacency matrix. σ is the activation function. Specifically, the $(j+1)$th feature map of a graph convolution layer is calculated as

$$H^{(j+1)} = relu(D^{-1/2}\hat{A}D^{-1/2}H^{(j)}W^{(j)}) \tag{4.6}$$

where $H^{(j)}$ is the jth input feature map, $\hat{A}$ is the normalized adjacent matrix of the graph, D is the degree matrix, and $W^{(j)}$ is the trainable parameters matrix of the jth GCN layer.

4.2 Introducing attention to GCN

From the section above, we know that GCN uses spectral graph theory to process the convolution on graphs. It requires spectral decomposition of Laplacian matrix to process graph convolution operation, which can hurt its generalizability. For example, GCN assigns the same weight to different nodes in a neighborhood. In order to solve this problem, Graph Attention Network (GAT) introduced *attention* mechanism as a substitute for the statically normalized convolution operation. By stacking layers where nodes are able to attend over their neighborhoods' features, it assigns different weights to different nodes in a neighborhood, without requiring any kind of costly matrix operation (such as inversion) or depending on knowing the graph structure upfront (Veličković et al., 2017)

Generally, they let *attention* α_{ij} be computed as a byproduct of an attentional mechanism, $a : \mathcal{R}^F \times \mathcal{R}^F \rightarrow \mathcal{R}$, where F is the number of features in each node. It computes unnormalised coefficients e_{ij} across pairs of nodes i, j, based on their features:

$$e_{ij} = a(\vec{h_i}, \vec{h_j}) \tag{4.7}$$

The graph structure is injected by only allowing node i to attend over nodes in its neighbourhood–only compute e_{ij} for nodes $j \in \mathcal{N}_i$, where $\mathcal{N}_i$ is some neighborhood of node i in the graph. Here is how self-attention is computed at each graph attention layer,

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k \in \mathcal{N}_i} \exp(e_{ik})} \tag{4.8}$$

In addition, multi-head attention (Veličković et al., 2017) is used to stabilise the learning process of GAT. Namely, the operations of the layer are independently replicated K times, and outputs are featurewise aggregated.

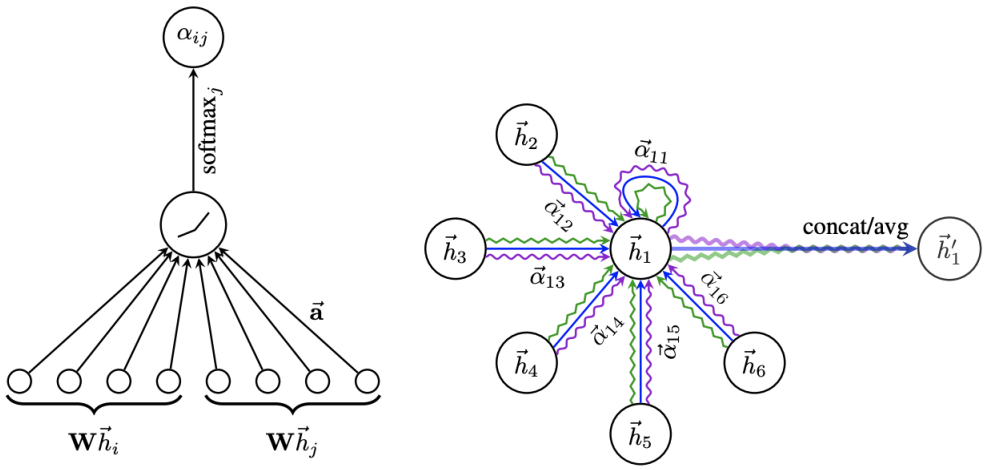

Figure 4.4: Left: The attention mechanism $a(\vec{h}_i, \vec{h}_j)$ employed by GAT model, applying a LeakyReLU activation. **Right**: An illustration of multihead attention (with K = 3 heads) by node 1 on its neighborhood. Different arrow styles and colors denote independent attention computations. The aggregated features from each head are concatenated or averaged to obtain $\vec{h}'_1$ (Veličković et al., 2017)

4.3 GAT approach for states analyses

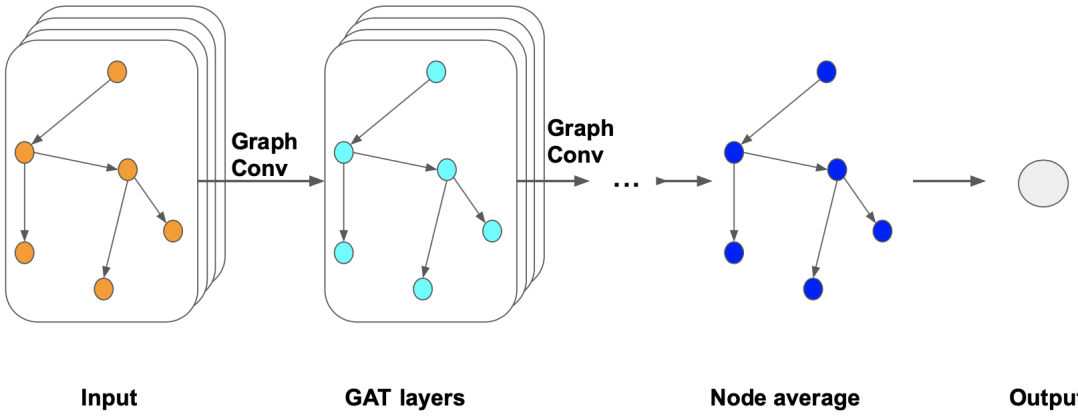

Figure 4.5: The structure of the GAT model

The GAT model applied in our thesis work is illustrated in Figure 4.5. The input X is passed through α graph attention layers. The output of the last GAT will be embedded into one value, which will be used as the output of the model.

In order to apply GAT to our problem, we reconstruct the input graph by adding an extra node between each pair of connected nodes in the original graph. We call them "middle nodes". The reason to do so is that we want to set features on edges instead of nodes. Because features in this problem are constraints, or dependent relationships. The "edge features" can perfectly represent this kind of relationship. For a full discussion of the encoder, we refer to Appendix A1.

5 The branching heuristic with GAT

van Ommen (2018) studied the behaviour of many different branching heuristics in a version of BBES with the pre-computed search space (oracle). They found that among the branching heuristics that look only at the maximum log-likelihood of the constrained state, the best performance is obtained by taking the branch for which this quantity is smallest. Besides the likelihood, the main consideration that should affect the choice of branching operator is the bound on the penalty. They observed that the heuristics that only look at the likelihood may often spend a long time on a single state, repeatedly branching off using a constraint that has a large effect on the likelihood. But as long as the bound on the penalty does not change, the remaining state will again be at the top of the queue for the next iteration. Heuristics that look at the parameter b are trying to choose branches that will lead to a refinement of the remaining state's bound on the penalty term, where b is the fraction of classes with the minimum number of parameters in the original state that are still in the remaining state. Specifically, $b = (bot\_old - bot\_rem)/bot\_old$, where bot_old is the number of classes with the minimum number of parameters in the original state and bot_rem is that in the remaining state.

Since b is proved to be promising in their experiments, we decided to use our GAT model to predict b given a state and a branch operator. Thus we can create a heuristic that does not have access to an oracle (explained in Section 6) while still can use the information of changes in the lower bound on the penalty term. Eventually, based on van Ommen (2018)'s work, we introduce our branching heuristic:

$$min(s, t) + \alpha * p * b \tag{5.1}$$

where s is the upper bound of the new constrained state taking into account its likelihood but not any changes in the number of parameters, t is the threshold score (van Ommen, 2018), α is a constant, $p = 0.5 \log(N)$ where N is the amount of data points which are used to generate equivalence classes, and b is the output of the GAT model.

6 Data

In order to generate the training and test datasets for the GAT model, we ran BBES for a couple of times under pre-computation case, where the algorithm will precompute the entire search space with all equivalence classes and child classes of each equivalence class. This is feasible up to $n = 6$, where the number of equivalence classes is 1,067,825 (Gillispie and Perlman, 2013). The representation effectively provides an oracle for queries such as for the exact minimum penalty among all classes contained in a state, or for the number of classes in the state having that number of parameters. So by querying this search space, we can easily get the fraction of classes with the minimum number of parameters in an original state that are still in the remaining state. We used $maxdep\_maxed\_fracbot\_nojump2$ as the branching heuristic, the data were collected when the algorithm took a state and chose among branches. Specifically, we set $num\_repeats = 5$ to genarate 5 linear Gaussian Bayesian networks, with each of them $N = 10000$ data points were randomly chosen as follows. First, independently for each pair of nodes with $v < w$, an arrow is added from v to w with probability $p = 0.5$. Then the nodes are shuffled using a uniformly chosen permutation. The edge coefficients are sampled independently from the standard Gaussian distribution, and the variances of the noise terms from $\Gamma(1/2, 1/5)$. Finally, 10000 data points are sampled from the distribution defined by this linear Gaussian model (van Ommen, 2018). We got 102224 data points as the training data to feed our GAT model.

7 Results and Discussion

In this section we present results using both simulated and real-world datasets. The experiments demonstrate that our modified method can improve the computational efficiency of BBES algorithm in some cases.[1]

7.1 Implementation Details

The hyper-parameters of the GAT model implemented in this thesis are determined using the training set with 102224 data points and the validation set with 25557 data points. Specifically, the model has 2 graph attention layers (with 20 filters and 1 filter respectively). Both GAT layers have 0.1 features dropout and 0.3 attentions dropout probabilities, which will randomly invalidate 10% features and 30% attentions. The Adam optimizer (Kingma and Ba, 2014) with an initial learning rate of 0.005 is used to train the model for 50 epochs. After that the learning rate is set to 0.0005 until the training loss converges. Early stopping is applied during the training process, that is, training process will stop if the loss on the validation dataset increases during 5 epochs in a row. We use Pytorch in Python to implement the GAT model. When trained with a 2.3 GHz Dual-Core Intel Core i5 CPU, the GAT model takes around 12 hours to train.

7.2 Results on simulated data

We used the same way described in Section 6 to generate the simulated data to run BBES on for this section.

We compared the performance of our branching heuristic to the original one in BBES. Another branching heuristic, which randomly chooses branch operations, is also implemented as a baseline. The results are shown in the tables below. Different repeats and random seeds indicate different Linear Gaussian Models which generate different artificial data points. Where *branch* shows that number of branching operations that were performed, *visit* indicates the number of states that were visited, and *totaltime* is the total time that BBES spent to find out the target equivalence class.

[1]The code for reproducing these results is available online at https://github.com/darrrrrk/bbes

	branch	visit	total time
Original BBES	501	136	22.28
GAT model	**184**	**20**	**5.46**
Random	12706	2005	245.64

Table 7.1: Random seed= 30 is chosen to generate the artificial data. Results are given by 1 time running of the BBES algorithm

	branch	visit	total time
Original BBES	432	92	**32.40**
GAT model.	**421**	**82**	33.60
Random	2613	316	62.59

Table 7.2: Random seed= 100 is chosen to generate the artificial data. Results are given by 1 time running of the BBES algorithm

	branch	visit	total time
Original BBES	3368.8	1433.3	4895.51
GAT model	**2910.4**	**1154.2**	**4270.32**
Random	14304.9	3165.3	6060.81

Table 7.3: Results are given by 10 times running of the BBES algorithm. Each running is under different artificial data generated by different Linear Gaussian Models. *Branch* and *visit* values are averaged by 10.

As the tables show, picking a branch at random gives much worse performance of the algorithm than other choices. This supports the claim that a good heuristic is a deciding factor for the algorithm's performance. Our branching heuristic yielded the best result in the cases shown in Table 7.1 and Table 7.3. We can tell from Table 7.2 that even though both the number of branching operations that were performed and the number of states that were visited are smaller with our methods, the original BBES still gave the shortest total time spent. There could be multiple reasons behind it. For example, it might take some time for the algorithm to load the GAT model and process the inputs.

We only picked some positive results here to support our method. Actually, we also witnessed many situations where our branching heuristic gave worse results than those given by the original branching heuristic. It is not surprising to see so since our training dataset is way smaller compared to the set with all possible situations, there is always some new inputs that our model is not familiar with.

7.3 Results on Real-world Datasets

We used the following real-world datasets from the UC Irvine Machine Learning Repository (Dua and Graff, 2017). "model complexity" in tables below shows the number of edges of the found equivalence class.

1. The ADULT dataset. This contains demographic information about individuals gathered from the Census Bureau database. The original dataset contained a mixture of discrete and continuous attributes, 16 in total. We used only the continuous attributes listed below:

   (a) age

   (b) fnlwgt

   (c) education_num

   (d) capital-gain

   (e) capital-loss

   (f) hours-per-week

Score	Method	branch	visit	total time	model complexity
BIC	Original BBES	5502.0	**1794.0**	**283.4**	11
BIC	GAT model	**5339.0**	1812.0	321.9	11
AIC	Original BBES	**292.0**	**31.0**	**16.6**	12
AIC	GAT model	358.0	62.0	25.0	12

Table 7.4: Results on Adult Dataset

2. The Bike Sharing dataset. Bike sharing systems are new generation of traditional bike rentals where whole process from membership, rental and return back has become automatic. There exists great interest in these systems due to their important role in traffic, environmental and health issues (Fanaee-T and Gama, 2013). The attributes used are:

   (a) temp

   (b) atemp

   (c) hum

(d) windspeed

(e) casual

(f) registered

Score	Method	branch	visit	total time	model complexity
BIC	Original BBES	**1172.0**	**355.0**	**98.3**	11
BIC	GAT model	1218.0	360.0	109.5	11
AIC	Original BBES	**321.0**	**37.0**	**18.8**	12
AIC	GAT model	323.0	41.0	21.1	12

Table 7.5: Results on Bike Sharing Dataset.

As we can see from Table 7.4 and Table 7.5, our model performs worse than original BBES algorithm. The reason could be that our model is not fed with enough training data, thus having a bad generalization. Another reason might be about the way we selected the data, since we just chose some of the features, there might a problem coming with that.

We also tested AIC and BIC on the real-world datasets. We can see from Table 7.4 and Table 7.5 that by using AIC the algorithm can find a solution much faster than using BIC. However the solution will be more complex. For example, in Table 7.4, AIC used original BBES algorithm spent 16.6 seconds to find a 12-edges solution while it took 283.4 seconds for BIC used original BBES algorithm to find a 11-edges solution. The result agrees our mention in Section 3 that BIC has a larger chance than AIC of choosing a small model.

8 Conclusion and Future Work

In this paper, we aimed to improve the computational efficiency of BBES algorithm. We discussed the difference between BIC and AIC score and then designed a graph attention model to find out a better branch heuristic. Test results showed the potential of our GAT model in dealing with graph branching problems. But due to the computer limitation during the lockdown time, we can not train/test our model on a bigger dataset. Thus, an interesting future work could be training our model on a bigger dataset. In addition, our model only works for graphs with 6 nodes. It would be better if it can be applied to graphs with more than 6 nodes as well.

References

Akaike, H. (1974). A new look at the statistical model identification. *IEEE transactions on automatic control*, 19(6):716–723.

Bishop, C. M. (2006). *Pattern recognition and machine learning*. springer.

Bruna, J., Zaremba, W., Szlam, A., and LeCun, Y. (2013). Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203*.

Chickering, D. M. (2002a). Learning equivalence classes of bayesian-network structures. *Journal of machine learning research*, 2(Feb):445–498.

Chickering, D. M. (2002b). Optimal structure identification with greedy search. *Journal of machine learning research*, 3(Nov):507–554.

Defferrard, M., Bresson, X., and Vandergheynst, P. (2016). Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in neural information processing systems*, pages 3844–3852.

Dua, D. and Graff, C. (2017). UCI machine learning repository.

Fanaee-T, H. and Gama, J. (2013). Event labeling combining ensemble detectors and background knowledge. *Progress in Artificial Intelligence*, pages 1–15.

Gillispie, S. B. and Perlman, M. D. (2013). Enumerating markov equivalence classes of acyclic digraph models. *arXiv preprint arXiv:1301.2272*.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Le, T., Hoang, T., Li, J., Liu, L., Liu, H., and Hu, S. (2016). A fast pc algorithm for high dimensional causal discovery with multi-core pcs. *IEEE/ACM transactions on computational biology and bioinformatics*.

Levie, R., Monti, F., Bresson, X., and Bronstein, M. M. (2018). Cayleynets: Graph convolutional neural networks with complex rational spectral filters. *IEEE Transactions on Signal Processing*, 67(1):97–109.

Meek, C. (1997). *Graphical Models: Selecting causal and statistical models*. PhD thesis, PhD thesis, Carnegie Mellon University.

Murphy, K. P. (2012). *Machine learning: a probabilistic perspective*. MIT press.

Needham, C. J., Bradford, J. R., Bulpitt, A. J., and Westhead, D. R. (2007). A primer on learning in bayesian networks for computational biology. *PLoS computational biology*, 3(8).

Nishii, R., Bai, Z. D., Krishnaiah, P. R., et al. (1988). Strong consistency of the information criterion for model selection in multivariate analysis. *Hiroshima mathematical journal*, 18(3):451–462.

Ramoni, R. B., Saccone, N. L., Hatsukami, D. K., Bierut, L. J., and Ramoni, M. F. (2009). A testable prognostic model of nicotine dependence. *Journal of neurogenetics*, 23(3):283–292.

Ramsey, J., Glymour, M., Sanchez-Romero, R., and Glymour, C. (2017). A million variables and more: the fast greedy equivalence search algorithm for learning high-dimensional graphical causal models, with an application to functional magnetic resonance images. *International journal of data science and analytics*, 3(2):121–129.

Schlosberg, C. E., Schwantes-An, T.-H., Duan, W., and Saccone, N. L. (2011). Application of bayesian network structure learning to identify causal variant snps from resequencing data. In *BMC proceedings*, volume 5, page S109. Springer.

Schwarz, G. et al. (1978). Estimating the dimension of a model. *The annals of statistics*, 6(2):461–464.

Spirtes, P., Glymour, C. N., Scheines, R., and Heckerman, D. (2000). *Causation, prediction, and search*. MIT press.

van Ommen, T. (2018). Learning bayesian networks by branching on constraints. In *International Conference on Probabilistic Graphical Models*, pages 511–522.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., and Bengio, Y. (2017). Graph attention networks. *arXiv preprint arXiv:1710.10903*.

Woolf, P. J., Prudhomme, W., Daheron, L., Daley, G. Q., and Lauffenburger, D. A. (2005). Bayesian analysis of signaling networks governing embryonic stem cell fate decisions. *Bioinformatics*, 21(6):741–753.

Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., and Philip, S. Y. (2020). A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*.

Yamashita, R., Nishio, M., Do, R. K. G., and Togashi, K. (2018). Convolutional neural networks: an overview and application in radiology. *Insights into imaging*, 9(4):611–629.

Appendix

A1 Input encoder

Our GAT model takes an adjacency matrix and a features map as inputs. However, in BBES algorithm, the data we can use is equivalence classes and constraints (states representation). So we need some encoders to transfer the data into those forms.

A1.1 Adjacency matrix encoder

The encoder below can transfer a graph into its adjacency matrix applied with "middle nodes" strategy. We assume the equivalence class we have is the graph in Figure A1.1. Our encoder takes the graph as an input, then it will add an extra node between any connected pair of nodes and reorient arrows. For example, it adds a node G between node A and node B, then reorienting the arrow from $A \rightarrow B$ to $A \rightarrow G$ and $G \rightarrow B$. Particularly, if the orientation between nodes is unclear, then still keep it that way during reorientation process. Figure A1.2 shows the reconstruction results. Finally, output the adjacency matrix (Table A1.1) of the reconstructed graph, where (α, β) ($\alpha, \beta \in \{A, B, \ldots, U\}$) will be assigned to 1 if there is an directed edge between α and β, otherwise the value will be 2.

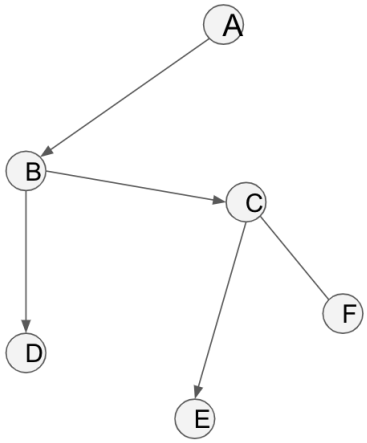

Figure A1.1: Original graph

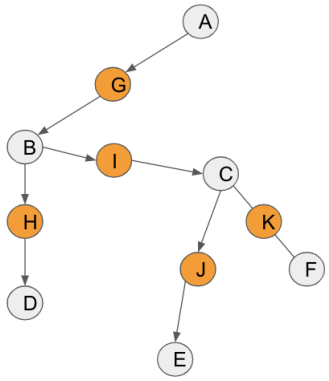

Figure A1.2: Graph with "middle nodes"

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
A	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
B	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0
C	0	0	0	0	0	0	0	0	0	1	2	0	0	0	0	0	0	0	0	0	0
D	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
E	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
F	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
G	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
H	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
I	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
J	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
K	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
L	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
M	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
N	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
O	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
P	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Q	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
S	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
T	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
U	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table A1.1: Matrix of encoded constraints

A1.2 Features encoder

Similarly, the features encoder can transfer (a set of) constraints to a matrix $X \in \mathcal{R}^{N \times F}$, N is the number of nodes of the graph, F is the dimension of nodes' eigenvectors. For example, we have some d-connection constraints listed below:

1. $B \perp\!\!\!\perp C$

2. $B \perp\!\!\!\perp D|C$

3. $C \perp\!\!\!\perp E | A, B, D, F$

For a graph with 6 nodes, there are $\sum_{i=0}^{4} \binom{6}{i} = 57$ situations for the conditional part, so the feature map has 57 columns. For the first case, because $B \perp\!\!\!\perp C$, and I is the middle node between B and C, so (I, 0) will be set to 1. Analogically, (H, 3) and (J, 38) will be also set to 1. The corresponding features map would be:

	0	1	2	3	4	...	56
A	0	0	0	0	0	...	0
B	0	0	0	0	0	...	0
C	0	0	0	0	0	...	0
D	0	0	0	0	0	...	0
E	0	0	0	0	0	...	0
F	0	0	0	0	0	...	0
G	0	0	0	0	0	...	0
H	0	0	0	1	0	...	0
I	1	0	0	0	0	...	0
J	0	0	0	0	0	...	0
K	0	0	0	0	0	...	0
L	0	0	0	0	0	...	0
M	0	0	0	0	0	...	0
N	0	0	0	0	0	...	0
O	0	0	0	0	0	...	0
P	0	0	0	0	0	...	0
Q	0	0	0	0	0	...	0
R	0	0	0	0	0	...	0
S	0	0	0	0	0	...	0
T	0	0	0	0	0	...	0
U	0	0	0	0	0	...	0

Table A1.2: Features map