Learning the Model Structure of Dynamic Bayesian Networks for Automated Speech Recognition

Gherry Harahap

1217461

August 2010

Learning the Model Structure of Dynamic Bayesian Network for Automated Speech Recognition

by

Gherry Harahap

A thesis submitted in partial satisfaction of the requirements for the degree of

Master of Science

presented at

Delft University of Technology
Faculty of Electrical Engineering, Mathematics and Computer Science
Man Machine Interaction Group

August 2010

Man Machine Interaction Group
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology
Mekelweg 4, 2628 CD
Delft, the Netherlands

Members of the supervising committee
Prof. Dr. Catholijn M. Jonker
Dr. ir. Pascal Wiggers
Dr. Hans-Gerhard Gross

August 2010
Gherry Harahap
1217461

Keywords
Artificial Intelligence, automated speech recognition, language model, model learning, data mining, Bayesian Network, search algorithm

ABSTRACT

Learning the Model Structure of Dynamic Bayesian Network for Automated Speech Recognition
by Gherry Harahap (1217461)
Man-Machine Interaction Group
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology

Members of the supervising committee
Prof. Dr. Catholijn M. Jonker
Dr. ir. Pascal Wiggers
Dr. Hans-Gerhard Gross

Improving the performance of Automated Speech Recognition system requires incorporating more knowledge in the model of Automated Speech Recognition system. Information such as the context of the conversation and the characteristics of the speaker can make the task of recognizing speech more accurate. The challenge is how this knowledge can be incorporated in the model of Automated Speech Recognition easily. The answer to this challenge is in using Dynamic Bayesian Network as the model of Automated Speech Recognition. Dynamic Bayesian Network makes extending Automated Speech Recognition model with new knowledge easier by representing the new knowledge as new variable(s) in the model. However, having these variables designing the most optimal model is still not an easy task, especially when there are a large number of variables.

In this thesis, a mechanism is developed to learn the Dynamic Bayesian Network model of Automated Speech Recognition system automatically. In essence, this mechanism can be decomposed into two important components, namely metric and search algorithm. The metric is a quantitative measure of how optimal the model is, while the search algorithm defines the process of learning the most optimal model. This thesis will focus on the model of ASR that has to do with the choice of word in a sentence and put less focus on the acoustic part of the model. For this purpose, a list of possible metrics and search algorithms are presented. For each of this metric and search algorithm, the details of the implementation are also provided. By testing each metric and search algorithm with artificial language and real conversational language, it will be discussed which metric and which search algorithm is suitable for learning the model of Automated Speech Recognition.

Learning is a treasure that will follow its owner everywhere

(Chinese Proverb)

PREFACE

When we watch the Sci-Fi movies, we are usually confronted with these modern technologies with which we can interact with computer or maybe robot in the same way we interact with human. We just speak a word or a sentence and it will be understood by this computer or robot. Unfortunately, the task of recognizing speech has always been one of the difficult tasks for computer. The state of the art of Automated Speech Recognition (ASR) has not yet fulfilled the high expectation set by those Sci-Fi movies. In the past years, the improvement in ASR field has always been minor. But, since Dynamic Bayesian Network (DBN) is introduced in the ASR field, there is a new hope for a more major improvement in the ASR systems since DBN makes addition of contextual knowledge easier.

With the ambition of contributing to the improvement of ASR systems, I have chosen to research the topic of learning the model structure of DBN for ASR system automatically. The work I have done during the research of this topic can be read in this thesis. This thesis is written as a final assignment of Master Thesis Project (IN5000).

My thanks and appreciation goes to Dr.Ir.Pascal Wiggers for letting me research this subject and giving me feedback during the whole research and writing process. Also my deep gratitude to my family and friends either in Indonesia or in the Netherlands from whom I am very lucky to have all of their supports and prayers.

August 5th, 2010

Gherry Harahap

TABLE OF CONTENTS

1 INTRODUCTION

In the last decades, scientists and computer engineers have invented technologies that improve the way we, humans, communicate with computer. Nowadays, almost daily we find ourselves interacting with the computer by means of the keyboard, mouse, touch screen, joystick or other kind of interfaces. These interfaces have now become inseparable parts of computers. However, how handy and useful these interfaces might be, the interaction between human and computers is still not natural and intuitive.

In their book, "*The Media Equation: How People Treat Computers, Television, and New Media Like Real People and Places*", Reeves and Nash has claimed the existence of a *Media Equation* (Reeves & Nass, 1996). What this *Media Equation* asserts is that human-machine interaction is natural and social, so that the rules of human-human interaction also apply to human-computer interaction. This *Media Equation* pleads that the human-human interaction can be used as guide to design the human-computer interaction.

Even in the age when virtual worlds and social networking websites grow rapidly, as a physical being human still relies prominently on speaking and listening when it comes to interacting with another human being. It can then be agreed that speech, whether it is done face-to-face or by means of telecommunication medium, is still the most natural and intuitive interaction in human-human interaction. By arguing that the *Media equation* still holds, it means that speech should also be the most natural and intuitive way to interact with computer. However, speech as a physical phenomenon cannot be understood directly by computer. Computer as a machine composed by binary representatives prefers discrete information such as text as an input and this is where Automated Speech Recognition (ASR) comes in. ASR is a mechanism to translate spoken words into computer readable input such as text or computer command. To emphasize the analogy with human-human interaction, this report will only consider ASR that translates the spoken word to texts.

1.1 APPROACHES IN AUTOMATED SPEECH RECOGNITION

Recognizing speech can be realized in many ways. A trivial approach to do so is by simply comparing the audio signal with another audio signal whose transcription is already known and decide whether these two audio signals are similar to each other. The comparison is done by calculating the difference between those audio signals. If the difference is small enough, then it can be decided that the signals are similar to each other. A well known example for such approach is a speech recognizer for spoken digits (Davis, Biddulph, & Balashek, 1952).

Directly comparing the audio signals may work well for recognizing solely one or two simple words, but in reality speech is more complex. Thus, such approach can be considered not sufficient. To deal with such complexity, more structure is needed in the ASR system.

The structure in speech can be defined from low to high level. As the smallest unit of speech sound, the phoneme is the lowest levels of structure in speech. Each phoneme can be distinguished from other phonemes by looking to its acoustical features. By modeling the acoustical features for each phoneme, a fraction of audio signal can be translated into a certain phoneme.

Each word in speech is composed by a sequence of phonemes. This makes words the next level in speech and going higher a discourse is a collection of sentences. Discussing the structure in speech raises the question how the words are chosen in a sentence. There are several factors that influence how words are chosen in speech. A few examples of these factors are:

1. *The language models*, this model determines the interplay of words. It reflects how the choice of a word influences the choice of subsequent word in a sentence.

2. *Type of the speaker*, this information reveals how the background of the speaker influences his or her choice of words.
3. *Context*, this information reveals how the context of the conversation is also decisive in the choice of words in a sentence.

Even though these factors are influential, standard ASR systems nowadays only incorporate the language model in their system.

Having these structures, an ASR model for the ASR system can be made. By means of this model, the audio speech can be structurally transcribed to collection of sentences. The reasoning behind the transcription is based on statistical reasoning which is the fundament of all ASR system. Statistical reasoning is basically choosing the transcription by weighing in how likely or probable all possible transcriptions are.

The performance of an ASR system is usually measured in the terms of accuracy and speed. The speed of the ASR system depends on how the recognizer is modeled and how efficient the implementation is. The number of words the recognizer has in its vocabulary also determines how large the model will be. A larger model implies slower recognition speed. On the other side, the accuracy of ASR system is measured quantitatively. One of the most popular accuracy evaluator is Word Error Rate (WER) which simply measures how many word errors are there in the recognized sentence.

Table 1, which is taken from (Wiggers, 2008), shows the accuracy level of the ASR systems nowadays. Even though the accuracy is not quite disastrous, the accuracy is still significantly lower than the accuracy of human speech recognition. This proves that automated speech recognition is still a difficult task for computer.

Table 1 Accuracy level of the state-of-the-art Automated Speech Recognition systems

Type of Speech	Lexicon Size	Word Error Rate	Human Error Rate
Digit Recognition	10	0.5%	0.009%
Read newspaper speech	20000	3%	
Read newspaper speech	64000	5%	1%
Broadcast news	64000	10% - 20%	
Conversational speech	64000	20% - 40%	4%

In (Forsberg, 2003), the difficulties with ASR are summed up. These are:

1. Human comprehension of speech is integrated in recognizing speech. This means that ASR system should also be aware of human's knowledge whether it is linguistic knowledge or world knowledge.
2. Human speaker does not only communicate with speech but also body signals.
3. In most of the cases, speech signal is always contaminated with other noises from the surrounding environments.
4. Spoken language does not always follow the rules of written language. It is dialogue oriented, instead of one-way communication.
5. Speech does not have boundaries between words.
6. Speakers have their special voices and speaking style. The factors that influence this variability of speaker are:
    a. Realization, the resulting speech signals of identical word never look the same
    b. Speaking style, human tends to adjust their choice of words or prosody according to their environment

c. Gender of the speaker
d. Anatomy of vocal tract
e. Speed of speech
f. Regional and social dialects

7. The amount of search space is enormous.
8. Natural language is inherently ambiguous. There are two kinds of ambiguity that each ASR might have difficulties with:
    a. *Homophones*, this occurs when the words sounds the same but have different orthography. For example, "the tail of a dog" and "the tale of a dog" sound quite similar.
    b. *Word boundary ambiguity*, this occurs because there are multiple ways of grouping phones into words. For example, "It's not easy to wreck a nice beach", "It's not easy to recognize speech" and "It's not easy to wreck an ice beach" are pronounced similarly.

These difficulties boils down to one essence point: there is a lot of knowledge that is still absent in the ASR model. So, the only way to improve ASR is to add that knowledge in the ASR model. This is why extending the model in the speech recognizer is important in tackling those difficulties. In the next section, a concept to accomplish this is introduced.

1.2 IMPROVING AUTOMATED SPEECH RECOGNITION

As explained in the end of the last section, in order to tackle a large portion of difficulties in realizing ASR, the model of the ASR should be easily extendable.

The standard ASR nowadays uses Hidden Markov Model (HMM) as model (Juang & Rabiner, 2005). HMM is a model for stochastic processes. Since speech processes can be modeled as stochastic processes, HMM is a suitable model for ASR systems. Extended from Markov chain, HMM describes speech as a chain of unobserved states (Rabiner, 1989). The states itself represent the structure of speech in HMM model. In this chain, a path for each possible combination of states is defined. It is the task of the designer of HMM for ASR system to define this chain so that all possible paths exist. We can imagine that when the number of states increases, it becomes harder for the designer to come up with a valid HMM. This is why HMM is not easily extendable.

However, there is another way to model speech processes and also makes the model easily extendable. This can be realized by applying Dynamic Bayesian Network (DBN) as a model of ASR system. DBN is a temporal probabilistic graph that defines stochastic processes by discretizing the system into variables and dependencies or independencies between these variables. For more details about DBN, we refer to the next chapter.

What makes DBN convenient in extending the model is that in DBN it only requires adding more variables and defining the dependencies with other variables by adding edges. Compared with Hidden Markov Model (HMM) which most of the conventional ASR models are based upon, this is far more convenient. The difference between HMM and DBN is in how these models define the processes. HMM defines the processes as the sequences of every variable's states whilst DBN defines it as dependencies of variable. The number of possible sequences of every variable's states grows exponentially. That number is especially much larger compared to the number of possible variable's dependencies. Therefore, extending the model in HMM becomes a complex task, whereas extending the model in DBN is simple and orderly.

In (Zweig & Russell, 1998), the result of an ASR experiment using DBN is discussed. As a test data, the authors have chosen a Phonebook database (John, Fong, Wong, Spitz, & Leung, 1995). In the test, there are 5 models compared, viz., Baseline-HMM, Correlation-DBN, PD-Correlation-DBN, Chain-DBN and Articulator-DBN. The

result of this test can be seen in Table 2. As it can be seen, the authors have made 4 variants of DBN models and 1 HMM model. In this test, the authors measure the accuracy of each model by means of Word Error Rate.

Table 2 Result of accuracy test for 5 ASR structure models using PhoneBook database

Network	Parameters	Word Error Rate
Baseline-HMM	127k	4.8%
Correlation-DBN	254k	3.7%
PD-Correlation-DBN	254k	4.2%
Chain-DBN	254k	3.6%
Articulator-DBN	255k	3.4%

From the table, even though the difference is not statistically significant, it can be seen that the accuracy of DBN models especially Articulator-DBN is slightly better than the Baseline-HMM. The result shows that DBN-based speech recognizer will at least be able to reach the accuracy of Baseline-HMM and hopefully with a good choice of DBN model it can outperform the Baseline-HMM.

But how do we get this DBN model? The conventional approach to obtain DBN models is by eliciting them from experts. Through a series of interviews and inquiries, the knowledge of the experts can be translated into a DBN model. However, there are the obstacles when using this approach:

- The experts are very good in qualitative knowledge, but are very poor in quantitative knowledge like giving an explicit dependencies between variables or giving the exact parameters
- The experts are not always willing to give the knowledge since they see it as a threat for their job
- Interviews and inquiries take a lot of time and are prone to error.

Fortunately, the experts are not the only source of knowledge. In the case of ASR, there are collections of speech recordings with its transcriptions and annotation that give examples of how the DBN model should behave. This pleads for a need for such mechanism to learn the DBN model automatically from these observation data. This approach also brings other advantages:

- Extending the model becomes easier since what the learning mechanism needs is just extra data for the extended part of the model.
- The learning mechanism can be designed such as that it also improves the performance of the ASR.

In the next section, it will be explained how the research in this thesis will contribute to finding such mechanism and thus making the aforementioned advantages more achievable.

1.3 RESEARCH GOALS

Considering how beneficial it is for the ASR system that is based on DBN, the research in this thesis will attempt to develop mechanism in learning the DBN model specifically for ASR. Such a mechanism takes the observation data as its main source and results in DBN model that is the most optimal according to the observation.

As it will be explained further in Chapter 3, such model learning mechanism can be decomposed into two components. These two components are metric and search algorithm. Metric is a quantitative measure of how good the model is, while search algorithm determines how the learning process is done.

The research looks into these two important components of such mechanism. There are many choices of metrics and also many choices of search algorithms. Thus, it is interesting to scrutinize which of those metrics and those search algorithms are the most suitable for learning the DBN model for specific ASR system.

Furthermore, learning the model for an ASR system can be made easier when there is more background knowledge about how the ASR system should behave. This implies that this background knowledge can be used as constraints when learning the DBN model to direct the learning process into a more reasonable direction. In this research, the effect of applying these constraints into the existing DBN model learning techniques will be examined.

1.4 FOCUS

In Section 1.1, it is described that in recognizing a sentence, structures in building a sentence must be incorporated. Such structure goes from high level structure such as word level to low level structure such as phonemes. The focus in this research is on the high level structure. This means that the research engages in learning the DBN model that leaves out the low level structure such as acoustical feature of the phoneme.

This research also focus more on the structure of DBN model and less on parameters. The challenge of learning DBN model lies in eliciting the dependency and independency between the variables. When the structure is known, the parameters can be easily obtained since in this case simple count statistics are sufficient.

1.5 STRUCTURE OF THE THESIS

After the introduction in this chapter, the thesis continues with the literature research about the mechanism of learning the structures and parameters of the DBN. In this chapter, each technique that can be found will be discussed and summarized. Having seen the possibilities in learning the structures and parameters of the DBN, we discuss how we can translate these techniques to learn the model for ASR systems in the subsequent chapter. In chapter 4, all the implementation details of model learning for ASR will be provided. At this point, it is clear how the model learning is done. Therefore, the next chapter we test the model learning techniques with artificial data and real conversational data. For each experiment, it is analyzed how these techniques perform. In the last chapter, conclusions with regards to effectiveness and efficiency of these techniques are provided.

2 BACKGROUND RESEARCH

In 1952, the first Automated Speech Recognition (ASR) was introduced. This first ASR was designed to solely recognize spoken digits (Davis, Biddulph, & Balashek, 1952). Since then, this field has advanced and various kinds of ASR are developed. A summary of how the development of ASR went can be read in (Juang & Rabiner, 2005). Even though there are various kinds of ASR, the modern ASR systems are similar in how they work.

The standard mechanism of modern ASR systems is depicted in the diagram in Figure 1. As an input, the ASR system takes an audio signal and split it into small time frames. The length of such a time frame is determined by how long the designer of the ASR system assumes the audio signal is stationary. The most common choice for this is between 10 to 20 ms. For each of these time frames, the acoustical features are extracted from it and then these acoustical features are sent to the Recognizer to be processed. The Recognizer outputs the text that is most probable.

Since the Recognizer outputs a text, it will help the accuracy of the Recognizer if it has more knowledge about the text itself. The knowledge that can be useful is for example how the sentence is structured, which words are more probable to occur, etc. This knowledge is modeled in the so-called language model. That's why the information that supports this language model also goes in the "Recognizer".

This means that the "Recognizer" is composed of two models:

- The acoustic model
- The language model

These two models exhibit two independent processes in speech. The acoustic model describes how the sound is produced whilst the language model specifies the choice of words and the structure of sentence.

Before proceeding to a deeper discussion about the Recognizer, we will first explain the language models further because the language models will play an important role in this thesis. A language model defines the statistical distribution of possible word sequences in a sentence. Translated into mathematical terms, a language model assigns probability to $P(w_1, \dots, w_n)$ where $w_1, \dots, w_n$ is the sequence of words in a sentence. The most common and broadly applied language model is N-gram model. N-gram model predicts the next word in the sentence by means of the previous $N-1$ words or in mathematical equation the statistical distribution is defined as follows:

$$P_{N-gram}(w_1, \dots, w_n) = \sum_{i=1}^{n} P(w_i | w_{i-1}, \dots, w_{i-(N-1)})$$

ASR system usually predicts the next word by means of 1 or 2 previous words. When 1 previous word predicts the next word (2-gram model), it is commonly called bigram. When it uses 2 previous word as predictor of the next word (3-gram model), this model is called trigram. Determining $P(w_i | w_{i-1}, \dots, w_{i-(N-1)})$ for each w_i is done by a simple count statistics.

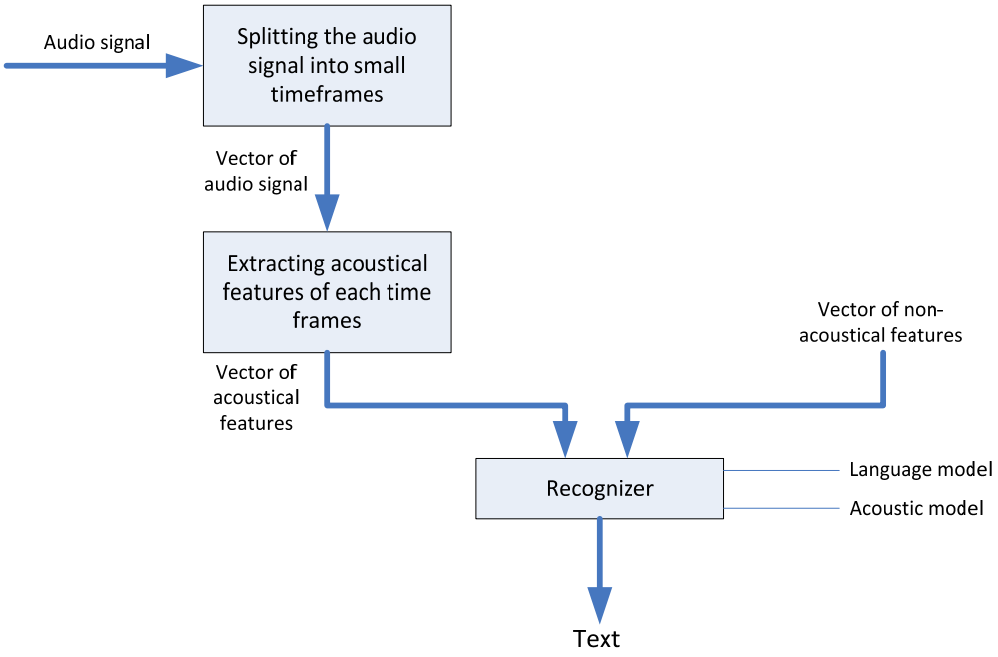


Figure 1 The standard mechanism of modern Automated Speech Recognition

In the diagram, we also treated the "Recognizer" as a black-box. This is because there are many approaches to interpret sets of features and turn it into a text. Fortunately, these approaches boil down to the same principle of maximizing the probability of word(s). Given that W is a word or a sequence of words and O is the sets of features, then $\widehat{W}$, the output of the recognizer, will be:

$$\widehat{W} = \underset{W}{\operatorname{argmax}}\, P(W|O)$$

The most popular approach in ASR systems nowadays is Hidden Markov Model (HMM)-based speech recognizer. This approach involves modeling all possible sequences of states from smallest level until the highest level using HMM. The smallest level in speech is the phoneme which is the smallest unit of speech sound. The highest level can be chosen according to how sophisticated the designer wants the ASR system to be. The HMM can be modeled for the purpose of handling in sentence level or even conversational level. However, the higher the level the ASR system handles, the bigger the HMM model will be and hence the more intensive the computation will be. HMM-based speech recognizer maximizes the probability $P(W|O)$ by means of the dynamic programming which allows the recognizer not to exhaustively count all of the possible word or word sequences.

Since all possible sequences must be incorporated into the HMM, we can imagine how extensive and complex the model will be when the HMM that contains more high level features has to be designed. It is in such complexity that a mistake in placing an edge may produce output that is not desirable. For designers of ASR system, this makes their job much more difficult because coming up with a model of HMM will need lots of creativity and caution in order to let the desirable paths to be accessible and the undesirable ones to be inaccessible. On the other side, extending the model for high level features is necessary to improve the accuracy of ASR system.

As mentioned in previous chapter, Dynamic Bayesian Network (DBN) is introduced as an alternative of HMM. The first section in this chapter will begin with introducing DBN and showing how it can be applied to ASR system. Subsequently, it will be discussed how the parameters in DBN model can be learnt. In the third section,

a collection of techniques in learning the structure in DBN model will be presented. The chapter will end with a summary of background research.

2.1 DYNAMIC BAYESIAN NETWORK FOR AUTOMATED SPEECH RECOGNITION

This chapter is meant as a preparation prior to reading the learning techniques discussed in the subsequent chapters. In the first section of this chapter, the basics and most important concepts of Bayesian Network is introduced and the extension of Bayesian Network to the Dynamic Bayesian Network follows in the next section. Finally, the application of Dynamic Bayesian Network in Speech Recognition is treated in the last section.

2.1.1 BAYESIAN NETWORK

To explain a real-world system, the first common step to do is discretizing the events in the real world system into a set of variables. Each of these variables can have a particular state that is defined either in discrete space or continuous space. Having these variables, the system can be explained by pointing out the relationship between these variables. One of the ways to model these relationships is by means of Bayesian Network (BN).

2.1.1.1 BASICS

A BN is a probabilistic model that represents a set of variables and independencies between these variables (Korb, 2004). As a graphical representation of BN, a directed graph is adopted. See Figure 2 for an example of BN with three variables. As can be seen from the figure, a BN is composed by two graphical components:

- A **Node** represents the modeled variable. In Figure 2, node A, B and C represent variable A, B and C, respectively.
- An **Edge** represents direct dependency between two variables that the edge connects. The direction of the edge specifies the causal relationship between the connected variables. In Figure 2, it can be deduced that variable A is the cause of B or B is the effect of A. We also refer A as a parent of B or equivalently B is a child of A.

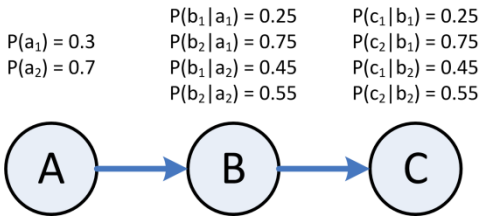


Figure 2 A simple example of Bayesian Network with three variables

Every BN must follow a strict constraint of acyclicity. A BN that is not a directed acyclic graph allows the logical fallacy *petitio principii* (assuming the initial point) into the model. Practically, this means that when calculating the probability of a certain variable, its calculation will include infinite amount of terms in the equation. Ergo, cyclicity in BN is neither logical nor computationally possible. So, notice in Figure 2 that adding an edge between node A and C with the line pointing to node A will make the BN incorrect, since the graph will become cyclic.

A BN is a probabilistic model because each node in the BN has a probabilistic table defining probabilities for all states of the corresponding variable. If the node that represents the variable has a parent in BN, then this node

will have a Conditional Probability Table (CPT). This CPT defines the posterior probability of each state of the variable given the state of the parent's variables. On the other side, a root node (node that doesn't have any parent) will have a probabilistic table that defines the prior probability for each state in the variables. In Figure 2, we see that variable A is the root node and it defines only prior probability, while variable B and C have CPT.

By representing the independences between the variables, a BN simplifies the joint probabilities between all variables. Given that $x_1, x_2, \dots x_n$ are the variables in the system. Without any knowledge of independences, the joint probability is defined as:

$$P(x_1, x_2, \dots x_n) = P(x_1)P(x_2|x_1) \dots P(x_n|x_1 \dots x_{n-1}) = \prod_i P(x_i|x_1 \dots x_{i-1})$$

Profiting from independences between the variables, the joint probability is simplified to:

$$P(x_1, x_2, \dots x_n) = \prod_i P\big(x_i|parents(x_i)\big)$$

This simplification certainly helps making the reasoning under uncertainty with a BN more comfortable, since in the calculation of conditional probability only relevant variables are needed. This means that there are less parameters in the CPTs (less cells in the table). Ergo, the faster calculations can be achieved.

Given an observation in the real-world system of some variables (we call these variables evidence variables), a BN reasons by updating the belief of other variables. The process of belief updating is done via a *flow of information* through the network. To make the idea of *flow of information* clearer, let's first denote X_Q as variables that are queried (variables whose belief are searched) and X_E as evidence variables, hence updating the belief means calculating $P\big(X_Q\big|X_E = x_E\big)$. By using Bayes rule, the term can be written as follows:

$$P\big(X_Q\big|X_E\big) = \frac{P\big(X_Q, X_E\big)}{P(X_E)} = \frac{\sum_{H \notin \{Q \cup E\}} P\big(X_H, X_Q, X_E\big)}{\sum_{H \notin \{Q \cup E\}} P(X_H, X_E)}$$

As can be seen from the equation, X_H are other variables that is neither query nor evidence variables, but these variables have influence in either query or evidence variables. Such variables are called hidden variable. However, the *flow of information* does not always update all connected variables in the network because of *conditional independence* between variables. This will be explained in the next section. For more details about the calculation and algorithm of belief updating procedure, see the third chapter of (Korb, 2004).

2.1.1.2 CONDITIONAL INDEPENDENCE

Given the structure of the graph, there are three kinds of conditional independence, viz. Causal Chains, Common Cause and Common Effects.

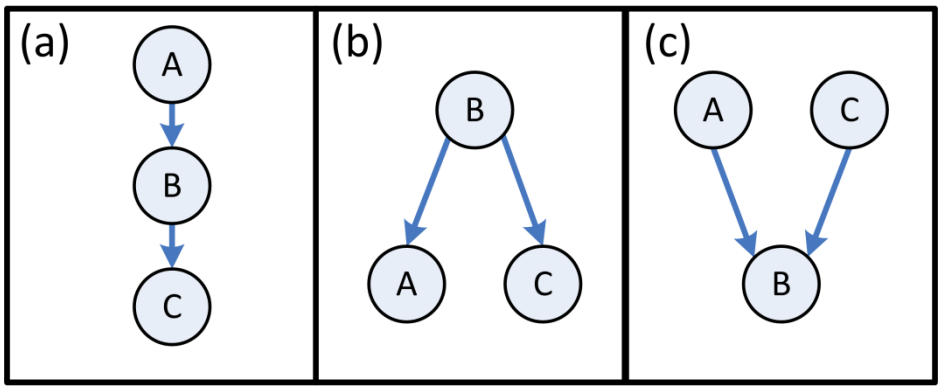


Figure 3 Example of (a) Causal Chains, (b) Common Causes and (c) Common Effects

CAUSAL CHAINS

The structure of causal chain is depicted in Figure 3(a). This structure gives rise to the conditional independence: $P(C|A,B) = P(C|B)$ or it is also denoted by $A \perp C|B$. The independence in causal chain can be interpreted as follows: Given that the evidence of B is present, more knowledge about A will not add any information about the belief of C.

For example, the rain makes the ground wet and it is also known that the grass will grow if the ground is wet. In this example the conditional independence suggests that given it has been proven that the ground is wet, our belief about whether grass will grow will not change when we observe that it has rained.

COMMON CAUSES

Figure 3(b) shows the structure of Common Cause that also gives rise to the conditional independence $P(C|A,B) = P(C|B) = A \perp C|B$. As an example consider the following case: It is known that rain will cause the increase of ground's humidity and also the decrease of ground's temperature. Given the evidence of the rain, then the fact about ground's humidity will not change our belief about the temperature of the ground.

COMMON EFFECTS

Figure 3(c) shows the v-structure which is the structure of Common Effects. In contrast to Causal Chains and Common Chains, this v-structure gives rise to the conditional dependence: $P(A|B,C) \neq P(A|C)$ or it is also denoted by $\neg(A \perp C|B)$. Consider the following example: A landslide has occurred yesterday and there are two causes to this landslide, viz. heavy rain and earthquake. Given that it is known that there is no earthquake reported, then it will increase our belief that it has rained heavily yesterday. It also applies when it is known that there is indeed an earthquake reported, then it will decrease our belief about the heavy rain. The reasoning that is exercised in this case is also known as *explaining away*.

2.1.1.3 MARKOV EQUIVALENCE

In (Verma & Pearl, 1990), the author defines that two causal models D_1 and D_2 are *equivalent* if for every theory $T_1 = \langle D_1, \Theta_1 \rangle$ there is a theory $T_2 = \langle D_2, \Theta_2 \rangle$ such that T_1 and T_2 define the same probability distribution and vice versa. Since BN can also be considered as causal model, this definition also applies to BN. In this report, we refer this equivalence for two BNs as *Markov Equivalence*.

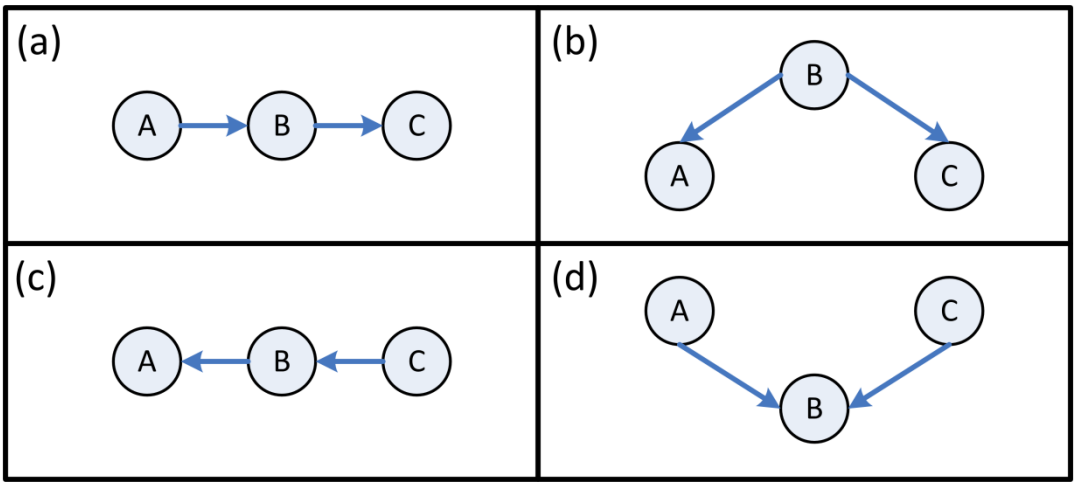


Figure 4 Four causal models from (Verma & Pearl, 1990)

To make the notion of Markov Equivalence more comprehensible, let's consider the following four causal models of Figure 4 (Verma & Pearl, 1990). For BN depicted in (a), the parameters that are needed are $P(A)$, $P(B|A)$, and $P(C|B)$, while for BN depicted in (b) requires the parameters $P(B)$, $P(A|B)$, and $P(C|B)$. The BN (a) and (b) are Markov equivalent because the parameters of BN in (b) can be derived from the parameters of (a). The parameters $P(B)$ and $P(A|B)$ from (b) can be derived as follows:

$$P(A)P(B|A) = P(A,B) = P(A|B)P(B)$$

This implies that (a) and (b) have the same probability distribution and vice versa, hence, (a) and (b) are Markov equivalent. This also applies for (c), because (c) can also be derived from both (a) and (b). On the contrary, parameters of (d) cannot be derived from (a), (b) or (c), hence, (d) is not Markov equivalent of (a), (b) or (c). Recalling the 3 structures that characterize conditional independence, from this example it can be concluded that Causal Chains and Common Causes structures are equivalent to each other, whereas Common Effects structure is not Markov equivalent to either Causal Chains structure or Common Causes structure.

2.1.2 DYNAMIC BAYESIAN NETWORK

A Dynamic Bayesian Network is an extension to a BN in order to model discrete-time stochastic process. In other words, DBN adds the time-dimension to a BN. This is done by linking one BN to another BN. See Figure 5, for an example of a DBN. The edges between two variables from the same timestep are named *intra-slice edges*, whereas the edges between two variables from two different timesteps are called *temporal (inter-slice) edges*. In (Murphy, 2002), DBN is also denoted as a pair$(B_0, B_{\rightarrow})$. B_0 represents an usual BN that keeps the information such as variables in the initial timestep, intra-slice edges and CPT for the initial timestep. Meanwhile, $B_{\rightarrow}$ contains information about the variables in the next timestep, temporal edges and the CPT that already incorporates the influence of the variables in the previous timestep.

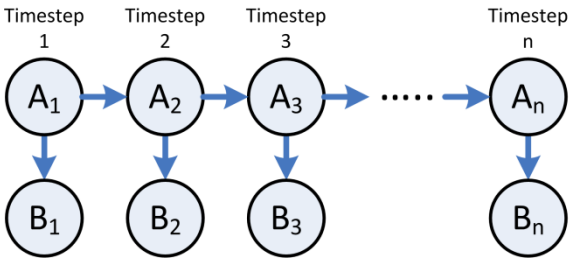


Figure 5 An example of Dynamic Bayesian Network

In principle, it is not necessary that the intra-slice and temporal edges do not change in each time step. There may be edges addition, deletion or reversal occurs in the next timestep compared to edges in the current timestep as long as the DBN in its fullness meets the acyclic constraint. If the edges do stay constant in each timestep, then the DBN possesses the first-Order Markov property. Suppose that timestep t is the current timestep, then first Order Markov implies that variables in timestep t are only dependent on the variables in timestep $t-1$. We can generalize the concept to n^{th}-order Markov property which means that variables in timestep t are only dependent on all variables between timestep $t-1$ to timestep $t-n$. Having the n^{th}-Order Markov property, it makes the belief updating in DBN more feasible. The belief updating can be done by putting the sliding window with the width of n timesteps in the DBN and updating the belief only for variables in that sliding window with the similar belief updating technique for BN. The details of belief updating by means of sliding window can be found in (Korb, 2004) and (Murphy, 2002).

2.1.3 SPEECH RECOGNITION WITH DYNAMIC BAYESIAN NETWORK

Extending BN to DBN enables modeling stochastic processes. Hence, DBN can be applied to the problem of Automated Speech Recognition (ASR). The most popular ASR systems model the transition of phonemes and language model by means of Hidden Markov Model (HMM). The strength of HMM is the ability to dynamically align the acoustic features to the length of the phoneme by introducing self-loops. In order to apply DBNs to recognize speech, DBNs should be modeled as such that dynamic alignment is also possible.

2.1.3.1 MODELLING DBN FOR AUTOMATED SPEECH RECOGNITION

The application of DBN in ASR is introduced by Geoffrey Zweig in (Zweig G. G., 1998) and (Zweig G. G., 2003). To illustrate how DBN can be applied to ASR system, let's consider a simple HMM that can recognize one word: "NO". This HMM is depicted in Figure 6. As we can recall from the previous section, each node in BN and DBN represent a random variable. An example of simple translation from HMM whose node represents the state to a DBN is portrayed in Figure 7.

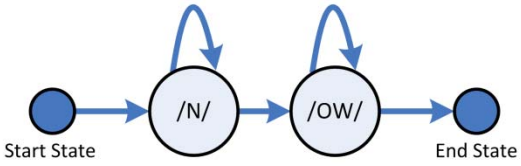


Figure 6 HMM for recognizing the word "NO"

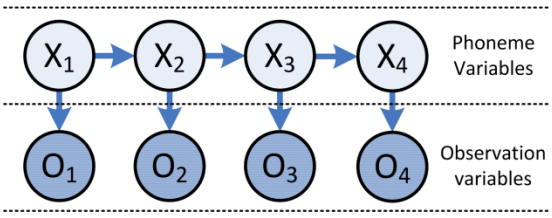


Figure 7 A DBN for recognizing word "NO"

Before we analyze the DBN in Figure 7, there are two types of variables in DBN for ASR, viz. hidden variables (X_i) and observation variables (O_i). The hidden variables are variables whose values are not yet known and it is a common assumption that these variables are discrete variables in ASR. On the other side, observation variables are the variables whose value are known and inputted to the DBN. In Figure 7, the observation variables are the nodes that are shaded and in ASR these variables are usually the observed acoustical features.

In the figure, phoneme variables, which in this case only have two possible values: "$/N/$" and "$/OW/$", are the hidden variables. Also notice that in DBN the time step is explicitly represented in the model. Each variable in DBN belongs to certain time-slice, for example X_1 and O_1 belong to the first time step. On the other hand, since a HMM defines a possible state sequence, it makes it difficult to perceive the time steps in the model. To make it easier to perceive the time steps in the model, HMM can be unrolled for each time step. This special graph is known as Trellis (Jurafsky & Martin, 2009).

The task of inference in DBN is simply assigning each hidden variables a value. The most probable assignments can be obtained by inputting values to the observation variables and let the information flows to hidden variables. An example of such assignment for DBN in Figure 7 is $X_1 =$ "$/N/$", $X_2 =$ "$/N/$", $X_3 =$ "$/OW/$", and $X_4 =$ "$/OW/$". In this simple example, the transition from "$/OW/$" to "$/N/$" should be made impossible, otherwise an assignment such as $X_1 =$ "$/N/$", $X_2 =$ "$/N/$", $X_3 =$ "$/OW/$", $X_4 =$ "$/N/$" can occur. This is undesirable since such assignment would also not be possible in the original HMM. To avoid this, the value in CPT for $P($"$/N/$"|"$/\mathrm{OW}/$"$)$ should equal zero.

The DBN presented in Figure 7 may work for the word "NO", but what happens when a word such as "DIGIT" that are composed of the sequence of phonemes "$/D/$","$/IH/$","$/JH/$","$/IH/$","$/T/$" is required to be recognized? This reveals the problem in similar structures such as depicted in Figure 7 when dealing with recurring phonemes in one word. Suppose that to make sure that first "$/IH/$" in the sequence does not switch to "$/T/$", $P($"$/T/$"|$/\mathrm{IH}/)$ is made zero. Given that the CPT of each hidden variable does not change in time, this implies that the second "$/IH/$" will never be able to switch to "$/T/$" to end the word. Ergo, all assignments that end with "$/T/$" will always have probability of zero. A simplistic solution to this problem is by creating two new phonemes "$/IH1/$" and "$/IH2/$" to avoid recurring phonemes. Instead of "$/D/$", "$/IH/$", "$/JH/$", "$/IH/$", "$/T/$", the word $DIGIT$ is now composed of "$/D/$", "$/IH1/$", "$/JH/$", "$/IH2/$", "$/T/$". However, this solution will become too complex as the amount of words that must be recognized increases. Another solution proposed in (Zweig G. G., 1998) and (Zweig G. G., 2003) is by introducing transition variables and position variables. As the name suggested, position variable is a variable that indicates in which position in the phonemes sequence it belongs and transition variable is a binary variable that indicates whether there a phoneme transition occurs. An example of DBN with position and transition variable is depicted in Figure 8.

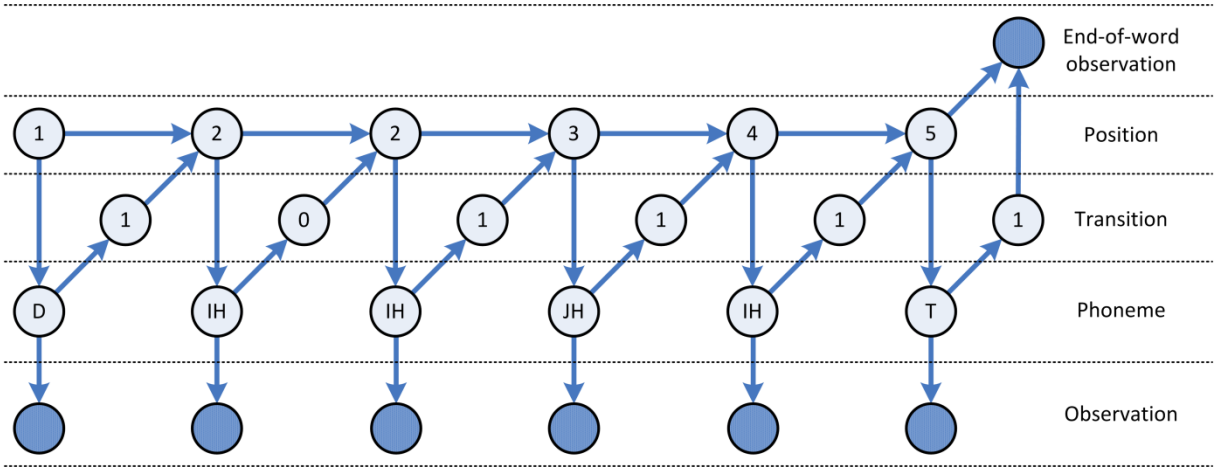


Figure 8 An Example of DBN to recognize the word "DIGIT" by means of Transition and Position variables

Notice that position variable is deterministically defined. Suppose that the position in the previous time-slice is n, the position for current time-slice is $n + 1$ if and only if the transition variable has a value 1 (true). Moreover, the CPT of phoneme variable is also deterministic. It can be clearly seen that given the position, the corresponding phoneme can be deterministically chosen. The CPT's that are stochastic are the CPT's of observation variable and transition variable. The CPT of transition variable describes the dynamic of the

phoneme transition and the CPT of the observation variable describes how likely it is that the observed acoustic features belongs to a certain phoneme.

Using the aforementioned structure, a speech recognizer for isolated words can be realized. Advancing to recognizer for continuous speech can be done by introducing transition and position for word level. Continuous speech will also demand knowledge about language model such as bi-gram or tri-gram. For more details about the structure of DBN for bi-gram and tri-gram, we refer to (Bilmes & Bartels, 2005).

2.1.3.2 EXAMPLES OF MODEL STRUCTURES FOR DBN

Knowing how the translation from HMM to DBN is done and how the language model such as bi-gram and tri-gram can be applied already gives us the ability to mimic HMM-based speech recognition system for DBN-based speech recognizer. However, this does not necessarily improve the quality of the speech recognizer. The speech recognizer still ignores the contextual and acoustical information that can be important in recognizing certain words or sentences. In (Wiggers, 2008), three kinds of knowledge that are influential to how people speak and listen are defined. These are:

1. *User Knowledge* (Dialects and languages, Gender, Social Group, Age)
2. *Conversational knowledge* (Speaking style, Interaction, Topic)
3. *World knowledge* (Background knowledge, Other modalities)

By also incorporating this knowledge in the DBN system, the speech recognizer will become aware of the characteristics of the given knowledge and hence improves the quality of the recognized speech.

In a DBN-based speech recognizer, adding this information about knowledge into the system becomes a lot easier since this information can be modeled as random variables in the DBN. When the information is already translated into variables, the next thing to find out is how these variables relate to other variables. Since the introduction of DBN-based speech recognition system, there are few model structures proposed for DBN. In the thesis in which Zweig introduces speech recognition with DBN (Zweig G. G., 1998), four different model structures are also proposed. Those are:

1. Articulatory Modeling (Figure 9); The model structure incorporates the knowledge of the tongue position and the configuration of the lips. This is argued to be influential in producing the sound of pronounced words.

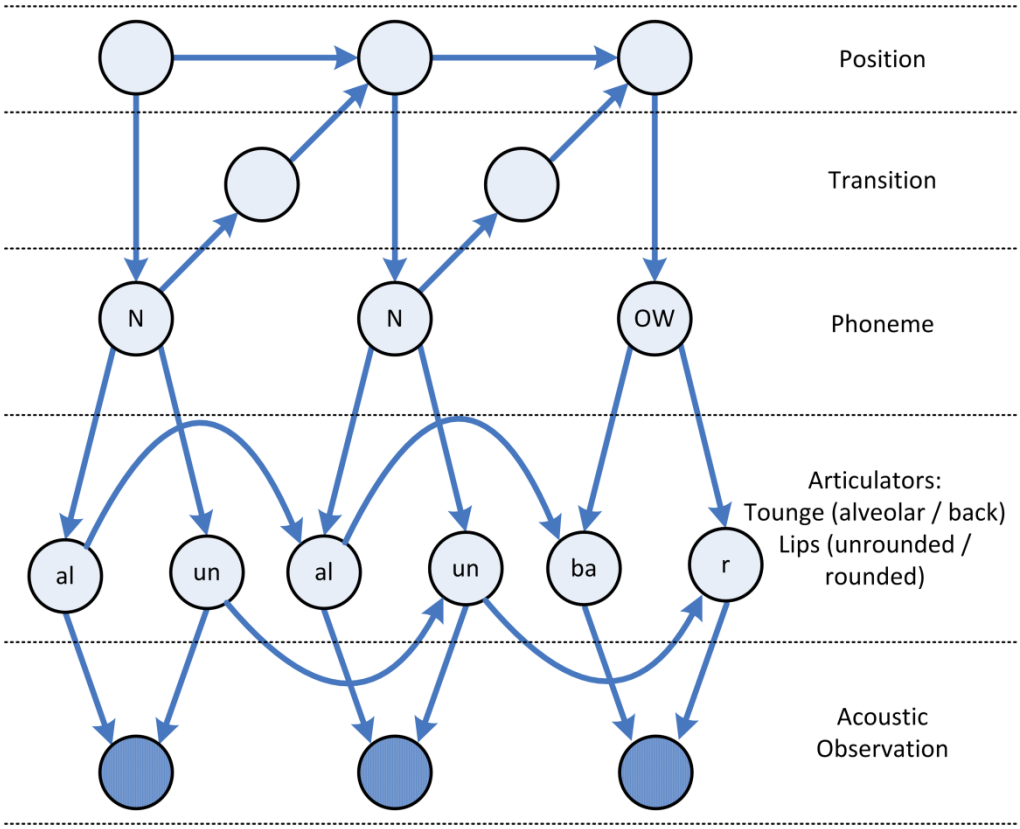


Figure 9 Dynamic Bayesian Network for ASR system that incorporates Articulatory model

2. Modeling Speaking Style (Figure 10 & Figure 11); Variables such as gender, accent and rate of speech are important characteristic in determining to which phoneme the observed sound belongs.

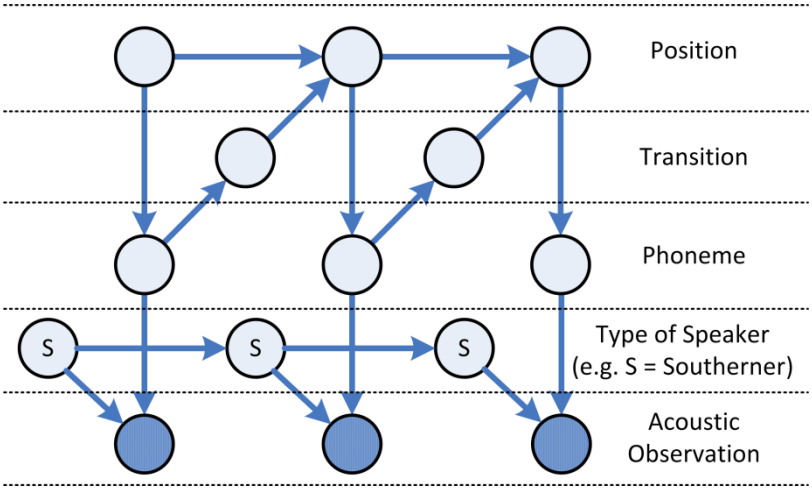


Figure 10 A Dynamic Bayesian Network to incorporate Type of Speaker in ASR system

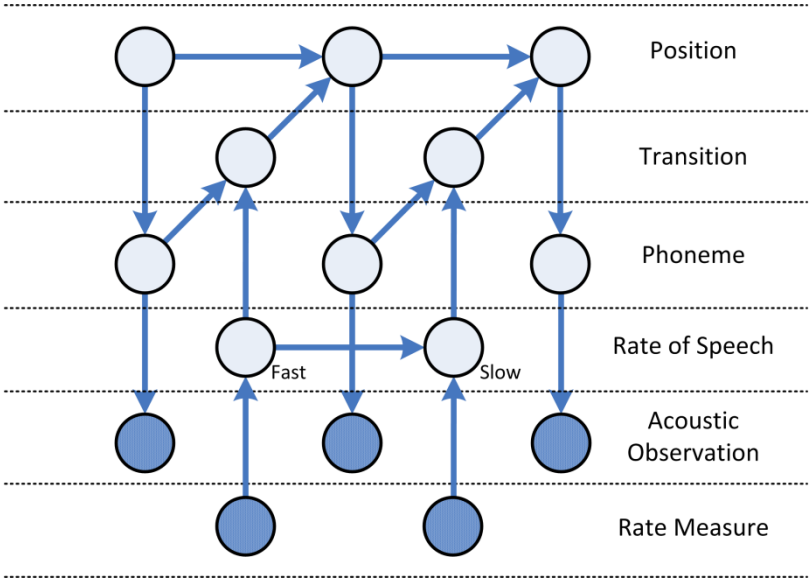


Figure 11 A Dynamic Bayesian Network to incorporate Rate of Speech in ASR system

3. Noise Modeling (Figure 12); In addition to modeling the speech, the noise model is also considered. In this way, the speech recognizer will perform better in noisy environment.

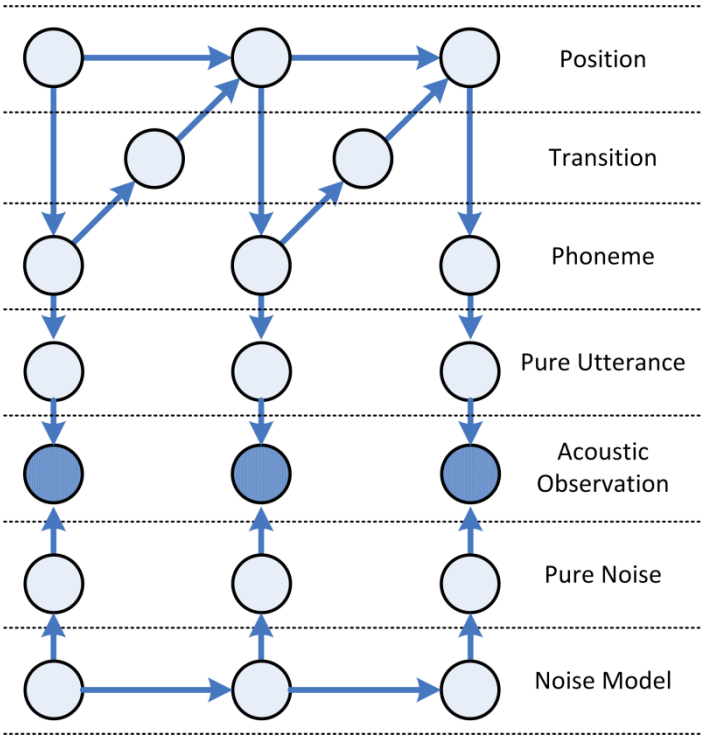


Figure 12 Dynamic Bayesian Network for ASR system in noisy environment

4. Perceptual and Combined Models (Figure 13); Modeling not only how the sound is produced but also how it is perceived by the listener.

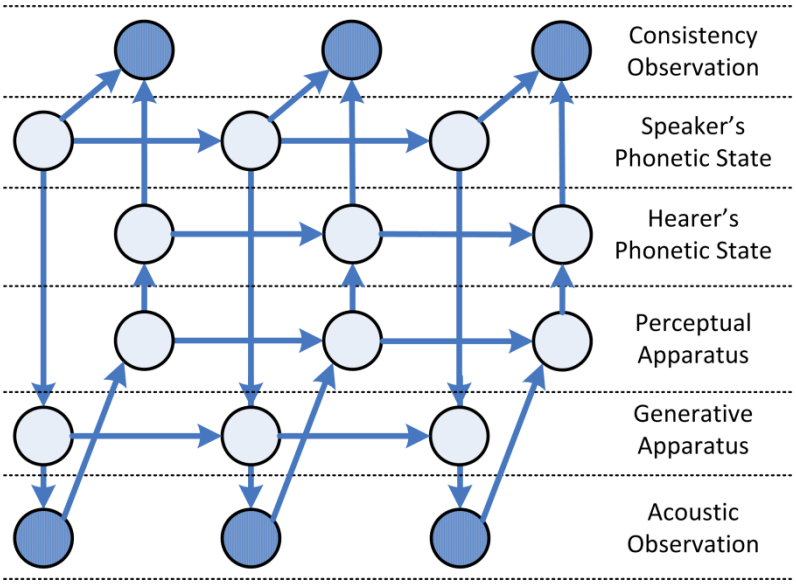


Figure 13 Dynamic Bayesian Network for ASR system that incorporates Perceptual and Combined Models

Moreover, in (Daoudi, Fohr, & Antoine, 2003) a DBN structure is proposed that not only considers the temporal domain of the audio signal but also frequency domain. Prior to analyzing the acoustical features, the audio signal is divided into different sub-bands. The idea behind multi-band speech recognition is to mimic the human auditory system. It is suggested that human auditory system also processes the audio signal locally for each sub-bands before recognizing the spoken words. Extending the model for multi-band speech recognizer in HMM is difficult because the sub-bands are likely to be dependent of each other. In DBN, it is easier because dividing the audio signal into sub-bands means that for each time slice there will be multiple acoustical observation variables. The dependencies between sub-bands can be modeled by putting an edge between the variables. The model structure for multi-band speech recognizer is depicted in Figure 14.

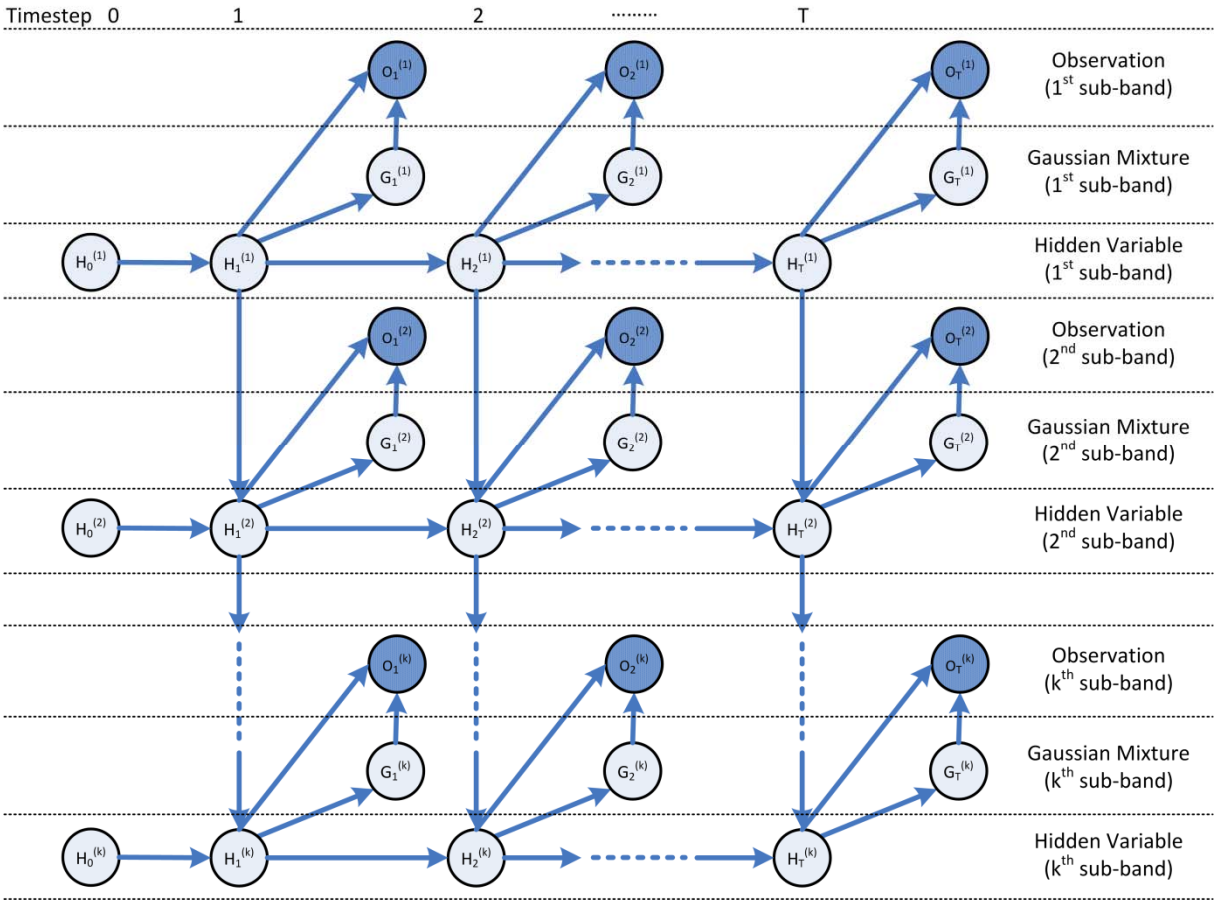


Figure 14 Dynamic Bayesian Network for k-band ASR system

In (Bilmes & Bartels, 2005), the authors give a suggestion in how the trigram as an example in language model can be applied to a DBN-based Speech Recognition system. This model that is depicted in Figure 15 incorporates phoneme as low level structure and word as high level structure. Since trigram implies that the next word is predicted by means of previous 2 words, the authors introduces “previous-word” variable besides the usual “word” variable.

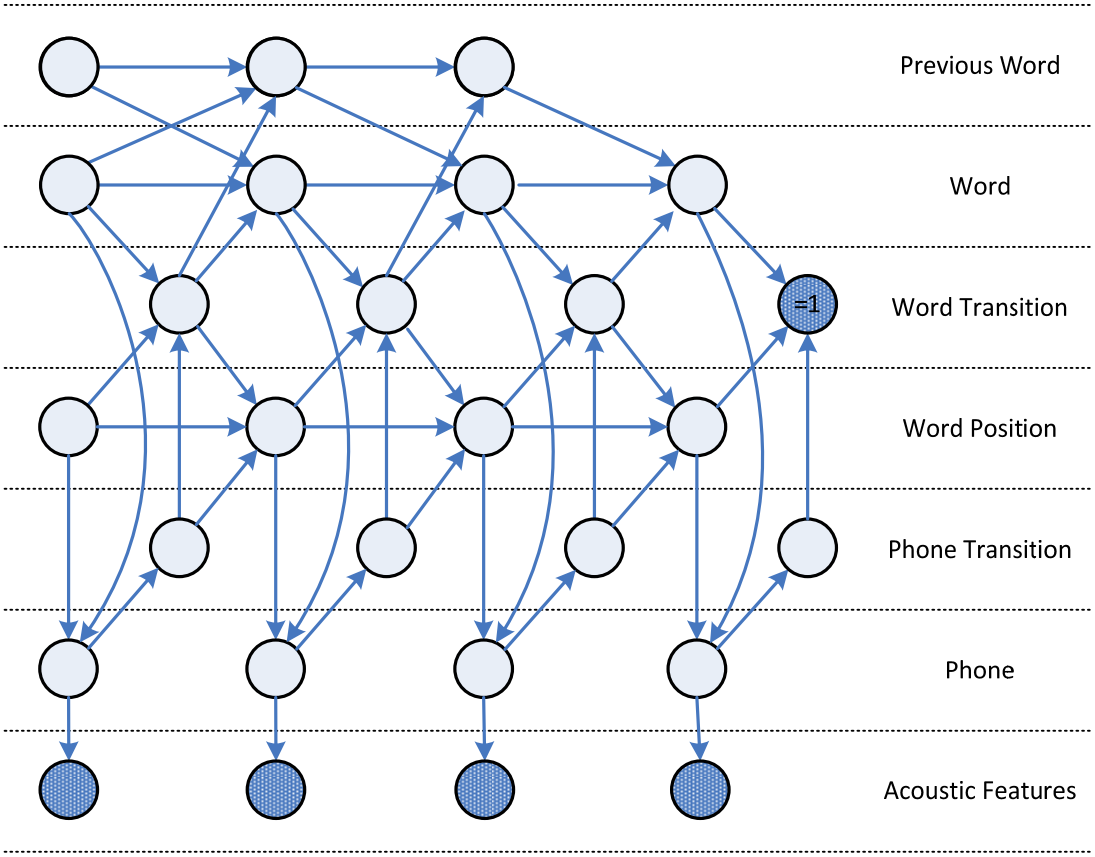


Figure 15 Dynamic Bayesian Network of trigram language model

The aforementioned structure models are examples of proposed structure models for DBN-based Speech Recognition system. Besides the examples discussed above, there are more papers or publications written in which structure models are proposed. However, the proposed structure only works when certain knowledge is known. This raises the question whether it is necessary to invent a DBN structure every time a new relevant knowledge has to be incorporated in the model. There is a necessity for a mechanism where the DBN structure can be learned automatically. This will ease the modeling of DBN structure, when all of user knowledge, conversational knowledge and world knowledge or only parts of it are known. Fortunately, by means of analyzing the empirical data the structure of the DBN can be learned. The structure learning techniques are discussed in *Learning Structure* chapter of this report.

2.2 PARAMETER LEARNING

In this chapter, we will address techniques of learning the parameters of BNs or DBNs assuming that the structure is already known. Learning the parameters involves estimating the Conditional Probability Table (CPT) or Conditional Probability Distribution (CPD) of each variable in the structures. In other words, given X_i the i^{th} variable and Π_i are the parents of i^{th} variable, the task of learning parameters are the task of defining:

$$P(X_i = j|\Pi_i = k), for\ all\ i, j\ and\ k$$

In this chapter we will only deal with estimating CPTs of discrete variables with discrete parents and estimating CPD of continuous variables with discrete parents, because for these cases exact computation when performing inference is ensured. In the last section, we will introduce the basic idea of Expectation

Maximization (EM) algorithm. EM algorithm is an iterative parameter estimation algorithm in the presence of hidden variables.

2.2.1 DISCRETE VARIABLE WITH DISCRETE PARENTS

In learning parameters for discrete variables with discrete parents, there are two methods to do it. To explain these two methods, let's first formulate the problem of estimating the parameters. Given that D is the training data and θ is the parameter that we are estimating, then learning parameter is simply finding the parameters that maximize the posterior probability $P(\theta \mid D)$. By using Bayesian theorem, this posterior probability can derived to the following equation:

$$P(\theta|D) = \beta P(D|\theta)P(\theta)$$

where β is the inverse of the probability of the training data. When parameters that maximize $P(\theta|D)$ is meant to be found, then β can simply be ignored.

2.2.1.1 MAXIMUM LIKELIHOOD ESTIMATION

It is a very common to assume that the training data is taken from the multinomial distribution. This will help assessing the likelihood probability $P(D|\theta)$. First, let's denote θ_{ijk} as the probability $P(X_i = j|\Pi_i = k)$. For the sake of simplicity, consider a variable X with 2 possible states: x and $\neg x$ and the parent of this binary variable is also binary variable Y with 2 possible states: y and $\neg y$. In this example, the parameters for variable X that should be found are $\theta_{X,y,x}$ and $\theta_{X,\neg y,x}$. Given that the training data D contains the following observations: $\{x, y\}, \{\neg x, y\}, \{x, y\}, \{\neg x, \neg y\}$ then the likelihood probability as follows:

$$P(\{x, y\}, \{\neg x, y\}, \{x, y\}, \{\neg x, \neg y\}|\theta) = \theta_{X,y,x}(1 - \theta_{X,\neg y,x})$$

ML estimation will then choose the parameters that maximizes the $P(\{x, y\}, \{\neg x, y\}, \{x, y\}, \{\neg x, \neg y\}|\theta)$. By solving them analytically, the estimated parameter for $\theta_{X,y,x} = \frac{|\{x,y\},\{x,y\}|}{|\{x,y\},\{\neg x,y\},\{x,y\}|}$. Generalizing for samples from multinomial distribution, the estimated parameters for MLE is:

$$\theta_{ijk} = \frac{N_{ijk}}{\sum_{k'} N_{ijk'}}$$

where N_{ijk} is the count of observation $\{j, k\}$ in the database.

2.2.1.2 MAXIMUM A POSTERIORI ESTIMATION

Instead of ignoring the prior probability like ML estimation suggested, MAP Estimation puts the prior probability of the parameters into consideration. The most common choice for the distribution for prior probability is the Dirichlet distribution because Dirichlet distribution is the prior conjugate of multinomial distribution. This property is very essential since choosing Dirichlet distribution make it possible to compute the posterior probability in closed form. The Dirichlet prior is defined as:

$$P(\theta_{ij}) = D(\alpha_{ij1}, \dots, \alpha_{ijr_i}) = \prod_{k=1}^{r_i} {\theta_{ijk}}^{\alpha_{ijk-1}} \frac{1}{B(\alpha_{ij1}, \dots, \alpha_{ijr_i})}$$

Where:

- α_{ijk} is the hyperparameter of the Dirichlet distribution
- r_i is the number of states of variable X_i

- $B(\alpha_{ij1}, \dots, \alpha_{ijr_i})$ is the beta function

As the consequence of Dirichlet distribution as conjugate prior, the posterior probability is also Dirichlet distributed. Given that the prior is $D(\alpha_{ij1}, \dots, \alpha_{ijr_i})$, the posterior probability becomes $D(\alpha_{ij1} + N_{ij1}, \dots, \alpha_{ijr_i} + N_{ijr_i})$. To find the estimated parameters, MAP estimation maximizes this posterior probability. Analytical solution of it is the following equation:

$$\theta_{ijk} = \frac{\alpha_{ijk} + N_{ijk} - 1}{\sum_{k'} \alpha_{ijk'} + N_{ijk'} - 1}$$

However, assessing the Dirichlet distribution involves knowing the hyperparameters α_{ijk}. This α_{ijk} should be chosen carefully in order to fit the distribution of the parameters. In the appendix of (Murphy, 2002), the author discussed some possibilities of choosing the hyperparameters. Notice that when all α_{ijk} is assigned as 1, MAP estimation becomes identical to ML estimation.

2.2.2 CONTINUOUS VARIABLE WITH DISCRETE PARENTS

Similar to the approach for discrete variable, there are two approaches in estimating the parameters, viz., Maximum Likelihood (ML) Estimation and Maximum A Posteriori (MAP) Estimation. However, in the case of continuous variable, we have to make an assumption about the distribution of that corresponding continuous variable. The most common choice for this distribution is Gaussian distribution. Ergo, in this section we also assume that the variable is Gaussian distributed that has two parameters: mean (μ) and covariance (σ).

2.2.2.1 MAXIMUM LIKELIHOOD ESTIMATION

Given that X is a continuous variable with parents Π, then the task of learning parameters boils down to finding the μ_j and Σ_j where $j = 1, \dots, m$ and m is the number of possible instances for Π. Hence, ML Estimation can be done by maximizing $P(D|\mu_j)$ and $P(D|\Sigma_j)$ for each instance of Π. Maximizing $P(D|\mu_j)$ equals to calculating the sample mean:

$$\hat{\mu} = \frac{1}{n} \sum_{i=1}^{n} d_{\Pi=j_i}$$

where $d_{\Pi=j_i}$ is the i^{th} sample of variable X when the $\Pi = j_i$.

While estimating the covariance can be done by the following equation:

$$\hat{\Sigma}_j = \frac{1}{n-1} \sum_{i=1}^{n} (d_{\Pi=j_i} - \hat{\mu}_j)(d_{\Pi=j_i} - \hat{\mu}_j)^{\mathrm{T}}$$

2.2.2.2 MAXIMUM A POSTERIOR ESTIMATION

The MAP Estimation can be done by choosing prior distribution for μ and Σ. The suitable choice for the joint prior distribution of μ and Σ is normal-Wishart distribution. Unfortunately, there is no analytical solution for maximizing the posterior probability $P(\mu, \Sigma|D)$.

2.2.3 EXPECTATION MAXIMIZATION ALGORITHM

EM algorithm is an iterative parameter estimation that also takes the presence of hidden variable into account. Let consider estimating parameter θ of random variable X by means of ML estimation. Since $\ln(x)$ function is a

strictly increasing function, ML estimation can be obtained by maximizing log likelihood function. The log likelihood function is as follows:

$$L(\theta) = \ln P(X|\theta)$$

By also considering hidden variables Z, the log likelihood function can be written as follows:

$$L(\theta) = \ln \sum_z P(X|z,\theta)P(z|\theta)$$

Furthermore, since EM algorithm is an iterative algorithm, let's denote the parameter estimation at n^{th} iteration as θ_n. In order to find the parameter that maximize $L(\theta)$ at the next iteration, we want to maximize the difference of log likelihood between the previously estimated parameter and the searched parameter:

$$L(\theta) - L(\theta_n) = \ln P(X|\theta) - \ln P(X|\theta_n) = \ln \textstyle\sum_z P(X|z,\theta)P(z|\theta) - \ln P(X|\theta_n)$$

By applying Jensen's inequality and rearranging the terms, we can conclude that:

$$L(\theta) - L(\theta_n) \geq \sum_z P(z|X,\theta_n) \ln \frac{P(X|z,\theta)P(z|\theta)}{P(z|X,\theta_n)P(X|\theta_n)} = \Delta(\theta|\theta_n)$$

We denote the term on right side of ≥ sign as $\Delta(\theta|\theta_n)$. Let's also denote $L(\theta_n) + \Delta(\theta|\theta_n)$ as $I(\theta|\theta_n)$, so we can rearrange the equations into the following equation:

$$L(\theta) \geq I(\theta|\theta_n)$$

Since $I(\theta|\theta_n)$ is bounded above by $L(\theta)$, maximizing $L(\theta)$ is equivalent to maximizing $I(\theta|\theta_n)$. EM algorithm chooses the estimated parameter in the next iteration by picking θ that maximizes $I(\theta|\theta_n)$. This is described in the next equation:

$$\begin{aligned}
\theta_{n+1} &= \operatorname*{argmax}_\theta I(\theta|\theta_n) \\
&= \operatorname*{argmax}_\theta \left\{ L(\theta_n) + \sum_z P(z|X,\theta_n) \ln \frac{P(X|z,\theta)P(z|\theta)}{P(z|X,\theta_n)P(X|\theta_n)} \right\} \\
&= \operatorname*{argmax}_\theta \left\{ \sum_z P(z|X,\theta_n) \ln P(X,z|\theta) \right\}
\end{aligned}$$

The equation above demonstrates the 2 essential steps of EM algorithm:

- E-step: Determine the $\sum_z P(z|X,\theta_n) \ln P(X,z|\theta)$
- M-step: Maximize $\sum_z P(z|X,\theta_n) \ln P(X,z|\theta)$ with respect to θ

The EM algorithm stops when the estimated θ does not change anymore. However, it depends on the likelihood function, the EM algorithm can get stuck in the local minima and thus the EM algorithm does not always give optimal result.

The EM algorithm can be applied for example to estimate the parameters of a mixture Gaussian distribution. In estimating the parameters of a mixture Gaussian distribution, we can consider the index to one of the Gaussian distributions as the hidden variable.

2.3 STRUCTURE LEARNING

In learning the structure of the (Dynamic) Bayesian Network, there are three axes that can be utilized to categorize structure learning algorithms (Murphy, 2002). These are:

1. Hypothesis Space
2. Evaluation Function
3. Search Algorithm

The first axis, *hypothesis space*, tells us in which form the structure is represented. The choice of the hypothesis space is essential since it can affect the complexity of the algorithm. Provided that a Directed Acyclic Graph (DAG) is a graphical representation of a Bayesian Network (BN), a DAG is the most straight-forward choice of hypothesis space. However, Directed Acyclic Graphs are known for their high complexity. Given N variables in the network, there are $2^{O(N^2 \log N)}$possible DAGs to search through (Robinson, 1977). This exponential growth of possibilities makes DAG an infeasible choice for hypothesis space, mainly if the structure has many variables. It is quite common that two DAGs that are distinguishable in terms of the directions of their edges cannot be distinguished empirically (See Markov Equivalence Section). These two DAGs are considered to be equivalent of each other and a collection of these equivalent DAGs is called a *Pattern*. According to (Gillispie & Perlman, 2001), choosing patterns instead of DAGs as the hypothesis space will reduce the search space up to 3.7-14 times. The search space can still be further reduced if the variable orderings are known. All DAGs or patterns that do not satisfy the constraints put by the variable orderings will not be considered, hence narrowing down the search space.

The *evaluation function* is a metric in which hypotheses from the hypothesis space can be compared for their closeness to the structure in the real world system. The calculation of the evaluation function is based on the training data from the real world system that is made available. However, applying a simple likelihood function as the evaluation function will not suffice in this case because a likelihood function will always tend to a more complex graph as the optimal structure. It indicates an overfitting to training data which is undesirable since we wish the learned structure to be as general as possible. To reduce the tendency to learn complex graphs, the prior probability distribution of graphs can be chosen in such way that it assigns a lower prior probability to more complex graphs. Other examples of evaluation functions are Minimum Message Length (MML) (Wallace & Boulton, 1968) and Minimum Description Length (MDL) (Lam & Bacchus, 1994). Both evaluation functions are based on Shannon's measure of information. MDL and MML evaluation function find a trade-off between model simplicity and how well the training data conforms to the corresponding model. More details about MDL and MML will be discussed in the later sections in this chapter.

The third and last axis is the *search algorithm*. When an optimal structure is desired, it is necessary to iterate through all possible structures. This is however infeasible because the number of possible structures grows exponentially as the number of variables increases. Instead of such a brute-force search algorithm (iterating through all possible structures), a local search algorithm can be applied. A local search algorithm starts from an initial structure and it iteratively moves to the neighbour structure that is more optimal than the previous structure. This makes the local search algorithm more feasible since not all possible structures will be iterated. Nonetheless, as a result of an arbitrary choice of initial structure it is possible that the optimal structure according to a local search algorithm is nothing more than a local minimum / maximum (Depends on the evaluation function, if an optimal structure has a high value then it's a local maximum, vice versa). This can be helped by choosing the initial structure carefully, having more than one initial structure or adding a stochastic element in the algorithm.

These three axes will be used as a framework to discuss the structure learning algorithms in the subsequent sections. The first section will be a survey of techniques in learning structure in BNs and in the next section extensions needed in order to apply these techniques will be discussed. The final section will give an insight

about learning structure not only from data but also applying the prior knowledge into the structure learning algorithm.

2.3.1 LEARNING STRUCTURE IN BAYESIAN NETWORKS

This section will give the basic idea of the techniques that can be applied to learn the structure in BNs. The techniques that are introduced cover choosing of hypothesis space, deriving the evaluation function, picking the search algorithm or the combination of the aforementioned items.

2.3.1.1 INDUCTIVE CAUSATION (IC) ALGORITHM

The IC algorithm is one of the pioneering algorithms for learning structure in BNs. The IC algorithm which is proposed by Verma and Pearl is based on the equivalence of BNs and the conditional independence between the variables (Verma & Pearl, 1990). Box 1 gives the details of IC algorithm.

1. For each pair of variables A and B, search for a set of variables S_{AB} that fulfill the following the condition:
    a. Neither A nor B is a member of S_{AB} $(A, B \notin S_{AB})$
    b. Given the evidence of S_{AB}, A and B are independent $(A \perp B | S_{AB})$
    - If such set S_{AB} does **not** exist, then place an undirected edge between A and B
2. As the consequence of Step 1, a pair of non-adjacent variables has a set variable S_{AB} that fulfill the condition stated at Step 1. For each pair of non-adjacent variables A and B with a common neighbour C, check if $C \in S_{AB}$
    - If it is not, then add arrowheads pointing at C realizing a v-structure, (i.e. $A \rightarrow C \leftarrow B$)
3. For each undirected edge B and C in the graph, add arrowheads pointing at C $(B \rightarrow C)$ if and only if one of the following conditions is satisfied:
    a. If there is such undirected chain $A - B - C$ from Step 2 and now between A and B there is a directed edge $A \rightarrow B$
    b. Directing $B \leftarrow C$ will introduce a cycle
    - Continue iterating through all the undirected edges until one such pass fails to direct any edges.

Box 1 Inductive Causation Algorithm

Instead of comparing DAGs, the IC algorithm constructs the most optimal DAGs. The algorithm begins with iterating all pairs of variable A & B and assigning an edge between them only when A & B directly influence each other. This is done by investigating the existence of such a set of variables S_{AB}. In this manner, unnecessary edges between variables can be avoided causing the complexity of the DAG to be kept low. The second step checks those pair of variables A and B that are not connected by edges and have a common neighbour (call this common neighbor C). Since A and B are not connected, then there must be a set of variables S_{AB} that makes A and B independent of each other. If C is not one of the variables in S_{AB}, it implies that $\neg(A \perp B | C)$. This is a property of v-structure (See Section 2.1.1.2). If C is one of the variables in S_{AB}, then we can't say anything about the edges between A, B and C because it can either be a causal chain or a common cause. The final step replaces each remaining undirected edge with directed edges in such a way that no new v-structure and no cycle are introduced.

The IC algorithm lays the ground for exact structure learning algorithms. By means of knowledge about the conditional dependence between variables, the algorithm can easily construct a DAG that represents the system in the real world. However, such knowledge is not always available. Most of the time, such knowledge

still has to be created on the basis of the empirical data from the real world system. Another drawback of the IC algorithm is that all pairs of variables should be checked and the existence of such a set of pair variables S_{AB} has to be investigated. Given that there are N variables, there are $\frac{N!}{2!(N-2)!}$ pairs of variables that have to be iterated and for each iteration there are $\binom{N-2}{1} + \binom{N-2}{2} + \cdots + \binom{N-2}{N-2}$ candidates for S_{AB}. This clearly shows that when there are lots of variables in the structure then IC algorithm will become infeasible since the iteration number will grow exponentially.

As a more feasible variant of the IC algorithm, the PC algorithm is proposed in (Spirtes, Glymour, & Scheines, 2001). The PC algorithm addresses the two drawbacks that IC algorithms have and give an alternative to solve them. To obtain the knowledge of conditional independence suggested by IC algorithm, a partial correlation can be calculated from the empirical data and infer the conditional independence from it. Furthermore, instead of constructing the DAG from a simple DAG without edges, the PC algorithm starts with a fully connected undirected graph and it removes the edges one by one on basis of conditional independence. This reduces the iteration number and hence makes PC algorithm more feasible. However, the PC algorithm doesn't mimic the exactness of the IC algorithm especially for structures with a high number of variables (Korb, 2004).

2.3.1.2 K2 ALGORITHM

Instead of learning the structure of a BN by constructing the BN like IC and PC algorithm, K2 compares all BNs by means of a metric calculated using an evaluation function. The evaluation function of the K2 algorithm estimates the joint probability of the hypothesized structure (h_i) and the evidence (e). In this case, the evidence refers to the training data from the real world system. To calculate the evaluation function, K2 depends on the following assumptions (Cooper & Herskovits, 1992):

1. The training data are joint samples and all variables in the structure are discrete. This assumption yields: $P(h_i, e) = \int_\theta P(e|h_i, \theta) f(\theta|h_i) P(h_i) d\theta$, where θ is the parameter of the structure and $f(\theta|h_i)$ is prior density over parameters given the structure
2. The training data is taken from an independent and identical distribution. This yields: $P(e|h_i, \theta) = \prod_{k=1}^{K} P(e_k|h_i, \theta)$. Given that K is the number of samples in the training data, from first and second assumption, it can be derived: $P(h_i, e) = p(h_i) \int_\theta f(\theta|h_i) \prod_{k=1}^{K} P(e_k|h_i, \theta)\, d\theta$
3. There are no samples in the training data with missing values. However if such case does exist, a Gibbs sampling procedure can be applied
4. $P\left(X_k = x \middle| h_i, \theta, \pi(X_k)\right)$ is uniformly distributed
5. $P(h_i)$ is uniformly distributed.

Under these assumptions, the joint probability can be derived into the following equation:

$$P_{CH}(h_i, e) = P(h_i) \prod_{k=1}^{N} \prod_{j=1}^{|\Phi_k|} \frac{(S_k - 1)!}{(S_{kj} + S_k - 1)!} \prod_{l=1}^{S_k} \alpha_{kjl}!$$

where:

1. N is the number of variables in the structure
2. $|\Phi_k|$ is the number of possible states of $\pi(X_k)$ (The parent of variable X_k)
3. S_k is the number of possible states of variable X_k
4. α_{kjl} is the number of samples in the training data where X_k is assigned to its l^{th} state and $\pi(X_k)$ is assigned to its j^{th} state
5. S_{kj} is the number of samples in the training data where $\pi(X_k)$ is assigned to its j^{th} value.

The details about how the joint probability is derived can be found in the appendix of (Cooper & Herskovits, 1992).

The equation of $P_{CH}(h_i, e)$, which is the evaluation function of the K2 algorithm, already gives a hint about the hypothesis space. Calculating $P_{CH}(h_i, e)$ will need the knowledge about the direction of all edges in the structure and only DAG can provide this knowledge.

Having the hypothesis space and the evaluation function, the search algorithm is the only component that is missing. Unfortunately, for an exact solution the K2 algorithms have to calculate $P_{CH}(h_i, e)$ for each possible DAG. Like we saw earlier, this will increase the search space exponentially. To make K2 more feasible, the K2 algorithm assumes that the ordering for all variables in the structure is already specified. This is, however, not always true.

Besides an exact solution which involves brute-force search through all possible DAGs, K2 also proposes a heuristic method. This heuristic method begins a structure without edge and for each variable it tries to find parents of corresponding variable by maximizes $g(i, \pi_i)$. $g(i, \pi_i)$ is defined as:

$$g(i, \pi_i) = \prod_{j=1}^{|\Phi_k|} \frac{(S_k - 1)!}{(S_{kj} + S_k - 1)!} \prod_{l=1}^{S_k} \alpha_{kjl}!$$

```
for i:= 1 to n do (n is the number of variables in the structure)
   π_i := {};
   P_old := g(i, π_i);
   OKToProceed := true;
   while OKToProceed and |π_i| < u do (u limits the number of parents)
      let z be the node that preceded X_i in variable ordering and maximizes
          g(i, π_i ∪ {z})
      P_new := g(i, π_i ∪ {z});
      if P_new > P_old then
         P_old := P_new;
         π_i := π_i ∪ {z};
      else
         OKToProceed = false;
   end
   Parent of X_i = π_i;
end
```

Box 2 Pseudo code of heuristic method for K2 algorithm

In Box 2, the pseudo code for the heuristic method is given (Cooper & Herskovits, 1992).

Because each variable is examined independently and there will be no parent added when $g(i, \pi_i)$ cannot be maximized anymore, the K2 algorithm becomes more feasible. The parameter u in the heuristic method can be used as an instrument to prevent the structure become too complex.

2.3.1.3 BDE METRIC

In (Heckerman, Geiger, & Chickering, 1995), the authors improve the metric that is introduced in the K2 algorithm and come up with the BDe metric. The abbreviation BDe stands for Bayesian Dirichlet Equivalence which points out that the metric is Bayesian with the Dirichlet prior and based on the Likelihood equivalence assumption. The metric from the K2 algorithm does not incorporate the Likelihood equivalence which is why the metric is often named BD (Bayesian Dirichlet) metric.

The likelihood equivalence assumption states that when two network structures are Markov equivalent and that the prior probabilities of both structure are non-zero, it means that the distribution of the parameters are also equivalent. Given that h_1 and h_2 are equivalent structures, ξ is the current state of information and Θ_U is the set of all parameters of the network with variables in domain U, the likelihood equivalence assumption can be expressed as follows:

$$P(\Theta_U|h_1,\xi) = P(\Theta_U|h_2,\xi)$$

This assumption implies that when structures h_1 and h_2 are equivalent, the parameters of h_1 can be derived from the parameters of h_2 and vice versa. By incorporating this assumption into the metric, the structure search space decreases since structures that are equivalent will have a similar value as the BDe metric. For the mathematical derivation from BD metric to BDe metric, we refer to (Heckerman, Geiger, & Chickering, 1995). Given that B_s^h is the hypothesized structure, D is the training data and ξ is is current state of information, the BDe metric is as follows:

$$P(D,B_s^h|\xi) = P(B_s^h|\xi)\prod_{i=1}^{n}\prod_{j=1}^{q_i}\frac{\Gamma(N'_{ij})}{\Gamma(N'_{ij}+N_{ij})}\prod_{k=1}^{r_i}\frac{\Gamma(N'_{ijk}+N_{ijk})}{\Gamma(N'_{ijk})}$$

Where:

- $\Gamma(x)$ is a gamma function with x as parameter
- n is the number of variables in the structure
- q_i is the number of possible states of i^{th} variable's parents
- r_i is the number of possible states of i^{th} variable
- N_{ijk} is the number of samples in the training data where i^{th} variable is assigned to its l^{th} state and its parent is assigned to its j^{th} state
- $N'_{ijk} = N'P(X_i = k, \Pi_i = j|B_s^h,\xi)$
- $N_{ij} = \sum_{k=1}^{r_i} N_{ijk}$
- $N'_{ij} = \sum_{k=1}^{r_i} N'_{ijk}$
- N' is equivalent sample size

Furthermore, it has been proven in (Heckerman, Geiger, & Chickering, 1995) that if the parameters of the structure are positive, which is always the case in BNs and DBNs, then the Dirichlet assumption is not really needed.

2.3.1.4 MDL CAUSAL DISCOVERY

MDL, which stands for Minimum Description Length, is a metric invented by Jorma Rissanen (Rissanen, 1978) based on the Shannon information measure ($I(m) = -\log P(m)$). MDL searches for a trade-off between low structure complexity and the fit of training data to the structure. This is done by minimizing the description of the structure and the training data. The trade-off can be expressed mathematically by deriving the Shannon information measure with Bayes Theorem:

$$\begin{aligned} P(h,e) &= P(h)P(e|h) \\ -\log P(h,e) &= -\log[P(h)P(e|h)] \\ -\log P(h,e) &= -\log P(h) - \log P(e|h) \\ I(h,e) &= I(h) + I(e|h) \end{aligned}$$

where $I(h,e)$ represents the joint information measure between the structure (h) and training data (e). This joint information measure can then be obtained by summing up $I(h)$ and $I(e|h)$. $I(h)$ or *hypothesis message* in this equation measures the complexity of the structure, while $I(e|h)$ or *data message* measures the

optimality of the structure assuming the data is generated by the model. Having $I(h,e)$ as our metric, we can compare structure by comparing $I(h,e)$. Furthermore, the learning structure of a BN can be done by searching for structure h_i with the smallest $I(e|h_i)$.

However, $I(h,e)$ that is proposed above is just a theoretical construction. At this point, we are more interested in the application of MDL in learning structure: How can we measure $I(h,e)$ for BN? In this subsection, two MDL-based metric for measuring the effectiveness of BN will be discussed, viz.: Lam and Bachus's MDL code and Suzuki's MDL code.

LAM AND BACCHUS'S MDL CODE

To measure $I(h,e)$ for a BN, Wai Lam and Fahiem Bacchus propose the following for hypothesis message (Lam & Bacchus, 1994):

$$I_{LB}(h) = \sum_{i=1}^{N} \left(k_i \log N + d(s_i - 1) \prod_{j \in \pi(i)} s_j \right)$$

Where

- N is the number of variables in the structure
- k_i is the number of parents of the i^{th} variable
- d is word size in bits
- s_i is number of states of i^{th} variable.

The hypothesis message according to Lam and Bacchus is characterized by two terms. The first term, $k_i \log N$, is the measure of complexity of the i^{th} variable only. The complexity of a variable is determined by the quantity of variable's parent; as the quantity of variable's parents increases, variable's complexity also increases. The second term, $d(s_i - 1)\prod_{j \in \pi(i)} s_j$, expresses the complexity of the variable with respect to parameters that are needed to be declared. In BN, this concerns the quantity of conditional probabilities that are needed to be declared in Conditional Probability Table (CPT) of corresponding variable.

Already having the hypothesis message, only the data message needs to be defined to calculate $I(h,e)$. Lam and Bacchus' MDL code proposes the following for data message (Lam & Bacchus, 1994):

$$I_{LB}(e|h) = K \sum_{i=1}^{N} H(X_i) - \sum_{i=1}^{N} H\big(X_i, \pi(i)\big)$$

where:

- K is the number of samples in the training data
- $H(X)$ is the entropy of variable X
  - $H(X) = -\sum_x P(X = x) \log P(X = x)$
- $H(X_i, \pi(i))$ is the mutual information between X_i and its parent set
  - $H\big(X_i, \pi(i)\big) = H(X_i) - H\big(X_i|\pi(i)\big) = \sum_{x_i, \phi(i)} P\big(x_i, \phi(i)\big) \log \frac{P(x_i, \phi(i))}{P(x_i)P(\phi(i))}$
  - Where $\phi(i)$ ranges over the distinct instantiations of the parent set $\pi(i)$

The data message in this case gives a measure of how much entropy (amount of uncertainty) remains given the information about the structure. The explanation of this can be seen from the equation. The first term of the equation is the measure of entropy which is measured by the well-known Shannon Entropy. This measures the amount of uncertainty in the information about the variable itself without any additional information. On the other side, the second term, $H\big(X_i, \pi(i)\big)$, is the decrease of entropy that information about the parents of the

variable can achieve. Subtracting the second term from the first term will obtain the remaining entropy for each variable given the information of the parents and summing them all up for each variable results in the remaining entropy given the whole structure.

In (Korb, 2004), a search algorithm that can be applied with this metric is introduced. This MDL causal discovery algorithm can be seen in Box 3.

1. Initial constraints are taken from a domain expert. This includes a partial variable order and whatever direct connections might be known
2. Greedy search is then applied
    - Every possible edge addition is tested; the one with the best MDL measure is accepted.
        - If no improvement, the algorithm stops with the current model.
    - Note that there are no edges are deleted. The search is always from less to more complex networks
3. Edges are then checked individually for an improved MDL measure via edge reversal
4. Loop at Step 2

Box 3 MDL causal discovery algorithm

Even though the search algorithm exploits the knowledge from a domain expert as optimal as possible, a greedy search or brute-force search is still applied. It suggests that the growth of candidates for optimal structure can still be exponential. The worst case will be when knowledge from a domain expert, for whatever reason, is not available in which makes the whole search algorithm infeasible.

Concerning the proposed metric itself, the hypothesis message that is proposed has ignored two constraints that bind BN. These are:

- BN is an acyclic graph
- One variable in a BN cannot have two identical parents.

Furthermore, since then number of parents for each variable is not explicitly included in the hypothesis message, it implies that the number of parents for each variable is known beforehand. These factors contribute towards a looser precision of the hypothesis message and, hence, a looser precision of the whole metric.

SUZUKI'S MDL CODE

Compared to Lam and Bacchus' MDL code, Suzuki's MDL code only differs in the hypothesis message (Suzuki, 1999):

$$I_S(h) = \left(\log\frac{K}{2}\right)\sum_{i=1}^{N}(s_i - 1)\prod_{j\in\pi(i)} s_j$$

Suzuki's MDL code attempts to improve the efficiency of the code by Bacchus & Lam by taking the size of the data set into account in hypothesis message. This will give a better representation of the parameters for each variable in the BN. However, on the other side the proposed hypothesis message entirely does away with the structural complexity of the BN. In the hypothesis message of Suzuki's MDL code, the parents of variables are ignored which also leads to a less efficient code.

2.3.1.5 MML CAUSAL DISCOVERY

The basic idea of MML is similar to MDL's which is to search for a trade-off between low structure complexity and the fit of training data to the structure. MDL and MML are also inspired by the work of Shannon on the Information Measure (See MDL causal discovery section for the equation of Information Measure). The only difference between MDL and MML lay in whether it is Bayesian or non-Bayesian. MDL is offered specifically as a non-Bayesian inference method, which avoids the probabilistic interpretation of its code, whereas MML is specifically offered as a Bayesian technique.

In this section, the method of structure learning by means of MML that is implemented in the program CaMML will be presented. The inventor of MML, Chris Wallace (Wallace & Boulton, 1968), is one of the developers of this CaMML program. From this point on, this method will be called CaMML method.

Analogous to MDL Causal Discovery, the CaMML method also proposes a metric that defines the information measure for BNs (This includes structure, parameter, and training data). Given that the number of variables in a structure is known, the CaMML method specifies the information measure for the structure ($I_{MML}h$) as follows (Korb, 2004):

$$I_{MML}(h) = \log N! + \frac{N(N-1)}{2} - \log M$$

where:

- N is the number of variables in the structure
- M is the number of linear extensions of the dag.

$I_{MML}(h)$ can be broken down into three terms. The first term, $\log N!$, specifies the message length needed to communicate the ordering of N variables. If the message is communicated in bits, then the log in the first term is a base-2 log. By specifying the ordering of the variables, the constraint of BN as an acyclic graph is guaranteed at once. Another constraint of BN as an directed graph is fulfilled by defining the edges in the second term, $\frac{N(N-1)}{2}$. This second term is derived from $\binom{N}{2}$ which calculates the number of possible edges in the structure. Because one DAG can have more than one ordering, all other orderings that belong to the same DAG, the so-called linear extensions, should be subtracted out the message length to keep the message as effective as possible. This subtraction is realized in the third term.

Furthermore, the information measure for the parameter $\left(I_{MML}(\theta|h)\right)$ and the training data $I_{MML}\left(x_j|h,\theta_j\right)$ are also defined:

$$I_{MML}(\theta|h) = \sum_{X_j} -\log \frac{f(\theta_j|h)}{\sqrt{F(\theta_j)}}$$

$$I_{MML}\left(x_j|h,\theta_j\right) = -\log P\left(x_j|h,\theta_j\right) = -\prod_{k=1}^{K} \frac{1}{\sigma_j\sqrt{2\pi}} e^{-\frac{\delta_{jk}^2}{2\sigma_j^2}}$$

Where

- $f(\theta_j|h)$ is the prior density function over the parameters for X_j given h
- $F(\theta_j)$ is the Fisher information for the parameters to X_j
- K is a number of samples in the training data
- θ_{jk} is the difference between the value in the training data and the prediction made by structure and parameters

- δ_{jk} is the difference between the sample in training data of X_j and its linear prediction
- σ_j is the variance of normal distributed likelihood $f(\theta_j|h)$

$I_{MML}(\theta|h)$ is derived from the message length for encoding linear parameters in (Chris & Freeman, 1987). The Fisher information in $I_{MML}(\theta|h)$ is a real number between 0 and infinity that controls the precision of the parameter estimation. Hence, it is desirable that Fisher information is made as large as possible to minimize $I_{MML}(\theta|h)$. The data code itself $I_{MML}(x_j|h,\theta_j)$ measures the fit to the data by means of likelihood function $P(x_j|h,\theta_j)$. In this case, the likelihood function is normally distributed.

Already having the metric, for each possible DAG can be measured whether it's an optimal structure according to the training data. However, the same problem arises; there are too many possible DAGs to measure. This is a reason to turn to stochastic search. The CaMML method proposes two stochastic searches to learn the structure of BN without iterating all possible DAGs. These are Genetic Algorithm Search and Metropolis Search.

GENETIC ALGORITHM SEARCH

Genetic Algorithm (GA) Search is a global heuristic search technique that is based on the analogy of evolution. The solutions in GA search are represented as a population and by means of manipulation techniques such as inheritance, mutation, selection, and crossover new solutions are reproduced. After lots of iterations, a group of solutions that are the most optimal will still be in the population. The most optimal solution is one of the solutions in this population.

In learning the structure for a BN using GA search, the CaMML method proposes that the variable ordering can be applied as hypothesis space and equal prior probability is assigned to all possible variable orderings (Neiland & Korb, 1999). This implies for BN as DAG whose variables are partially ordered that one DAG can have more than one linear extension. Hence, the more linear extensions a DAG has, the more probable its prior becomes.

The pseudo code of learning structure using GA search is stated in Box 4.

1. Choose *n* DAGs as initial population
2. Calculate the MML metric ($I_{MML}(h,e)$) of all DAGs in the population
3. Repeat until termination: (time limit or sufficient $I_{MML}(h,e)$ achieved)
    a. Select best-ranking DAGs to reproduce
    b. Generate new offsprings by
        i. Swapping a subgraph from one parent into the graph of the other parent
        ii. Mutation: Edge deletions, additions or reversals
    c. Calculate the MML metric ($I_{MML}(h,e)$) of the new offsprings
    d. Replace worst ranked part of population with new offsprings
4. Choose the DAG in the population with the lowest $I_{MML}(h,e)$ as the most optimal structure

Box 4 Pseudo code of learning Structure using Genetic Algorithm Search

As can be seen from the pseudo code, GA search does not iterate all possible DAGs which make the algorithm more feasible. The quality of the produced DAG in the end of the algorithm is, under the assumption that the $I_{MML}(h,e)$ decreases as more generations are created, can be controlled by terminating the algorithm when a certain level of $I_{MML}(h,e)$ is met. However, GA search is most likely to suffer from being trapped in local minima. This will cause GA search to come up with a sub-optimal structure instead of an optimal structure. To avoid this problem, a temperature parameter, resembling to the one in simulated annealing, can be introduced (Korb, 2004).

METROPOLIS SEARCH

To understand Metropolis Search as proposed in (Wallace & Korb, 1999), we will begin this section with introducing the Totally Ordered Model (TOM). A TOM is simply a DAG with one of its linear extension. As a DAG is partially ordered, there is more than one possible total ordering for each DAG. This collection of total orderings for one partially ordered DAG is called linear extensions. Wallace & Korb argue that to avoid the ignorance about the underlying causal story (the exact total ordering of DAG), all TOMs should be assigned equal prior probability. This means that the more ways a DAG can be realized, the higher prior probability it will get.

The Metropolis Search is a type of Markov Chain Monte Carlo (MCMC) search. Through the sampling process, Metropolis search tries to approximate the posterior probability distribution over the space of TOMs ($P(TOM_i|Training\ Data)$). Having the approximation of the posterior probability distribution, the most optimal TOM can be easily chosen as the most optimal TOM will have the highest posterior probability. Because in this Metropolis Search TOMs are applied instead of DAGs, the $I_{MML}(h)$ is changed by dropping the term that subtracts out the linear extensions:

$$I_{MML}(h) = \log N! + \frac{N(N-1)}{2}$$

The step-by-step details of Metropolis Search in order to learn BN structure (Korb, 2004) is portrayed in Box 5.

1. Select initial TOM randomly. Call this M.
2. Choose uniformly randomly between the following transformations of the TOM. Applying the transformation to M will produce M':
    a. Temporal order change: Swap the order of two neighbouring nodes. If there is an edge between them, it reverses direction.
    b. Skeletal change: Add/delete an edge between two randomly selected nodes.
    c. Double Skeletal change: Randomly choose 3 nodes. Add/delete edges between the first two nodes and the last (in the temporal order).
3. Accept M' as the new M if $\frac{P_{MML}(M')}{P_{MML}(M)} > U[0,1]$; otherwise retain M. where $P_{MML}(x) = e^{-I_{MML}(x)}$ and $U[0,1]$ is uniform random variate in the interval $[0,1]$.
4. Count the current visit to M.
5. loop at 2 (until an input number of samples has been completed)

Box 5 Pseudo code of learning structure using Metropolis Search

According to the principle of Monte Carlo sampling, the approximation of the posterior probability distribution becomes more precise when more samples are taken to build the distribution. For more details in sampling and selecting the structure, we refer to (Wallace & Korb, 1999) and (Korb, 2004).

2.3.1.6 BCGE METRIC

The metrics that are introduced above assume that the variables of the BN are all discrete variables. However, in some domain such as automated speech recognition, a BN will consist of both discrete and continuous variables. The easiest approach that we can do to solve this problem is to translate the continuous variables to discrete variables. By doing this, the metrics that are discussed in the previous sections can be applied to learn the structure of the BN. Yet, discretizing the continuous variables is not desirable since it will decrease the quality of the learned BN's inference. In (Geiger & Heckerman, 1994), a metric that also take the continuous

variables into account is introduced. The name of the metric, BcGe metric, stands for "Bayesian conditional Gaussian event equivalent". As the name already suggests, the metric assumes that the continuous variables in the BN is distributed according to multivariate normal distribution.

Before the metric is introduced, let's consider the denotation that will be used in this section. Let $x^{\rightarrow} = \Delta \cup \Gamma$, where all variables in Δ are discrete variables and all variables in Γ are continuous variables. Given $x^{\rightarrow}$, a *connected component* Γ_i is a subset of Γ in such way that every pair of variables in Γ_i are connected by undirected path that does not include a discrete variable. Furthermore, Δ is the collection of the parents of the variables in Γ.

BcGe Metric put constraints on the BN structure. These constraints are

- No continuous variable is the parent of a discrete variable
- The parents of each variable in Γ_i is Δ_i
- $\Gamma_i \cup \Delta_i$ has a conditional Gaussian distribution (A set X is said to have conditional Gaussian distribution if the distribution of the continuous variables in X, conditioned on each instance of the discrete variables in X, is a multivariate normal distribution)

Only BN structures that satisfies these constraints can be learned by means of this metric.

Similar to other metrics that are based on Bayesian approach, the metric calculates the posterior probability $P(D|B_S^e, \xi)$. Let's decompose the posterior probability based on the training data:

$$\begin{aligned} P(D|B_S^e, \xi) &= \prod_{l=1}^{m} P(C_l|C_1 \dots C_{l-1}, B_S^e, \xi) \\ &= \prod_{l=1}^{m} P\left(C_l^{\Delta}|C_1 \dots C_{l-1}, B_S^e, \xi\right) P\left(C_l^{\Gamma}|C_l^{\Delta}, C_1^{\Gamma} \dots C_{l-1}^{\Gamma}, B_S^e, \xi\right) \\ &= P(D^{\Delta}|B_S^e, \xi) \prod_{l=1}^{m} P\left(C_l^{\Gamma}|C_l^{\Delta}, C_1^{\Gamma} \dots C_{l-1}^{\Gamma}, B_S^e, \xi\right) \end{aligned}$$

where C_l^{Δ} and C_l^{Γ} is l^{th} observation in the training data that is restricted to Δ and Γ, respectively. The first term of the posterior probability, $p(D^{\Delta}|B_S^e, \xi)$, is simply the metric for measuring the BN structure with only discrete variables. Metric such as BDe metric can be applied to calculate $p(D^{\Delta}|B_S^e, \xi)$. The product in the rest of the term can be written as follows:

$$\prod_{l=1}^{m} P\left(C_l^{\Gamma}|C_l^{\Delta}, C_1^{\Gamma} \dots C_{l-1}^{\Gamma}, B_S^e, \xi\right) = \prod_{i=1}^{\gamma} \prod_{j=1}^{s_i} P\left(D^{\Gamma_{i,j}}|\Delta_i = j, B_S^e, \xi\right)$$

where $D^{\Gamma_{i,j}}$ is the training data restricted to variables Γ_i and where $\Delta_i = j$.

$P\left(D^{\Gamma_{i,j}}|\Delta_i = j, B_S^e, \xi\right)$ is the metric that handles the "mixed" part of the network. In similar manner as how BDe metric is derived, (Geiger & Heckerman, 1994) proposes a metric that calculates the posterior probability of the network that also contains continuous variables. The authors derive the metric by means of the following assumptions:

- Training data is a random sample from a multivariate normal distribution with unknown means $m^{\rightarrow}$ and unknown precision matrix W
- *Event equivalence*, Events $B_{S_1}^e$ and $B_{S_2}^e$ are equivalent if and only if the structures B_{S_1} and B_{S_2} are markov equivalent
- The training data is complete

- The prior distribution $P(m^{\rightarrow}, W|B_{SC}^{e}, \xi)$ is a normal-Wishart distribution given in page 178 of (DeGroot, 1970)
- *Parameter independence*, for every network B_S, $P(v^{\rightarrow}, B|B_S^e, \xi) = \prod_{i=1}^{n} P(v_i, b_i^{\rightarrow}|B_S^e, \xi)$, where n is the number of variables in the network
- *Parameter modularity*, if variable X_i has the same parents in two networks B_{S_1} and B_{S_2}, then $P(v_i, b_i^{\rightarrow}|B_{S_1}^e, \xi) = P(v_i, b_i^{\rightarrow}|B_{S_2}^e, \xi)$

As the first assumption has suggested, the metric only handles training data that is distributed by multivariate normal distribution. Hence, the authors have named this metric BGe Metric. The abbreviation BGe stands for "Bayesian Gaussian event equivalent". The BGe metric is calculated with the following equation (for detailed mathematical derivation, see (Geiger & Heckerman, 1994)):

$$P(D|B_S^e) = \prod_{i=1}^{n} \frac{P(D^{X_i,\Pi_i}|B_{SC}^e, \xi)}{P(D^{\Pi_i}|B_{SC}^e, \xi)}$$

where D^{X_i,Π_i} is the training data restricted only to variables X_i and Π_i, while D^{Π_i} is the training data restricted only to variables Π_i.

$P(D^{\Pi_i}|B_{SC}^e, \xi)$ and $P(D^{X_i,\Pi_i}|B_{SC}^e, \xi)$ refer to the following equation with the difference in training data:

$$\begin{aligned} P(D|B_{SC}^e, \xi) &= \prod_{l=1}^{m} P(C_l|C_1, \dots, C_{l-1}, B_{SC}^e, \xi) \\ &= 2\pi^{-\frac{nm}{2}} \left(\frac{v}{v+m}\right)^{\frac{n}{2}} \frac{c(n,\alpha)}{c(n,\alpha+m)} |T_0|^{\frac{\alpha}{2}} |T_m|^{\frac{-\alpha+m}{2}} \end{aligned}$$

where:

- v is the equivalent sample size for $m^{\rightarrow}$
- m are the number of samples in the training data
- $c(n, \alpha)$ is the multivariate gamma function
- α and T_0 are the parameters of the Wishart distribution that determines the distribution of the precision matrix W
- T_m can be calculated by the Theorem 1 in page 178 of (DeGroot, 1970).

Furthermore, if we assume that the prior probability of the whole network can be decomposed into:

- prior probability for discrete component of the network ($P(B_S^{e\Delta}|\xi)$)
- prior probability for continuous component of the network ($P(B_S^{e\Gamma}|\xi)$)
- prior probability for the component describing the edges between the discrete and continuous variables ($P(B_S^{e\Delta\Gamma}|\xi)$),

then these prior probabilities can also be incorporated into the BcGe metric:

$$P(D, B_S^e|\xi) = \{P(B_S^{e\Delta}|\xi) P(D^{\Delta}|B_S^{e\Delta}, \xi)\} P(B_S^{e\Delta\Gamma}|\xi) \left\{ P(B_S^{e\Gamma}|\xi) \prod_{i=1}^{\gamma} \prod_{j=1}^{s_i} P(D^{\Gamma_{i,j}}|\Delta_i = j, B_S^e, \xi) \right\}$$

2.3.1.7 PARTICLE SWARM OPTIMIZATION

The Particle Swarm Optimization (PSO) is a global heuristic search method whose concept is based on the social interaction of individuals in a swarm (Kennedy & Eberhart, 1995). These individuals who are usually referred to

as particles are spread out on the search space and by using the information from the neighbouring particles, each particle roams in the search space to find the area whose heuristic value is the most optimal.

1. Initialize the swarm, $P(t)$, of particles such that the position $x_i(t)$ of each particle $P_i \in P(t)$ is random within the hyperspace, with $t = 0$
2. Calculate the heuristic F of each particle by means of its position
3. Compare the performance of each particle to the performance in previously best position. If $F(x_i(t))$ is more optimal than $F(x_i^{pbest})$, then:
   - $x_i^{pbest} = x_i(t)$
4. Compare the performance of each particle to the performance in the global best position. If $F(x_i(t))$ is more optimal than $F(x_i^{gbest})$, then:
   - $x_i^{gbest} = x_i(t)$
5. Change the velocity vector for each particle:
   - $v_i(t) = v_i(t-1) + \rho_1 \left(x_i^{pbest} - x_i(t)\right) + \rho_2 \left(x_i^{gbest} - x_i(t)\right)$
   - ρ_1 and ρ_2 are positive random variables. ρ_1 controls the cognitive component, whereas ρ_2 controls the social component.
6. Move each particle to a new position:
   a. $x_i(t) = x_i(t-1) + v_i(t)$
   b. $t = t + 1$
7. Go to step 2, and repeat until convergence.

Box 6 Pseudocode of Particle Swarm Optimization

Each particle in the swarm has a position at timestep t ($x_i(t)$) in the search space and it takes record of the previously best position (x_i^{pbest}) and the previous velocity ($v_i(t-1)$). Besides the interaction with the neighbourhood, those parameters also play an important role in determining the velocity of the particle. The neighbourhood itself is defined by the labels of the particles and not by any spatial information such as the Euclidean distance or other topological distances. Furthermore, another important system parameter of PSO is the position of the global best performance ($gbest$). As the name suggests, it keeps record of the position whose performance is the best in the swarm. The basic pseudocode of PSO (Engelbrecht, 2007) is stated in Box 6.

However, this raises the question of how we can encode BN in order to incorporate it to PSO. In (Du, Zhang, & Wang, 2005), the author proposes that PSO can be applied in space of DAGs. Given that there are N variables, the DAG space is an $N(N-1)$ dimensional space. Each dimension represents an edge between two variables and has binary value, viz.: present or not present. Ergo, it implies that each position in the space is a DAG and it is represented by a binary vector. Having the DAG already encoded, the addition and subtraction operations between this binary vector should also be defined. The addition between two binary vector x_1 and x_2 can be done by applying an OR operation: $x_1 + x_2 = x_1 \vee x_2$. While subtraction is evaluated by using the following binary equation: $x_1 - x_2 = x_1 \wedge \neg x_2$. Since in this case, positive random variables ρ_1 and ρ_2 is not applicable anymore in calculating the next velocity, (Du, Zhang, & Wang, 2005) modifies the calculation of the new velocity. The proposed new velocity is calculated as follow:

$$v_i(t) = v_i(t) + \left(x_i^{pbest} - x_i(t)\right) + \left(x_i^{gbest} - x_i(t)\right)$$

Furthermore, to measure the performance of the position, metrics such as BDe metric, MML or MDL can be applied as heuristic measure. See previous sections for the details about the metrics.

2.3.2 LEARNING STRUCTURE IN DYNAMIC BAYESIAN NETWORK

Learning Structure in DBNs is not very different from learning structure in BNs, since a DBN can also be considered as a BN with variables that have a time index. Accordingly, learning DBN can be done by applying the learning techniques from BNs for variables from all time slices with the constraint that future dependencies are not allowed. However, this will result in a network that can only handle limited time slices which also implies that the dynamics of the DBN becomes limited. To make it easier, let's make assumptions about the temporal process that we would like to model. We assume that:

1. The process is Markovian. It means that when the process has n^{th} order Markov, then:
   - $P(X[t]|X[0],\dots,X[t-1]) = P(X[t]|X[t-n],\dots,X[t-1])$
2. The process is stationary. For example, if the process is first order Markov, then:
   - $P(X[t]|X[t-1])$ is independent of any timesteps t

Based on these assumptions, it is valid to decompose a DBN into two BNs: B_0 and $B_{\rightarrow}$. B_0, the initial network, is a BN that defines the beginning of the time-series process and $B_{\rightarrow}$, the transition network, is a BN that defines the transition of between the variables from one timestep to the next timestep(s).

In (Friedman, Murphy, & Russell, 1998), the authors attempt to propagate the decomposition of a DBN to the decomposition of the calculation of metrics. Let's consider that the metrics that are introduced in the previous section are measuring the posterior probability $P(D|G)$ where D is training samples and G is hypothesized DBN. This posterior probability can be calculated by involving parameters of the DBN:

$$P(D|G) = \int P(D|G,\Theta)P(\Theta|G)d\Theta$$

First, let's analyze the likelihood function $P(D|G,\Theta)$. By utilizing the conditional independence property of DBN and assuming that the samples are multinomial samples, the likelihood function can be decomposed into the following equation:

$$P(D|G,\Theta) = \prod_i \prod_{ji'} \prod_{ki'} \theta^{0}_{i,ji',ki'}{}^{N^{0}_{i,ji',ki'}} \prod_i \prod_{ji} \prod_{ki} \theta^{\rightarrow}_{i,ji,ki}{}^{N^{\rightarrow}_{i,ji,ki}}$$

Where:

- i is the index of the variable
- ji' is the index of the state of the i^{th} variable's parents in B_0
- ki' is the index of the state of i^{th} variable in B_0
- ji is the index of the state of the i^{th} variable's parents in $B_{\rightarrow}$
- ki is the index of the state of i^{th} variable in $B_{\rightarrow}$
- $N_{i,j,k}$ is the count in the training samples for data with value corresponded to index i, j and k.

It can be seen from the equation that the likelihood function can be decomposed to into two likelihood functions for the initial network and the transition network. The likelihood function of the whole DBN is simply the product of those two likelihood functions.

Having the possibility to decompose the likelihood function, the whole metric can be decomposed when the prior probability of the parameters can also be decomposed. So, let's analyze the prior probability $P(\Theta|G)$. Assume that the choice of the prior distribution for each variable's parameters are independent from each other, then the prior probability of the parameters can also be decomposed:

$$P(\Theta|G) = \prod_i \prod_{ji'} P\left(\theta^{0}_{i,ji',ki'}\right) \prod_i \prod_{ji} P\left(\theta^{\rightarrow}_{i,ji,ki}\right)$$

In the equation above, it is also clear that the prior probability of the parameters can also decomposed into the prior probabilities of the initial network and the transition network. The decompositions of those two terms conclude that:

1. If we want to find the maximum likelihood parameters, we can maximize within each family independently
2. The structure of initial network (B_0) and transition network ($B_{\rightarrow}$) can be learnt independently.

As the closing argument of this section, it is also possible that the structure of initial network is no different from the first time-slice of the transition network. This implies that we only need to learn the structure of transition network. However, it is encouraged to learn the initial network separately since the structure of initial timeslice (timestep 0) may have a quite different independence structure from other slices due to:

- Initialization process that may allow aberration
- Influence of other unobserved process prior to the initial timeslice.

2.3.3 COMBINING PRIOR KNOWLEDGE

Besides the training data that contains the observations from the system itself, prior knowledge from the experts is also an important source to help learning the structure of the BN or DBN. Unfortunately, experts are usually not very good in providing quantitative information. On the bright side, the experts usually can easily explain how the one variable influences the other variables. This means that the prior knowledge from the experts are more qualified as an improvement in learning the structure than learning the parameters of the BN or DBN. In this section, extensions to some of the metrics discussed in the previous section are introduced in order to embed the prior knowledge into the calculation of metric.

2.3.3.1 PRIOR DISTRIBUTION OF BSE METRIC

In the equation of the BSe metric, the term $P(B_s^h|\xi)$ represents the prior distribution of the structures. When there is no information about the distribution of the structures, this term becomes less significant to the BSe metric. Such a prior distribution is called an uninformative prior or a flat prior since such prior treats each structure as equally possible. In a case where BSe metric is used to compare one structure to another, the term $P(B_s^h|\xi)$ can be left out.

On the other side, the term $P(B_s^h|\xi)$ can be used as the medium to embed the prior knowledge from experts in BSe metric. In (Heckerman, Geiger, & Chickering, 1995), the authors propose a method to assess $P(B_s^h|\xi)$ based on prior knowledge from experts. In the proposed method, the knowledge of the experts is translated into a form of prior network. Prior network is a network with directed edges that is elicited from the experts. This prior network shows how the experts expect the structure of the BN or DBN for the system should be. The difference between this prior network and BN/DBN is the absence of the parameters in the prior network.

Before we introduce how $P(B_s^h|\xi)$ for the hypothesized structure B_s^h can be assessed, we will introduce some notations. Given that prior network (P) and B_s^h consist of the same variables and variable ordering in both networks are also the same, let δ_i be the symmetric difference between the parents of i^{th} node in P and in B_s^h:

$$\delta_i = \left|\left(\Pi_i(B_S^h) \cup \Pi_i(P)\right) \setminus \left(\Pi_i(B_S^h) \cap \Pi_i(P)\right)\right|$$

Given that there are N variables, let δ be the sum of symmetric difference of all variables:

$$\delta = \sum_{i=1}^{N} \delta_i$$

δ tells us how much the structure of B_S^h differs from the prior network. The idea of assessing $P(B_s^h|\xi)$ proposed by (Heckerman, Geiger, & Chickering, 1995) is penalizing B_s^h for every difference with prior network. This is realized by the following equation:

$$P(B_s^h|\xi) = ck^{\delta}$$

Where

- c is a normalization factor
- k is a constant factor between 0 and 1

When the BSe metric is calculated in order to compare between one structure with another, then normalization factor c can be ignored. The choice of k is dependent on how hard the difference with the prior network is penalized. By putting k to 1, the prior distribution of the structures becomes uninformative prior.

2.3.3.2 THEORY REFINEMENT BY WRAY BUNTINE

Compared to the method previously introduced, the theory refinement that is proposed by Wray Buntine (Buntine, 1991) goes further than eliciting the prior network from experts. The first information that is needed in this method is the variable order that is denoted by the symbol $\prec$. The second information is the prior network. The difference of the prior network in this case is the addition of the belief of the expert for each edge in the network. This belief is expressed in a form of probability and it indicates how strong experts believe that the relation between two variables connected by the corresponding edge actually exists. We denote this second information with ξ.

Given $\prec$ and ξ, the prior probability of the structure B_S^h will be assessed. In (Buntine, 1991), this prior, $P(B_S^h|\prec,\xi)$, can be simplified into the following equation:

$$P(B_S^h|\prec,\xi) = \prod_{x\in X} P(\Pi_x|\prec,\xi)$$

Where Π_x are the parents of x in B_S^h.

Furthermore, $P(\Pi_x)$ can be calculated by the following equation:

$$P(\Pi_x|\prec,\xi) = \left(\prod_{y\in\Pi_x} P(y\to x|\prec,\xi)\right)\left(\prod_{y\notin\Pi_x} 1 - P(y\to x|\prec,\xi)\right)$$

Where $P(y\to x|\prec,\xi)$ is the probability that edge $y\to x$ exist. This probability is incorporated in ξ.

The essence of theory refinement is updating the structure of the BN or DBN when new information such as observations is found. In (Buntine, 1991), the authors also suggests the mechanism of updating the probability of hypothesized structure B_S^h, $P(B_S^h|Samples,\prec,\xi)$, given that the observation data is available. According to standard rules of probability, this can be calculated as:

$$P(B_S^h|Samples,\prec,\xi) \propto P(B_S^h|\prec,\xi)\int_\theta P(\theta|B_S^h)P(Samples|B_S^h,\theta) = \prod_{x\in X} P(\Pi_x|Samples,\prec,\xi)$$

Where

$$P(\Pi_x|Samples,\prec,\xi) \propto P(\Pi_x|\prec,\xi)\prod_{j\in v(\Pi_x)} \frac{\beta_{mx}(n_{(x=1|j)}+\alpha_x,\dots,n_{(x=mx|j)}+\alpha_x)}{\beta_{mx}(\alpha_x,\dots,\alpha_x)}$$

- $v(\Pi_x)$ is the cartesian product of variables in Π_x
- β_{mx} is a beta function with mx parameters

Notice that the derivation of the $P(\Pi_x|Samples, \prec, \xi)$ is produced by assuming that the samples are taken from multinomial distribution.

2.4 SUMMARY

In order to learn the model structure of Dynamic Bayesian Network (DBN) for Automated Speech Recognition (ASR), we looked through the techniques of learning the parameters and learning the structure of DBN.

In this report, there are three parameter estimation methods introduced to learn the parameters of DBN. These are Maximum Likelihood (ML), Maximum a Posteriori (MAP) and Expectation Maximization (EM) Algorithm. The difference between these methods is stated in Table 3.

Table 3 Differences between Maximum Likelihood, Maximum a Posteriori and Expectation Maximization Algorithm

Estimation Method	Incorporate Prior Probability	Presence of Hidden Variables	Computation
Maximum Likelihood (ML)	✗	✗	Exact
Maximum a Posteriori (MAP)	✓	✗	Exact
Expectation Maximization (EM) Algorithm	✓	✓	Iterative

In learning parameters for DBNs, the EM algorithm is more suitable because, except when all variables that play roles in the speech recognition process are modeled, learning the parameters are always done in the presence of hidden variables. However, as an iterative algorithm, EM algorithm can get stuck in local maxima. To tackle this problem, the algorithm should be run multiple times and the parameters can be derived from the results of multiple runs.

When the presence of hidden variables in ASR is chosen to be ignored, then either a ML or MAP estimation method can be chosen. MAP estimation becomes more preferable than ML estimation if the knowledge of the prior probability is known. An interesting question in this case is to what extent the choice of ignoring the presence of hidden variable will affect the performance of ASR system.

In this report, it has also been discussed that learning the structure of a DBN can be done by extending the techniques in learning the structure of BN. In the introduced techniques of learning the structure of BN, we can recognize two distinct approaches, viz.:

1. *Constraint-based techniques*; the basic idea is constructing the BN by checking each possible edge.
2. *Metric-based techniques*; the basic idea of this technique is comparing the BNs by its metric.

For constraint-based techniques, two examples are given: the IC algorithm and the PC algorithm. Applying constraint-based technique to learn the DBN for ASR will require a lot of computation time since the model for ASR is known to have lots of variables. This means that the edges that are needed to be checked grow exponentially.

On the other side, metric-based techniques can be quite efficient if the choice of search algorithm is also efficient. A brute-force search algorithm (calculating the metric of all possible BN) will not suffice to examine the search space efficiently, so local search algorithm should be applied to find BN for ASR system. However, choosing local search algorithm also means that the resulting BN is not always optimal, while when all possible BNs are examined and compared with the metric then we are sure that the result is indeed the most optimal. In the techniques that are discussed in the previous chapter, the following algorithms are the possible choice to be applied as search algorithm:

- Genetic Algorithm
- Metropolis Search
- Particle Swarm Optimization

Besides those algorithms, there are also other algorithms such as Simulated Annealing or Ant Colony Optimization that are not mentioned but still can be applied. However, choosing between these algorithms are not quite simple, since the behavior of the algorithm due to its stochastic nature becomes less predictable. For these algorithms, we know that the optimality cannot be guaranteed, so the most ideal choice of algorithm will be the one that has the highest probability of reaching the most optimal result and takes less number of iteration before it converges. The only way to find out which algorithm is the most ideal for learning DBN of ASR system is by testing each of these algorithms and comparing the test results.

In this report, there are five metrics discussed. Besides those five metrics, there are also many other metrics proposed in other scientific papers or publications, but the basic idea of them are usually quite similar with the ones that are discussed here. In Table 4 the metrics introduced in this report are compared.

Table 4 Comparison of metrics (DAG = Directed Acyclic Graph, TOM = Totally Ordered Model)

Metric	Bayesian or Frequentist approach	Search Space	Variables	Possibility to combine prior knowledge
MDL	Frequentist	DAG	Discrete only	✖
MML	Bayesian	TOM	Discrete only	✖
K2/BD	Bayesian	DAG	Discrete only	✓
BDe	Bayesian	Equivalent DAG (Pattern)	Discrete only	✓
BcGe	Bayesian	Equivalent DAG (Pattern)	Mixed of discrete and continuous	✓

In ASR system, it always includes variables of acoustical variable that is usually a continuous variable while the other hidden variables are mostly discrete variables. This fact makes the choice for BcGe metric logical for ASR system. However, this does not make other metrics useless. By discretizing the continuous variables, the metric MDL, MML, K2/BD or BDe can be applied for learning the DBN of ASR system.

The drawback of metric-based algorithm is that it does not take the constraints of DBNs for ASR systems into account. As can be seen in *Dynamic Bayesian Network and its application in Automated Speech Recognition*

chapter, we can come up with three constraints that DBN should fulfill in order to be successfully applied to ASR system. These are:

1. Acyclic constraint
2. None of the variables should have a continuous variable as a parent
3. ASR structural constraint (Edges between position variables, observation variables and transition variables are fixed).

To tackle this problem, for metrics with Bayesian approach, the prior probability of DBN hypothesis that does not fulfill the constraints can be set to zero. Another more efficient solution is modifying the search algorithm in such way that it excludes the DBN that does not fulfill these constraints.

3 MODEL

Based on the background research from the previous chapter, we start to think about how these techniques can be applied to automatically find models for Automated Speech Recognition (ASR) system based on observational data. Based on prior knowledge about the Speech Recognition field, preferences for certain techniques will be motivated.

The chapter starts off with the analysis of aspects that are typical in learning Bayesian network for Speech Recognition. The second section will reiterate the two strategies of learning Bayesian Network: Constraint-based and Score-based. This section will also motivate why Score-based is more suitable for Speech Recognition. Lastly, the choices of the metrics and algorithms, which are the essential part of score-based strategy, will be clarified.

3.1 INHERENT PROPERTIES OF LEARNING BAYESIAN NETWORK FOR SPEECH RECOGNITION FIELDS

The literature study shows that the techniques that can be used to learn Bayesian network models using observation data already exist and have been successfully tested using observational data of popular Bayesian networks such as the ALARM network (Cooper & Herskovits, 1992) (Larrañaga, Kuijpers, Murga, & Yurramendi, 1996). However, there is no literature that tests these techniques on Speech Recognition field.

This raises the question of what makes learning Bayesian networks in Speech Recognition field different from other common fields. If we focus on the language model, which is a part of a Speech Recognition model, we can see that a method for learning Bayesian networks for Speech Recognition typically has to deal with the following properties:

1. The Bayesian network should be able to model temporal process. Such a Bayesian network is called Dynamic Bayesian network (Murphy, 2002).
2. Bayesian network models contain a mix of continuous and discrete variables.
3. At least one variable has a large amount of possible states. In a language model, one of the variables that is always present is the “word” variable whose states are all possible words that are considered in the model. For real conversational data, the amount of states can go from hundreds of states to tens of thousands of states.
4. The network contains a large amount of variables. If there is only a small amount of variables, then it will be easier to model it manually using expert knowledge. Thus, learning it from observational data will seem pointless.
5. Property 3 and property 4 cause the possible combinations of all states to increase exponentially as more variables are added. To be able to learn from the observational data, it is desirable that the size of the data is much larger than the amount of possible combinations. However, data with such size is not available as it is not feasible to compile such massive dataset. This means that learning Bayesian networks have to take data insufficiency into account.

To examine the second property further regarding to our experiments, in this thesis, we argue that the speech recognition model consists of 2 parts:

1. Language model: this models the choice of words based on variables that represent background and linguistic knowledge.
2. Acoustic model: this models the structure of phonemes by using information from variables that influences how words are pronounced.

It is possible that some variables occur in both parts, but nonetheless these two parts outlines the two processes in producing speech that is independent from each other. This assumption allows the simplification that these two parts (modeled as 2 separate Bayesian Networks) can be learnt separately and subsequently combined to create a Speech Recognition system. In this thesis, we will only focus on the language model part whose variables are all discrete variables. However, the acoustic model part can also be learnt in a similar manner as how the language model part is learnt. This means that the results of the experiments are also applicable for learning acoustic models.

In acoustic models, it is certain that the variables consist of a mix between discrete and continuous variables. One variable that always exist in an acoustic model is the variable acoustical features. Even currently, there is not much literature that discuss learning Bayesian network with mixed variables. One paper (Bøttcher, 2004) suggests that the Bayesian Network with mixed variables can be decomposed into two separate networks: a network with discrete variables and a network with continuous variables. These networks can then be learnt separately. If in the acoustic model the acoustical features variable is the only continuous variable, then by modeling the acoustical features as a mixture of Gaussians, a mixture weight variable can be introduced. This mixture weight variable is discrete and is the one that can be influenced by other variables. This omits direct linking between the variable acoustical features with other discrete variables. The mixture weight variable is then linked directly to the variable acoustical features and the parameters of the variable acoustical features can be learnt separately. Such a construction can be seen in Figure 16.

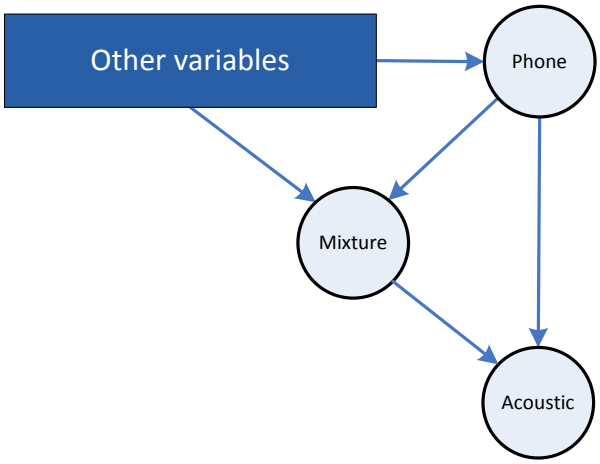


Figure 16 Construction of the acoustical model with a mixture weight variable

As mentioned in section 2.3.2, learning the model of a Dynamic Bayesian Network is not much different from learning the model of a Bayesian Network. (Friedman, Murphy, & Russell, 1998) suggested that Dynamic Bayesian Networks can be decomposed into an initial network and a transition network and that each network can be learnt independently. In the speech recognition field, the temporal process is represented by the order of words in a sentence for the language model part and the chain of phonemes for the acoustic model part. Other variables besides words and phonemes are most likely to be constant in one sentence. Given this prior knowledge, creating dummy words or phonemes just to mark the beginning and the end of a sentence or a chain of phoneme will eliminate the necessity of learning the initial network. However, this is done at cost of ignoring the possibility that the initialization of temporal process may deviate from the stationary part of the temporal process. This approach also does not cope with the possibility that there is an influence from the different processes prior to the initialization.

In this thesis, padding the sentence with dummy words is done to simplify the experiments. This means that in this thesis learning the model of a DBN can be translated into learning only the transition network. By adding these dummy words, the learnt Conditional Probability Table will show what's the most probable first word will be because it is obvious that apart from other words in a sentence the first word is only preceded by a chain of

dummy words. In this way, the initial network is embedded in the transition network itself. Regarding the influence from the different processes prior to the corresponding initialization, a DBN can be built that is modeling conversations instead of sentences to make the language model part more precise. However, this thesis will focus on building a DBN that models a sentence.

3.2 STRATEGIES IN LEARNING BAYESIAN NETWORK

As discussed in Chapter 2, there are 2 strategies of learning Bayesian Network when the observational data is given:

1. Constraint-based
   A Bayesian network is learnt by building the network based on testing the independence between two variables. Simple constraints are then applied to determine which pair of variables should be tested in each iteration. The concept of constraint-based Bayesian network learning is defined in the IC algorithm (Verma & Pearl, 1990) and implemented in the PC algorithm (Spirtes, Glymour, & Scheines, 2001).
2. Score-based
   Learning a Bayesian network is realized by firstly defining a quantitative measure to compare one Bayesian network to another. This quantitative measure is called score or metric. When the score is defined, the best Bayesian network can be chosen amongst all possible Bayesian networks. Some examples of the scores are the MDL metric, the BSe metric and the MML metric.

Considering the nature of learning Bayesian networks for speech recognition, the score-based strategy is more suitable for this task. From the aforementioned properties, it is clear that the network will contain a large amount of variables. This will cause the amount of possible networks to increase exponentially. The advantage of using a score-based strategy is that local search algorithm can be incorporated as a procedure to search for the network with the highest score. Iterating through all possible networks is therefore not necessary.

3.3 METRICS AND ALGORITHMS

Choosing a score-based strategy also implies that learning the Bayesian Network can be broken down into two aspects:

- The choice of metrics
- The choice of algorithm

3.3.1 THE CHOICE OF METRICS

To choose the metrics, we should weigh in what kind of variables the language model has. Let's see some examples of variables that can be part of the language model:

- Variable word; this variable contains all words in the vocabulary
- Variable conversation topic; conversation topic can influence which words are used
- Variable gender; gender of the speaker can influence the choice of words
- Variable age; there is a difference between the language of old generation and young generation

Because the language model has a lot to do with modeling word choices, it is unlikely that the model will contain continuous variables. As the examples show, all variables can be represented as discrete variables.

The language model mimics how human makes his or her choices of words. When human makes this choice, it doesn't depend on the quantitative influences, but it depends on qualitative influences. For example, the choice of words of a speaker who is 20 years old would not differ a lot from someone who 21 years old. However, the difference can be found when comparing the choice of words of a teenager and an elderly. We can relate this with the variables in the language model. It is difficult (or probably impossible) to find variables in language model that must be represented as continuous variables.

Because of the aforementioned reason, it is plausible to assume that the variables in language model are limited to discrete variables. From the literatures that discuss learning Bayesian Networks, there are 3 metrics that can handle discrete variables. These are MDL (Rissanen, 1978), MML (Wallace & Boulton, 1968), and BSe (Cooper & Herskovits, 1992). However, MML is not suitable because it needs the states in the variables to have an order. The equation of the MML metric involves the calculation of mean and variance. We already know words do not have such an order that makes the calculation of mean and variance possible.

As a language model that later will be a part of ASR system, it is important that the resulting language model also has the best performance in predicting the words in a sentence. In the language model literature, there is a way to measure the performance of the language model in predicting the words in a sentence. The measurement is known as Perplexity (Wiggers, 2008). In this thesis, we propose that Perplexity is added to a list of metrics for learning the language model. Perplexity measures the quality of the language model by calculating the probability that the language model predicts the test sentences right. The mathematical equation of this metric can be seen in the following equation (Wiggers, 2008):

$$P(v_{1,t}) = \prod_{n=1}^{N} \prod_{i=1}^{t} P(x_i|\Pi_i)_n$$

where

- x_i is the corresponding variable in the Bayesian network
- Π_i is the collection of x_i's parents in the structure
- N is the amount of row in the dataset
- t is the amount of variable in the Bayesian network.

In order to emphasize that in the Language Model, the predictability of words is more important than other variables, we propose a modified version of Perplexity. This version of Perplexity puts more weight on dependencies involving word variables and less weight on dependencies not involving word variables. From this point on, we refer this version of Perplexity as Modified Perplexity. This modified the equation for Perplexity into the following equation:

$$P(v_{1,t}) = \prod_{n=1}^{N} \prod_{i=1}^{t} c_{x_i} P(x_i|\Pi_i)_n$$

where c_{x_i} is a constant between zero and one for variable x_i.

In the experiments, the effectiveness of each one of these four metrics will be analyzed. A good metric will be the one that gives the best score to the original model or a model that is close to the original model.

3.3.2 THE CHOICE OF ALGORITHM

The second aspect is the choice of the algorithm. As we have mentioned before, a score-based strategy is very useful because any local search algorithm can be adopted in this strategy to find the best Bayesian Network. There are many forms and version of local search algorithms and so the choice of algorithm from various local search algorithms may seem arbitrary. In this thesis, we have chosen to implement a Genetic Algorithm and Particle Swarm Optimization. The Genetic Algorithm is chosen because this algorithm is considered to be a general solution for any search or optimization problem. This may not be the best choice, but in most of the cases it may be the second best choice, with the advantage that it requires little to no domain knowledge or heuristics. Much literature discussing learning Bayesian Network also suggests Genetic Algorithm as the choice of algorithm (Larrañaga, Kuijpers, Murga, & Yurramendi, 1996) (Wallace & Korb, 1999). The second choice, Particle Swarm Optimization, is picked because optimization literature also suggests that Particle Swarm Optimization outperforms Genetic Algorithm on many tasks (Wang & Yang, 2009).

The experiments will compare the efficiency of these two algorithms. An algorithm is considered better than the other when the algorithm can find the most optimal Bayesian network in less iterations.

3.4 APPLYING THE RESULT OF MODEL LEARNING TO AUTOMATED SPEECH RECOGNITION SYSTEM

Having the suitable algorithm, suitable metric and the most optimal setting for model learning, the most optimal DBN according to the observational data can be found. The question now is: can we use this result directly as a model for an Automated Speech Recognition (ASR) system? There are few trivial steps we have to do before the result can be applied as an ASR model.

As we have mentioned in section 3.1, a speech recognition model consists of two parts: a language model part and an acoustic model part. If both parts are successfully obtained by using model learning, one of the steps to convert these results to an ASR model is combining both models. To see how these two models can be combined, firstly let's see how the ASR model looks when all we got is only word variables as the language model part and both phone (phone is a phonetic sound) and acoustic features variables as acoustic model part. This model is taken from (Bilmes & Bartels, 2005) and is depicted in Figure 17.

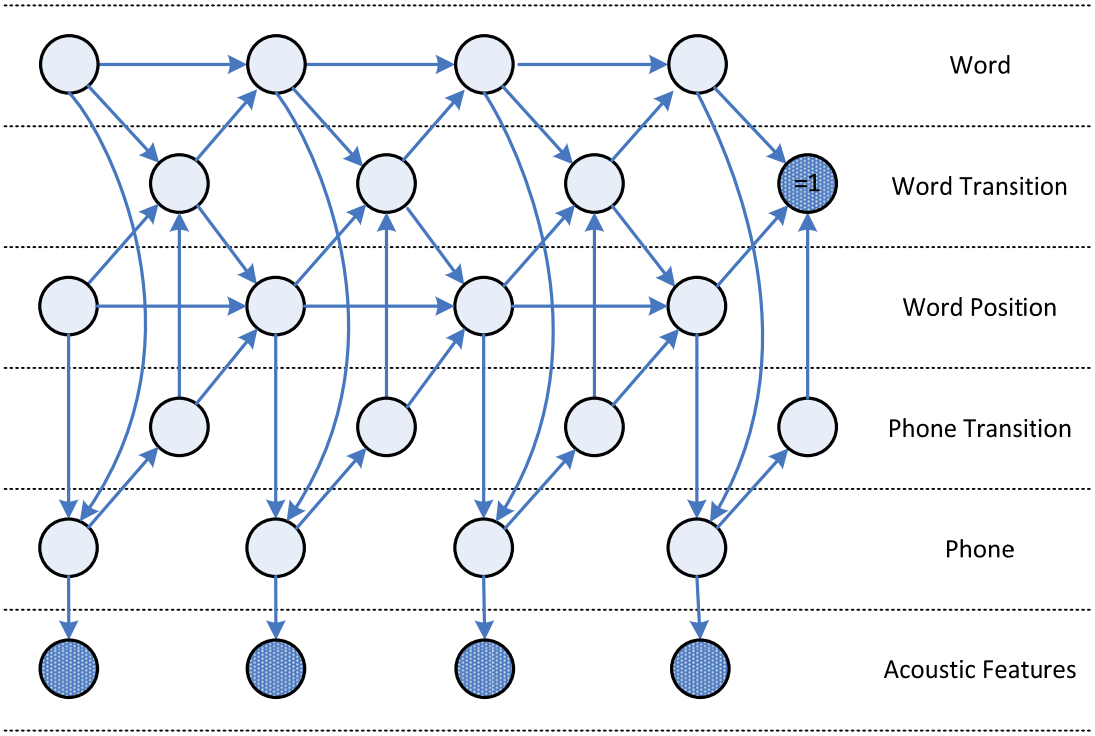


Figure 17 Model for ASR system that consists of the language model and the acoustic model

When the language model part contains not only word variable, the language model part can be put in this ASR model by aligning the word variable in the ASR model with the word variable in the language model part. The other variables in the language model will then be put around the word variable and are not linked with other variables in the ASR model.

For the acoustic model part, a similar procedure is applied. Phone and acoustic feature variables from the model learning's result are set in the places that are reserved for those variables. All other variables will then be placed around it. For a better visualization of the combined models, see Figure 18.

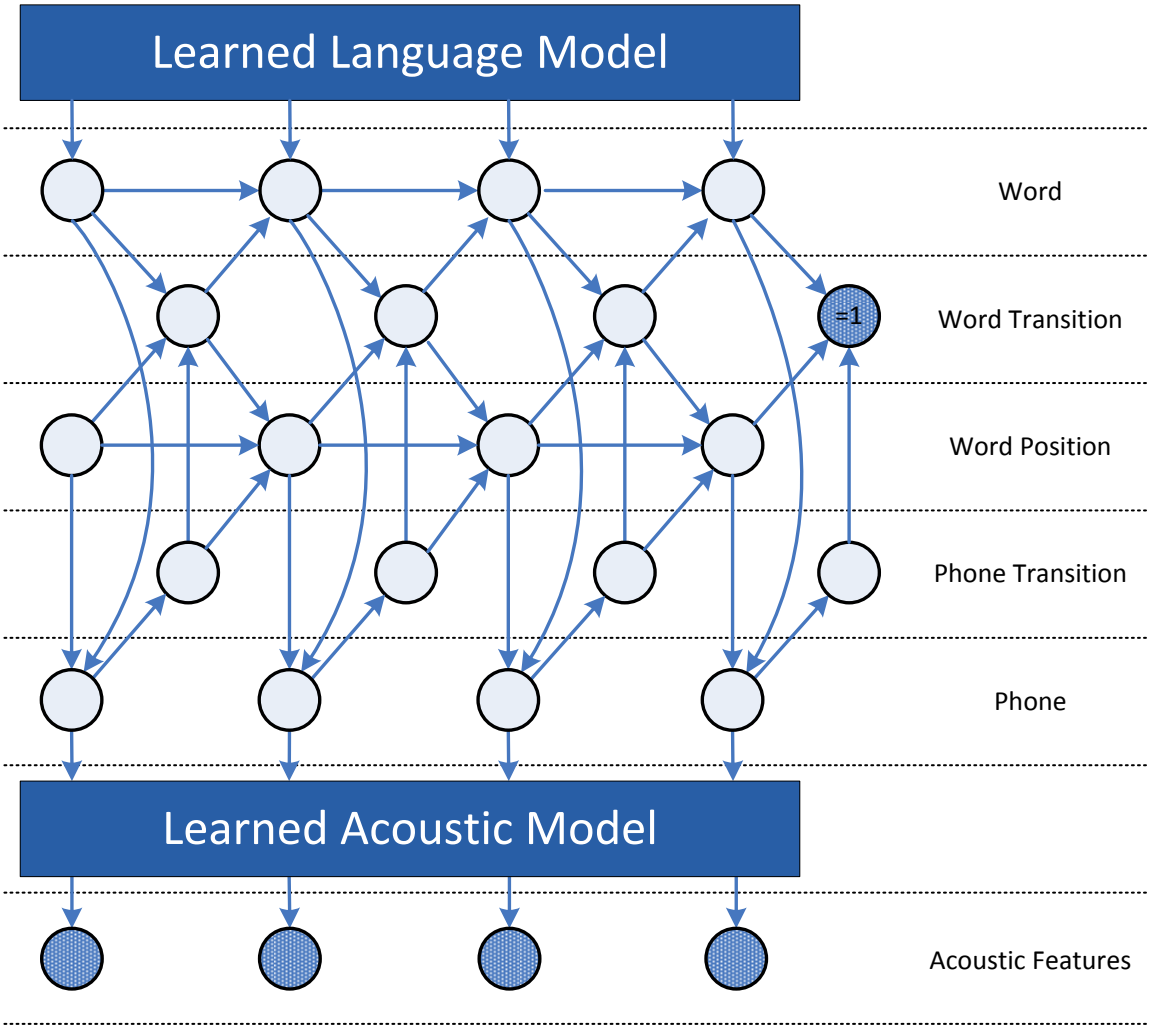


Figure 18 How the learned language model and the learned acoustic model are combined

As reader may have noticed, aside from variables from language model and acoustic model there are other additional variables in the ASR model. These variables are:

- Word transition
- Word position
- Phone transition

The variable word position keeps track of the ordinal position of the phone in a word. The states of this variable are the ordinal form of the number, i.e., FIRST, SECOND, THIRD, up to N^{th}, where N is the highest number of phonemes in a word. On the other hand, word transition and phone transition regulate the transition between respectively words and phones. Word transition tells the variable word when it's time to go to the next word in the sentence, while phone transition tells the variable word position to go to the next phone of the processed word. Ergo, the word transition and phone transition variables are binary variables whose states are TRANSITION and NO TRANSITION.

The conditional probability table of the word transition variable is a deterministic one. This means that the probabilities in the table are filled only with the number 0 and 1 or in other word there is no uncertainty in the choice of the state in this variable. For the word transition variables, the conditional probability table will portray the following rules:

- The variable will go to the state TRANSITION when the word position variable's state refer to the position of the last phoneme in the word and the phone transition variable's state is TRANSITION
- Otherwise, the variable will go to the state NO TRANSITION.

Similar to the word transition variable, the word position variable's conditional probability table is also deterministic. The conditional probability table of the word position variable is designed using the following rules:

- The state of the variable is incremented from the state of the variable in the previous time slice (e.g. state FIRST will be incremented to the state SECOND), when the word transition variable and the phone transition variable have TRANSITION as their state
- Otherwise, the state of the variable does not change in respect to the state of the variable in the previous time slice.

Meanwhile, the conditional probability table of the phone transition variable is not deterministic. The conditional probability table of this variable should mimic the distribution of phoneme duration in a word (this is similar to the state transition probability in conventional HMM-based ASR systems). The probabilities can be learnt separately from the annotated audio signals dataset. The probabilities in the table are non-zero and it tells how probable the transition to the next phone is when the phone variable is at a certain state. To avoid learning the conditional probability table separately, this phone transition variable can be added in the variables for the learned acoustic model.

Having the additional variables and its conditional probability tables, the steps in applying both the learned language model and the learned acoustic model to ASR system are complete. However, there is still one point of discussion. Because the variables in both the language model and the acoustic model are not necessarily exclusive of each other, it's possible that there are variables that are parents or children of variables from both models. This is not problematic since it is assumed that the process of choosing the words and process of producing the sound itself are independent of each other. To see how the conditional probability of both models can be combined, the combined conditional probability can be written as follows:

$$\begin{aligned} P\left(X \middle| Y_{language\ model}, Y_{phoneme\ model}\right) &= \frac{P\left(X, Y_{language\ model}, Y_{phoneme\ model}\right)}{P\left(Y_{language\ model}, Y_{phoneme\ model}\right)} \\ &= \frac{P\left(X, Y_{language\ model}\right) P\left(X, Y_{phoneme\ model}\right)}{P\left(Y_{language\ model}, Y_{phoneme\ model}\right)} \\ &= P(X|Y_{language\ model}) P(X|Y_{phoneme\ model}) \end{aligned}$$

This means that the combined conditional probability table can be derived by multiplying the conditional probability table of both models.

4 IMPLEMENTATION

The design of the experiments gives the theoretical concept of how the experiments will go. In this chapter, we will work out this theoretical concept to a more feasible and workable implementation.

In the first section, the issues around the implementation of the metrics will be discussed. Subsequently, the second section discusses the implementation of the algorithms. Since the implementation of the metrics requires frequent querying in a large dataset, we are going to address this problem in the third section and give solutions to make querying faster. Dealing with Bayesian Networks also means dealing with directed acyclic graphs. This is the reason to dedicate the last section to discuss how to make sure that acyclicity is preserved in the algorithm.

4.1 METRIC

From the last chapter, there are 4 metrics that are implemented for the experiments. Each metric will be discussed thoroughly in the following subsections.

4.1.1 MDL METRIC

As mentioned in the literature, there is more than one form of the MDL Metric. One of the most common MDL metric forms is the one proposed by Wai Lam and Fahiem Bachus (Lam & Bacchus, 1994). The hypothesis message is formulated as follows:

$$I_{LB}(h) = \sum_{i=1}^{N} \left(k_i \log N + d(s_i - 1) \prod_{j \in \pi(i)} s_j \right)$$

Where

- N is the number of variables in the structure
- k_i is the number of parents of the i-th variable
- d is word size in bits
- s_i is number of states of ith variable.

To calculate the hypothesis message given a Bayesian Network h is straightforward because every parameter in the formula can be derived from the structure in Bayesian Network h itself. A parameter that can differ from one computer system to other is parameter d. d represents how big each cell in a Conditional Probability Table is stored in memory. In our implementation, each cell in a Conditional Probability Table is stored as `double` whose size is 4 bytes.

To complete the MDL metric as a whole, Wai Lam and Fahiem Bachus construct the data message as follows:

$$I_{LB}(e|h) = K \sum_{i=1}^{N} H(X_i) - \sum_{i=1}^{N} H\big(X_i, \pi(i)\big)$$

where:

- K is the number of samples in the training data
- $H(X)$ is the entropy of variable X

$$H(X) = -\sum_{x} P(X = x) \log P(X = x)$$

- $H(X_i, \pi(i))$ is the mutual information between X_i and its parent set

$$H(X_i, \pi(i)) = H(X_i) - H(X_i|\pi(i)) = \sum_{x_i, \phi(i)} P(x_i, \phi(i)) \log \frac{P(x_i, \phi(i))}{P(x_i)P(\phi(i))}$$

  Where $\phi(i)$ ranges over the distinct instantiations of the parent set $\pi(i)$

The implementation of the data message is also straightforward. The probabilities such as $P(x_i, \phi(i))$, $P(x_i)$ and $P(\phi(i))$ can be calculated by doing a query in the dataset. However, querying the dataset is an expensive operation in terms of computing power. In the third section, we will suggest options to cope with this problem.

4.1.2 BSE METRIC

The BSe metric (Cooper & Herskovits, 1992) is formulated in the following equation:

$$P_{CH}(h_i, e) = P(h_i) \prod_{k=1}^{N} \prod_{j=1}^{|\Phi_k|} \frac{(S_k - 1)!}{(S_{kj} + S_k - 1)!} \prod_{l=1}^{S_k} \alpha_{kjl}!$$

where:

1. N is the number of variables in the structure
2. $|\Phi_k|$ is the number of possible states of $\pi(X_k)$ (The parents of variable X_k)
3. S_k is the number of possible states of variable X_k
4. α_{kjl} is the number of samples in the training data where X_k is assigned to its l^{th} state and $\pi(X_k)$ is assigned to its j^{th} state.
5. S_{kj} is the number of samples in the training data where $\pi(X_k)$ is assigned to its j^{th} value.

In the last chapter, we have recognized that in Speech Recognition, it is inevitable that we have to deal with a large dataset. Given the equation of the BSe metric above, there are 2 issues that should be handled to implement this metric. These are:

1. It is inevitable that S_k, S_{kj} or α_{kjl} is a very big number and calculating the factorial of it will induce an overflow.
2. It is also computationally not feasible to calculate the exact factorial result of S_k, S_{kj} or α_{kjl} when those numbers are big.

The equation of the BSe Metric makes the first issue more problematic since according to the equation the metric is a product of factorials. To cope with this issue, instead of calculating the original BSe metric, a logarithm of the original equation is calculated. This results in the following equation:

$$\log P_{CH}(h_i, e) = \log P(h_i) \sum_{k=1}^{N} \sum_{j=1}^{|\Phi_k|} \frac{\log(S_k - 1)!}{\log(S_{kj} + S_k - 1)!} \sum_{l=1}^{S_k} \log \alpha_{kjl}!$$

It should be avoided to calculate the factorial of a number more than 10, because the result will be exponentially large and for some computer systems the result will cause an overflow. To give an impression of how the result grows exponentially, the factorial of 10 is approximately 3.6 million while the factorial of 15 already results in approximately 1.3 trillion. This means that when calculating the of the logarithm factorial, the logarithm should be calculated first before the numbers are multiplied to calculate the factorial.

Concerning the second issue, calculating the exact factorial of a very big number is a problem that numerical mathematicians have researched for years. There are many algorithms designed to calculate the factorial efficiently. In (Ugur & Thompson, 2006), one of those algorithms is introduced. The pseudocode of this algorithm can be seen in Box 7 and Box 8.

```
procedure initializeDifferenceTable(p:integer, X:array of integers, size p+1,
the first index is 0)
{computes the initial difference table using p-sized partitions, 1 ≤ p ≤ n}

begin
        products: array of integers, size p+1, the first index is 0
        q,r,s,t: integers

        {initialization}
        q = p+1
        r = q*p
        s = 1

        {Section 1: compute initial p+1 products of size p}
        for(i = 0; i < q; i = i+1)
        begin
                t = 1
                for(j = 0; j < p; j = j+1)
                begin
                        t = t*(s+j)
                end

                products[i] = t
                s = s+p
        end

        {Section 2: compute first elements of each of p differences}
        X[0] = products[0]
        for(i = 0; i < p; i = i+1)
        begin
                q = q-1
                for(j = 0; j < q; j = j+1)
                begin
                        products[j] = products[j+1] - products[j]
                end

                X[i+1] = products[0]
        end
end
```

Box 7 Subroutine of factorial calculation

```
function factorial_p(n:integer, p:integer)
{computes n! using p-sized partitions, 1 ≤ p ≤ n}
    begin
        X: array of integers, size p + 1, the first index is 0
        iterates, fact: integers

        {Section 1: set the initial elements of X which stores the difference table}
        initializeDifferenceTable(p,X)

        {Section 2: obtain elements of difference table successively and multiply elements of the
        sequence}
        iterates = n − p
        fact = X[0]
        while(iterates ≥ p)
        begin
            {create the first order difference}
            for(i = 0; i < p; i = i + 1)
                X[i] = X[i] + X[i + 1]
            fact = fact * X[0]
            iterates = iterates – p
        end

        {Section 3: multiply remaining numbers to complete the factorial}
        for(i = 0; i < iterates; i = i + 1)
        begin
            fact = fact * n
            n = n − 1
        end

        return fact; {final factorial value}

    end
```

Box 8 Algorithm to calculate the factorial using p-sized partitions

Box 8 shows how the factorial is calculated, whilst Box 7 shows the subroutine in the calculation of factorial. The idea of the algorithm is increasing the calculation time of factorial by decreasing the amount of multiplications done in the calculation and replacing them with additions. In the pseudocode, the function factorial takes two parameters: n and p. It means that it computes $n!$ using p-sized partitions. p itself is a parameter that controls how much multiplications are done in the calculation. Given n and p, the number of multiplication follows the next equation:

$$T(n,p) = \left\lfloor \frac{n}{p} \right\rfloor + p^2 + n \bmod p$$

For the purpose of model learning for the language model, it is sufficient to pick p between 15 and 20. For the details of how p affects the number of multiplications, we refer to (Ugur & Thompson, 2006).

However, the bigger the number, the computational cost also grows higher because there are more numbers that must be multiplied and added. To reduce the computational cost, instead of calculating the exact result of

the factorial, an approximation can be made. The approximation can be done using Stirling's Approximation as follows:

$$n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$$

4.1.3 PERPLEXITY METRIC

To calculate the perplexity metric, the dataset should be divided into two separate dataset: a test dataset and a train dataset. It is plausible to presume that the size of the train dataset should be much larger than the test dataset. In our experiments, the size of test dataset is pinned at 30% of the size of train dataset.

The Perplexity metric is formulated in the following equation (Wiggers, 2008):

$$PP(v_{1,t}) = 2^{LP(v_{1,t})}$$

$$LP(v_{1,t}) = -\frac{1}{t}\log P(v_{1,t})$$

Where $v_{1,t}$ denotes the whole test dataset as a chain of data and calculating $P(v_{1,t})$ can be performed by applying Bayes' theorem:

$$P(v_{1,t}) = P(v_1)P(v_2|v_1)P(v_3|v_1v_2)\dots P(v_n|v_{1,t-1})$$

However, having a structure in a Bayesian network it simplifies the equation to:

$$P(v_{1,t}) = \prod_{n=1}^{N}\prod_{i=1}^{t} P(x_i|\Pi_i)_n$$

Where

- x_i is the corresponding variable in the Bayesian network
- Π_i is the collection of x_i's parents in the structure
- N is the amount of row in the dataset
- t is the amount of variable in the Bayesian network.

Every probability $P(v_i|\Pi_i)$ can be derived from the train dataset.

Let a table represents the dataset with each column that represents a variable and each row is an instance of all variable (More details in the representation of dataset will be discussed in Section 4.3). Because $v_{1,t}$ denotes the whole test dataset, calculating $P(v_{1,t})$ using the aforementioned equation implies iterating each and every row in the dataset. This will cause a huge overload in computation as the test dataset grows. When the count statistics of test dataset are known, the overload in computation can be drastically decreased. Rearranging the equation of $P(v_{1,t})$, it results in the following equation

$$P(v_{1,t}) = \prod_{i=1}^{t}\prod_{\phi=1}^{\Phi} P(x_i|\Pi_i)_\phi^{\#(x_i,\Pi_i)_\phi}$$

Where

- Φ is the amount of possible combination of variable x_i's and Π_i's states

- $\#(x_i, \Pi_i)_\phi$ is the amount of rows in test dataset that has the ϕ-th state of variable x_i and Π_i

According to the Perplexity Metric that is represented in equation $PP(v_{1,t})$, the model that has the lowest score is the most optimal model.

4.1.4 MODIFIED PERPLEXITY

As the name suggests, the modified perplexity metric is derived from the perplexity metric. The idea of modified perplexity metric which differs from the perplexity metric is putting more emphasis on particular variables. In a model where different variables are not equally essential, this metric can be useful. This is also the case in speech recognition. In N-gram model frequently used in speech recognition, the most essential variable is the N^{th} word variable because this variable is what the speech recognition analyzes to construct the recognized sentence after the inference is done. Therefore it makes more sense if this variable is given more weight for its data-fit. Putting this idea in terms of perplexity can be done by inserting a weight for each variable resulting in a equation as follows:

$$P(v_{1,t}) = \prod_{i=1}^{t} c_{x_i} \prod_{\phi=1}^{\Phi} P(x_i|\Pi_i)_\phi^{\#(x_i,\Pi_i)_\phi}$$

Where c_{x_i} is the weight for variable x_i. To normalize the metric, $c_{x_1}, c_{x_2}, \dots, c_{x_t}$ should add up to one.

4.2 ALGORITHM

In the following subsections, the implementation details of the Genetic Algorithm and Particle Swarm Optimization will be described. In every subsection, it will begin with answering how a Bayesian Network is encoded in the algorithms. Then we will continue with explaining the details of the algorithms.

4.2.1 GENETIC ALGORITHM

Genetic Algorithms use the metaphor of evolution in a group of organism. We refer to this group of organisms as the population. In the realm of learning Bayesian Network, an organism represents a Bayesian Network. The question is how a Bayesian Network is encoded as an organism so that Genetic Algorithm operators such as crossover and mutation can be applied to the Bayesian Network.

The encoding of a Bayesian Network must be able to put the directionality and acyclicity of the graph into the organism. However, putting acyclicity into the organism is complex because there are no straightforward rules in recognizing cycles. To tackle this problem, in these experiments recognizing and eliminating cycles is done afterwards as a compulsory step after a new organism is introduced. How recognizing and eliminating cycles is done is explained in the last section of this chapter.

In the Genetic Algorithm, the Bayesian network is represented as an array of symbols. The array represents all possible pairs of variables in the Bayesian Network. A variable can however not be paired with itself. Given that there are N variables in the Bayesian Network, the array will then be $(\sum_{i=1}^{N-1} i)$ symbols long. Each symbol represents the existence of dependence between two variables and it also shows the direction of that dependence. That means that there are only 3 possible symbols and those are $+$, $-$ and 0. For a pair of variables A and B, these symbols have the following meaning:

- $+$: there is a dependence between variable A and B and A causes B

- – : there is a dependence between variable A and B and B causes A
- 0 : there is no dependence between variable A and B.

Figure 19 shows how the symbols in the array ordered by depicting the transformation from the Bayesian network into the array of symbols.

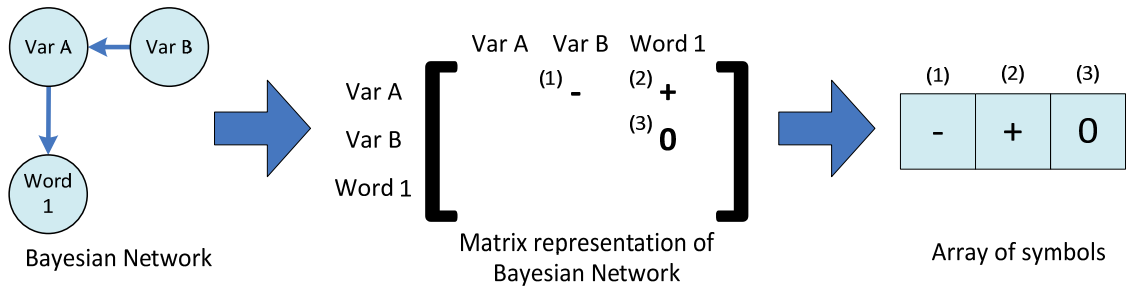


Figure 19 Encoding Bayesian Network into an array of symbols

Having the aforementioned encoding, the outline of the Genetic Algorithm can be revealed at this point. The procedure of the Genetic Algorithm can be seen in Box 9.

1. Initialize the population. Create N random organisms to fill the population.
2. If stop criterium is met, go to step 8
3. CROSSOVER
    a. Choose 2 organisms from the population as crossover candidates
    b. Perform crossover with these two organisms
    c. Make a copy for each of the offspring
    d. Insert offsprings in the population
    e. For every copy of offspring made in step 3c, draw a random decimal number between 0 and 1. Denote this decimal number as r.
        i. If $r < mutation\ rate$, then mutate the copy of offspring and insert the mutated version in the population
        ii. Else eliminate the copy of offspring
4. MUTATION
    a. Choose 2 organisms from the population as mutation candidates.
    b. For each of the organism, draw a random decimal number between 0 and 1. Denote this decimal number as r.
        i. If $r < mutation\ rate$, then mutate this organism and insert the mutated version in the population
5. BEST MUTATION
    a. Choose the organism with the highest score as mutation candidate
    b. draw a random decimal number between 0 and 1. Denote this decimal number as r_1.
        i. If $r_1 < mutation\ rate$, then mutate this organism and insert the mutated version in the population
        ii. Draw another random decimal number between 0 and 1. Denote this decimal number as r_2. If $r_2 < best\ mutation\ rate$, then go back to step 5b.
6. Eliminate identical organisms and organisms with lowest scores
7. Go back to step 2
8. Give the organism in the population with the highest score as the most optimal solution.

Box 9 How the Genetic Algorithm is implemented

The algorithm stops when the amount of iterations reaches a maximum or when for a certain period the highest score in the population does not change.

In the pseudocode, operations such as cross over, mutation, best mutation and selection play a great role in exploring possible Bayesian Networks. It's therefore necessary to get into the details of these operations.

4.2.1.1 CROSS OVER

Cross over begins with choosing two candidates for this operation. From the population, two organisms are chosen randomly. However, it is preferable that the cross over is done with candidates that have high score because by doing this, it is expected that the offspring from this cross over will have higher score than the original two candidates. In order to make this possible, the probability of each organism in population to be chosen is derived from its score rank in the population. This probability is calculated by the following equation:

$$p(organism_i) = \frac{\sqrt{rank_i + 1} - \sqrt{rank_i}}{\sqrt{\#population}}$$

where:

- $rank_i$ is the rank of organism i in the population
- $\#population$ is the amount of organisms in the population

The dividend in the equation $(\sqrt{rank_i + 1} - \sqrt{rank_i})$ causes the probability of the high-ranked organism in the population to be higher than the ones that are lower-ranked. This can be seen in Figure 20 where $(\sqrt{rank_i + 1} - \sqrt{rank_i})$ is calculated for rank 1 up to 20.

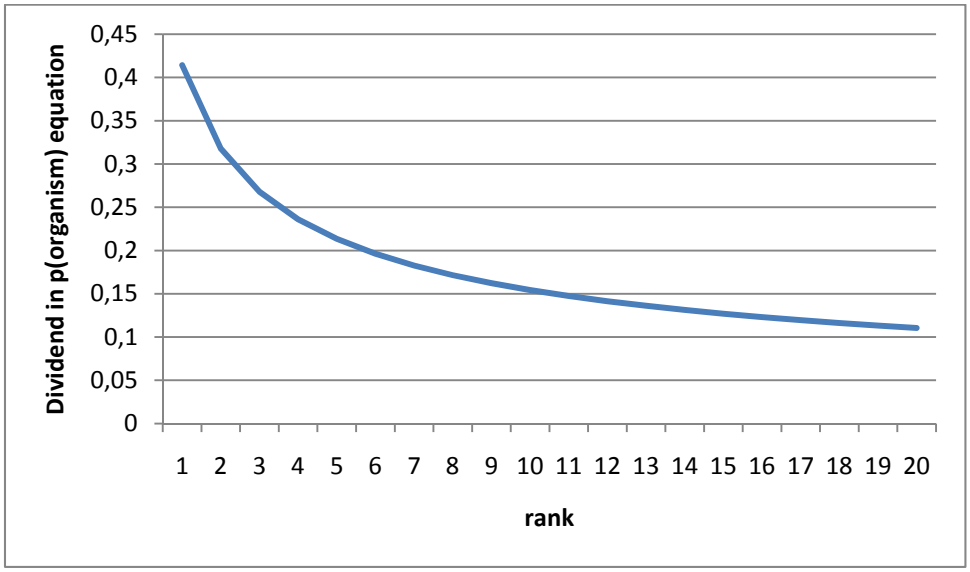


Figure 20 The result of calculating the dividend in the equation for the organism in the first rank down to the twentieth rank

When these two organisms are chosen by means of its probability calculated using the equation above, we can commence with the cross over operation. As we have mentioned before, an organism is simply an array of symbols. Cross over begins with randomly choosing whether an index of the array goes to the first mask or the second mask. A mask in this context is a collection of indices in the array and it is a requirement that the first mask and the second mask are mutually exclusive. Having these masks, the first offspring can be created by obtaining the symbols from the first candidate according to the first mask and combine it with the symbols from the second candidate according to the second mask. In the same procedure, the second offspring can be created by combining symbols from the first candidate according to the second mask and symbols from the second candidate according to the first mask. Figure 21 illustrates this procedure.

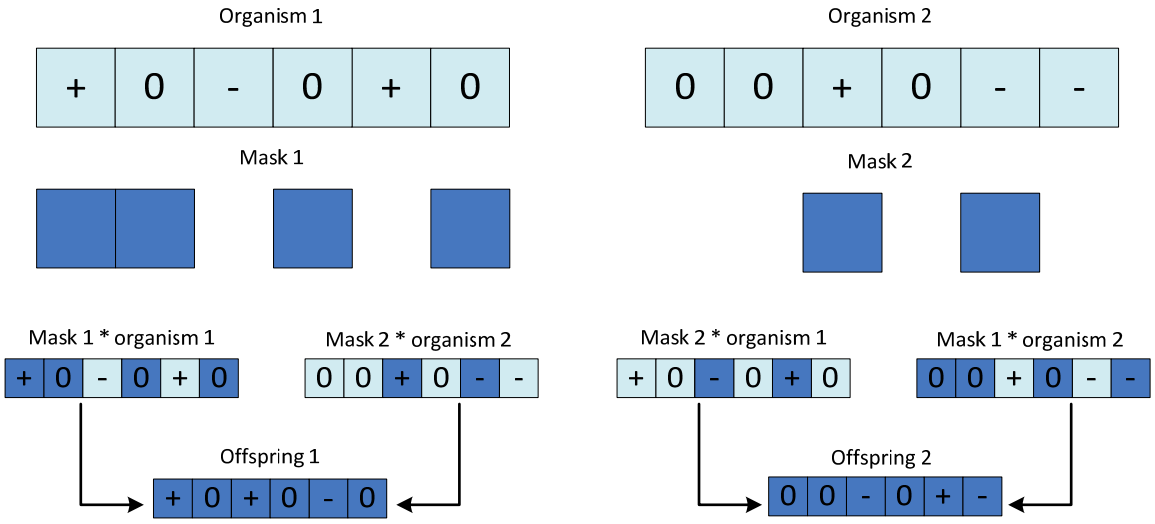


Figure 21 The illustration of how crossover is performed

4.2.1.2 MUTATION

The idea of mutation is simple. From the array of symbols, one index is chosen and the symbol in that index is altered into another symbol. The index is chosen randomly and each index is equally probable to get chosen. Figure 22 portrays the procedure of this operation. As it can be seen in Box 9, the mutation is done on two randomly chosen organisms in the population.

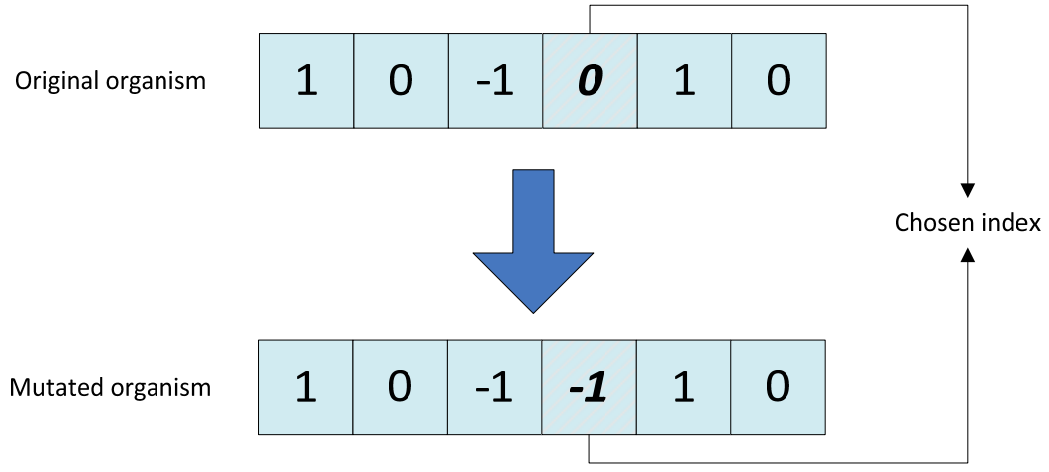


Figure 22 The illustration of how mutation is performed

4.2.1.3 BEST MUTATION

Best Mutation is mutating the organism in population that has the highest score. The mutation can be done more than one time. In these experiments, the mutation is performed a random amount of times. The aim of doing this is to find the organism that is structurally close to the best organism whilst in terms of the score it is better.

4.2.1.4 NEXT GENERATION

As consequence of cross-over, mutation and best mutation, the population grows. Selection in this context stands for selecting organisms in the population that will make it through to the next iteration.

In this thesis, the selection process is formulated as a process of choosing organisms that should be eliminated from the population. There are 2 factors to look at when eliminating organisms:

- The size of the population: The size of population is kept under a certain threshold for every iteration. This means that when the size of population exceeds that threshold, organisms that have worst scores are eliminated to keep the size of population constant.
- Identical organisms: If there are organisms that are identical to each other than one of them should be eliminated. Recognizing identical organisms can be done by comparing both arrays of symbols. However, to make the population even more diverse, the definition of identical can be revised. In this thesis, since a Bayesian Network with a large amount of variables are expected, identical organisms are two organisms that differ only in 2 places or less in their arrays. The choice of the constant 2 is arbitrary. When a more diverse population is desired and larger amount of variable is in case, a higher constant is recommended.

4.2.2 PARTICLE SWARM OPTIMIZATION

Particle Swarm Optimization is based on the idea of particles moving in a space. In this case, it is a space of Bayesian networks. This also implies that each Bayesian Network is encoded into Particle Swarm Optimization as a position in a space.

Encoding a Bayesian Network into a position is done by means of an adjacency list. An adjacency list is simply a list of each variable and its corresponding parent(s) in the Bayesian Network. An example of this adjacency list can be seen in Figure 23.

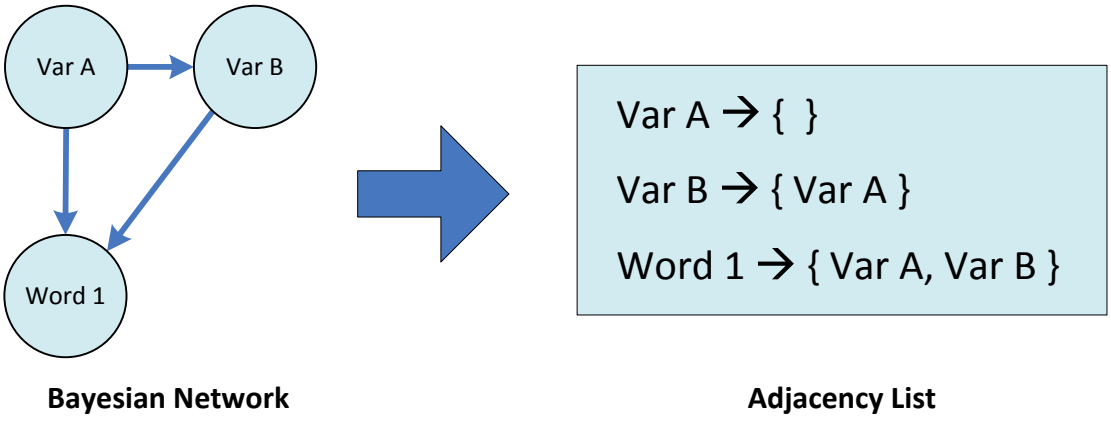


Figure 23 The translation from a Bayesian Network to an adjancency list

It is also necessary to define how velocity is represented. A velocity of the particle can be decomposed in a velocity of each variable or in other words each variable has its own velocity. The velocity of variable defines how the composition of its parents may be changed. Thus, there are 3 kinds of changes that can happen in its parents' list:

- There are variables that are added (This is referred as positive velocity)
- There are variables that remain unchanged
- There are variables that are removed (This is referred as negative velocity)

To represent this 3 changes for all variables, the velocity is implemented as a pair of adjacency lists. The first adjacency list contains the positive velocity for all variables and the second one contains the negative velocity. The variables that remain unchanged will occur in neither positive velocity nor negative velocity.

The pseudocode of Particle Swarm Optimization can be seen in Box 10.

1. Initialize all particles. Give each particle random position and random velocity. Assign each particle a set of neighbours.
2. If stopping criterion is met, go to step 5
3. For each particle i in the swarm calculate the next position ($\hat{X}_{id}$) according to the current position (X_{id}):
    a. Calculate the difference α by $\alpha = P_{id} - X_{id}$, where P_{id} is the best previous position of particle i
    b. Calculate the difference β by $\beta = P_{gd} - X_{id}$, where P_{gd} is the best previous position of particle i's best neighbours
    c. Calculate the velocity $\hat{V}_{id}$ in term of the following equation:
    $$\hat{V}_{id} = w * V_{id} + c_1 * rand(\ \ ) * \alpha + c_2 * rand(\ \ ) * \beta$$
    where w is the inertia weight factor; c_1 and c_2 are two positive constants; and $rand(\ \ )$ is the random number generator within the range $[0,1]$ and uniformly distributed.
    d. Calculate the new position of particle i ($\hat{X}_{id}$) by using the following equation:
    $$\hat{X}_{id} = X_{id} + \hat{V}_{id}$$
    e. If $\hat{X}_{id}$ has a better score than P_{id}, update P_{id}
4. If a better position is found for the whole swarm, update P_{gd} and go to step 2
5. Give P_{gd} as the most optimal solution

Box 10 How Particle Swarm Optimization is implemented

The algorithm stops when the amount of iterations reaches a maximum or when the highest score of all particles in a certain amount of iterations does not change.

As already mentioned in the Section 2.3.1.7, neighbours of the particle are not defined by the position of the particle. It is chosen beforehand and stays constant in every iteration. There are choices on how many neighbours a particle should have. Two of the most common choices are (Engelbrecht, 2007):

- Star neighbourhood structure: all particles are connected with each other.
- Ring neighbourhood structure: each particle is not connected with all other particles.

The advantage of the ring neighbourhood structure compared to the star neighbourhood structure is that a larger space is explored using the ring neighbourhood structure (Engelbrecht, 2007). This is the reason that the ring neighbourhood structure is applied in this thesis. We choose to connect each particle with a quarter of all particles. The set of neighbours is unique for each particle.

In pseudocode, calculating the position of a particle involves mathematical operation such as addition, subtraction and multiplication applied on adjacency lists. Addition between a position and a position or between a velocity and a velocity is trivial. The addition is similar to union for each list in the adjacency list.

The subtraction of two positions is less obvious. Subtracting one position from another position results in a velocity. To explain it further, consider a subtraction of position B from position A ($A - B$). Everything in position A goes to the positive velocity except everything in $A \cap B$. Furthermore, everything in position B goes

to the negative velocity except everything in $A \cap B$. The results of both positive velocity and negative velocity make the resulting velocity for the subtraction operation of B from A. An example of the subtraction between two positions and the decomposition of the velocity to positive and negative velocity can be seen in Figure 24.

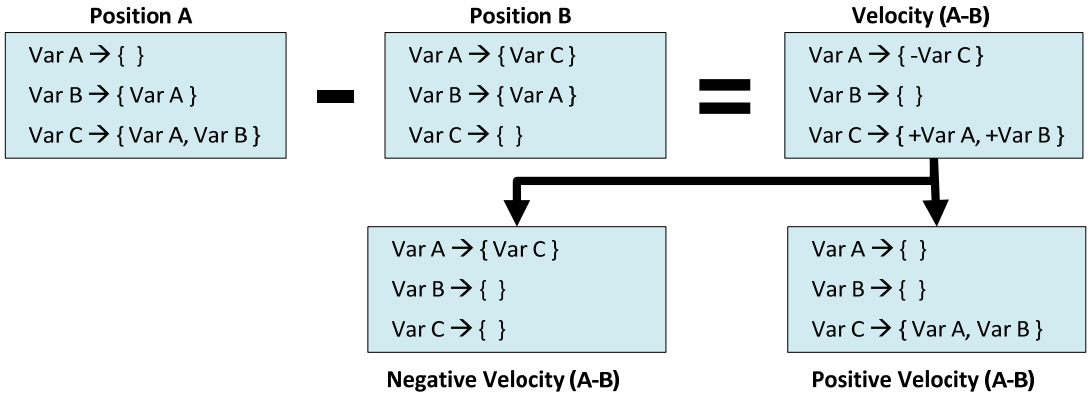


Figure 24 An example of a subtraction between two positions

In this case, multiplication is only done with a positive real number smaller than or equal to one. This multiplication represents nothing other than selecting a portion from every list in adjacency list. The size of that portion is determined by the given positive real number and this portion is chosen randomly. To clarify this, let's take an example. Consider an adjacency list $\{(A, B, C), (A, B), (\ )\}$ and an multiplier $\frac{1}{3}$. In the first list, there are 3 variables and therefore $\frac{1}{3}$ of it is chosen. Therefore, 1 variable is chosen randomly from those 3 variables. In this example let that variable be B. This also happens with the second list. By rounding up $\frac{2}{3}$, this means that 1 variable is chosen from the second list. In this example, let this be A. Obviously, there are no variable chosen from the third list because the list itself is empty. So, the result of multiplication between adjacency list $\{(A, B, C), (A, B), (\ )\}$ and $\frac{1}{3}$ is $\{(B), (A), (\ )\}$.

By applying these operations, the formula that is given in the pseudocode can be calculated to move the particle into the position for the next iteration.

4.3 QUERYING THE DATASET

As the equation of all metrics shows, the calculation of one Bayesian Network's score requires querying the dataset. How many queries are needed to calculate this depends on how many states the variables in the Bayesian Network have. However, chances are that there are a large number of states in one or more variables for the reasons that are stated in Chapter 3. When the number of variables grows, the number of queries will grow exponentially. Given this fact, querying the dataset will take most computation power in calculation of the metric. Therefore the key to make the calculation of the metrics more efficient lays in the ability to make the querying of database more efficient.

We can avoid the troubles of organizing the process of querying by using a Database Management System (DBMS). Examples of such systems are MySQL, Microsoft Access and Oracle. By making the dataset available for the DBMS, it will manage all the processes of querying and optimizing it when necessary. Querying can be done by sending a query request in a query language that is defined by the system itself. However, most of these DBMSs manage more than the tasks of querying solely. DBMS are usually implemented to cope with other database operations that are not necessary for calculating metrics such as inserting data, editing data or

deleting data. So a DBMS is too bulky for the purpose of calculating metrics and therefore the speed in querying the dataset is not optimal. For some DBMS such as MySQL, yet there are tricks to optimize querying the dataset e.g. indexing the column in the database. Unfortunately, this will not make much difference since indexing means specifying a small set of columns that are queried more frequently than other columns. In model learning, exploration is important and therefore all columns in the dataset are likely to be queried at equal frequency.

In the following subsections, the details of database querying that is implemented for the experiments are discussed. In the last subsection, suggestions for optimizing the querying will be given.

4.3.1 QUERYING IN THESE EXPERIMENTS

Because of the aforementioned reasons, we decline to use DBMS to query database. Instead, the dataset is loaded in computer memory and using an algorithm specially designed for these experiments, the querying is done.

The problem that can occur when loading all the data in the dataset into the computer memory is that the size of the dataset is too big to be loaded into computer memory. However, when all the variables are discrete, such as the dataset for these experiments, the size of the dataset can be compressed. 2 bytes (16 bits) in memory can already represent one sample data of a variable that has 65536 states or less which is sufficient for the dataset in these experiments. By compressing each sample in the dataset into this 2 byte form, the size of the dataset can be made small in memory.

In the memory, the dataset is represented in the form of a table. Each column represents a variable and each row is a data sample in the dataset. To optimize querying, indexing is done for each column in the table. The idea of indexing in this case is listing the row indices for each state of the variable where they are spread in the corresponding column.

To make the indexing clearer, let's illustrate it with an example. Consider Table 5 an example of a database that contains variable A and B. In the left column, the index of the row can be found. In this example variable A has two states (a_1, a_2) and variable B has three states (b_1, b_2, b_3).

Table 5 An example of database consists of variable A and variable B

Index \ Column Name	A	B
1	a_1	b_1
2	a_1	b_3
3	a_2	b_2
4	a_1	b_2
5	a_2	b_1

Indexing the database for columns A and B in this database results in Table 6 and Table 7 respectively.

Table 6 Indices of variable A

	Indices
a_1	{1,2,4}
a_2	{3,5}

Table 7 Indices of variable B

	Indices
b_1	{1,5}
b_2	{3,4}
b_3	{2}

From the results, it can be interpreted that for example state a_1 can be found in the row 1, 2 and 4 in the database.

Having already indexed the column, it is easier to find an answer to a query. But before we commence with how the querying is done, let's see some examples of queries. For a Bayesian Network that has 3 variables, viz.: variable A, B and C, and each has 2 states, some examples of queries will be:

- How many times does it occur that A has the first state?
- How many times does it occur that B has the second state and C has the first state?
- How many times does it occur that A has the first state, B has the second state and C has the first state?

As you can see these are not all possible queries. A query does not necessary appeal to one variable or all variables. A query can appeal to all possible combinations of variables. Ergo, listing all possible queries and calculating them beforehand is plain impossible when there are extensive amount of variables or states.

Finding the answer of a query can simply be done by scanning the whole table and counting in how many rows the condition that is posed by the query occurs in the table. This is not a very efficient strategy when handling a large dataset. In these experiments, a smarter strategy is implemented. In this strategy, the query itself is explored first by means of the information obtained from indexing. For each variable in the query condition, the number of occurrences in the table is obtained. This can be easily done by looking up the size in the indices list from indexing. When these numbers are obtained, variables in the query condition are sorted by these numbers. The list of indices for a variable with the lowest number is used to scan certain rows in the table. In this manner, it is not necessary to scan the whole table to get the result. For a pseudocode of this procedure, see Box 11.

1. From query Q, count for each condition $(V = v_i)$ how many rows in the database satisfy this condition
2. Choose the condition that has the lowest amount of rows. Denote this condition as $(V = v_c)$
3. Fetch all rows that satisfy $(V = v_c)$. Denote this portion of database as D_c
4. Count how many rows in D_c that satisfy other conditions in query Q and present it as the result of the query.

Box 11 How the answer of the query is processed in the database

This approach has saved a fair amount of computation time. There are more ways to improve the speed of computation. This will be discussed in the following subsection.

4.3.2 SUGGESTIONS IN SPEEDING UP THE QUERYING

When reducing the amount of rows that are scanned when finding the answer of a query is not sufficient, other aspects besides the scanning of table itself should be optimized.

4.3.2.1 PRECOUNT THE DATABASE

Instead of having a database filled with rows of data that may not be unique in the database itself, the amount of rows in the database can be reduced by identifying the unique data in the database and counting how many times that unique data occurs in the original database. Having these unique data and the frequency of that unique data, the example in Table 5 can be transformed into the following form:

Table 8 An example of database after precounting the database

Index \ Column Name	A	B	Frequency
1	a_1	b_1	1
2	a_1	b_3	1
3	a_2	b_2	1
4	a_1	b_2	1
5	a_2	b_1	1

The advantage of precounting the database is that when there are many duplicates in the database which is always the case in a language model database, the number of rows can be reduced significantly. When the number of row is reduced, querying becomes faster because there will be less rows to iterate. This also contributes in making the size of database smaller so it is more feasible to be loaded in the computer's RAM memory.

4.3.2.2 GROUPING THE QUERIES

When analyzing the querying behavior of the metric calculation, queries for all possible state combinations of the parent variables and the variable itself are done consecutively. The idea of grouping the queries is

portrayed in the following example. Consider variables A and B that have each 2 states. Two queries that can come from these variables are:

- (Variable A == first state) and (Variable B == first state)
- (Variable A == first state) and (Variable B == second state)

Given the aforementioned scanning procedure, instead of counting only for condition (variable B == first state) when scanning the rows, counting for conditions (Variable B == first state) and (Variable B == second state) can be done in one scan. In this example, this already saves the half of computation time.

To save more computation time when calculating the metric, grouping of the queries should be done in such way that variable B in the example is the variable that has the highest number of state.

4.3.2.3 SAVING THE RESULT FROM PREVIOUS METRIC CALCULATION

As can be seen from the discussion above, querying the dataset is the most expensive computation in the calculation of a metric. That's why it can save a lot of computation time, when the metric calculation that takes a certain computation time is saved in order to avoid recalculation of the metric next time.

4.3.2.4 DESIGNING THE DATABASE AS RELATIONAL DATABASE

By taking examples from relational database, reorganize the database table into multiple small databases. These databases will link with each other. For example, a database that contains conversations and the details of its speakers can be reorganized into two smaller tables: the one with the conversations and the one with only the details of each speaker. In this example, the database of the conversations will link to the database of speaker details when the data about the speaker in a certain conversation is needed.

Reorganizing the database in such way is also a form of compressing the database. When implementation of the querying also takes this reorganizing also into account, the amount of the columns that need to be checked will also decrease.

4.3.2.5 KNOWING WHICH QUERIES TAKE MOST OF THE CALCULATION TIME

When we know that a group of queries which includes certain variables needs a lot of calculation, the result of these queries can be counted beforehand and saved as additional data. This means that when the result of one of these queries is needed, no calculation is needed.

In calculation of metric score, it always takes a long time to find the answer of queries that includes variable that has the largest number of states. It is then recommended to find the answer that includes this variable beforehand.

4.3.2.6 SPREADING THE COMPUTATION

When there is more than one computer available to run the experiments, the equation of the metric can be decomposed in such way that querying the dataset can be outsourced to other computers. The computation will then be done parallel and therefore save a lot of computation time compared to sequential computation. However, the biggest obstacle will be the communication time between the computers. When the communication time takes longer than the computation itself, the computation as a whole will not be improved. It can also get worse and the computation will take a lot longer compared to sequential computation.

4.4 ACYCLICITY

A hard constraint for any Bayesian Network is that it should never contain a cycle. That's why when exploring possible graphs, there is a need to recognize a cycle and eliminate these cycles to get a valid candidate Bayesian Network.

In the following subsections, the technique to implement cycle listing will be discussed. Furthermore, how cycle elimination is done in this thesis will be explained.

4.4.1 CYCLE LISTING

There are many algorithms invented to list the cycles in a directed graph. One of the most advanced algorithms in listing the cycles is an algorithm introduced by Donald B. Johnson in (Johnson, 1975) and this is also the algorithm that is implemented in these experiments. This algorithm is based on depth-first search technique which makes it possible to limit the time bound of $O\big((n+e)(c+1)\big)$ in the graph where n is the amount of variables, e is the amount of edges and c is the amount of cycles.

The pseudocode of this algorithm is shown in Box 12.

```
begin
      integer list array A_K(n), B(n); logical array blocked(n); integer s;
      logical procedure CIRCUIT (integer value v);
            begin logical f;
                  procedure UNBLOCK (integer value u);
                        begin
                              blocked (u) := false;
                              for w ∈ B(u) do
                                    begin
                                          delete w from B(u);
                                          if blocked(w) then
                                          UNBLOCK(w);
                                    end
                        end UNBLOCK
                  f := false;
                  stack v;
                  blocked(v) := true;
            L1:   for w ∈ A_K(v) do
                        if w = s then
                              begin
                                    output circuit composed of stack
                                      followed by s;
                                    f := true;
                              end
                        else if blocked(w) is false then
                              if CIRCUIT(w) then f := true;
            L2:   if f then UNBLOCK(v)
                  else for w ∈ A_K(v) do
                        if v ∉ B(w) then put v on B(w);
                  unstack v;
                  CIRCUIT := f;
            end CIRCUIT;
      empty stack;
      s := 1;
      while s < n do
            begin
                  A_K := adjacency structure of strong component K with
                        least vertex in subgraph of G induced by
                        {s, s + 1, ..., n};
                  if A_K ≠ ∅ then
                        begin
                              s := least vertex in V;
                              for i ∈ V_K do
                                    begin
                                          blocked(i) := false;
                                          B(i) := ∅;
                                    end;
            L3:                     dummy := CIRCUIT(s);
                                    s := s + 1;
                        end
                  else s := n;
            end
end;
```

Box 12 The algorithm to find all the cycles in the directed graph

In the pseudocode, an algorithm to find the strong component is utilized. The details of this algorithm can be found in (Tarjan, 1973).

4.4.2 CYCLES ELIMINATION

The aim of cycles elimination is to obtain a directed graph that is acyclic. However, there are many ways of modifying edges in a graph to get to the point where the graph is acyclic. When it comes to modifying edges in the graph, in this thesis we strive to reserve the original edges as much as possible. This is done by deleting the edges that occur frequently in the cycles.

Given the list of cycles obtained with the algorithm discussed in the previous subsection, first thing we do is count for each variable how many times it occurs in that list of cycles. Based on how frequent a variable occurs in the cycles, we rank the variables. Assuming that the variables in the edge that occurs the most are also high ranked. We pick edges composed by the highest rank variable and the second highest rank and delete that edge from the graph. Then, we go off the list of cycles and delete the cycles in the list that contain that edge. We repeat this procedure until there are no cycles in the list of cycles anymore.

The details of this procedure is put into details in Box 13.

1. Get the list of cycles C
2. Count for every variable in the graph how many time it occurs in C
3. Given V is an array that contains all variables, sort V based on how many times variable occurs in C
4. Pick the first variable in array V. Call this variable v_1.
5. Pick the second variable in array V. Check whether there is an edge between them. If there are no edge between them, pick second, third, fourth and so on. Call this variable v_2.
6. Eliminate the edge between v_1 and v_2 and remove all cycles in C that contains this edge.
7. If there is still cycles in C, go back to step 2.

Box 13 The procedure of cycles elimination

4.5 PROGRAMMING ENVIRONMENT

The model learning techniques and experiments for this thesis are implemented in the C++ language. C++ is chosen in this case because this programming language is supported by Visual Studio which makes it more comfortable and quicker to produce the source codes. Furthermore, it is well accepted that the resulting software written in C++ is faster than other object-oriented programming language. This is favourable for implementation of the experiments since its computation is extensive.

Furthermore, the implemented software only uses Boost (Dawes, 2009) as external library to generate random numbers. The databases for the experiments are stored in a hard disk as a simple text file that is loaded into RAM memory when the access to the database is needed.

In Appendix A, the design of the software modeled with class diagram can be found.

5 EXPERIMENTS

In this chapter, we will discuss the results from experiments with the models and algorithms discussed in Chapter 3 Modeland 4. First, we will start off by explaining the rationale of the experiments that are performed. In the next section, the two kind of dataset that are used in these experiments are explained. The subsequent sections are dedicated to explain each experiment and discuss the result of this experiment.

5.1 RATIONALE OF EXPERIMENTS

Each experiment will test how effective and how efficient the techniques are. This can only be realized when the original model is known, so the learnt Bayesian Network can be compared with the original model to evaluate effectiveness and efficiency. Applying real conversational data as observational data is in this case not an option because the underlying model still has to be learnt and it is thus unknown. To tackle this problem, an artificial language is created. This artificial language is generated according to a certain Bayesian Network that can be chosen freely. This makes it possible to generate sentences that follow the dependency and independency of a chosen Bayesian Network.

The first experiment will involve learning the Bayesian Network for language model based on sentences from artificial languages. This is done using the aforementioned score-based strategy. In the second experiment, constraints based on prior knowledge concerning speech recognition system will be added in the score-based strategy to see whether it will help to increase effectiveness and efficiency. This is also tested using sentences from the artificial language. The third experiment will test the techniques used on previous experiments on real conversational data.

5.2 DATASET

There are two kinds of dataset that are in use in these experiments. These are artificial language data and real conversational data. In the next subsections, each of the data will be explained further.

5.2.1 ARTIFICIAL LANGUAGE DATA

Artificial language is a fake language that is made up for the purpose of mimicking real language in a smaller extent. It is not essential which words are in use in this language, since the language is created to put a certain language model that the artificial language's creator has in mind into practice. The dataset of artificial language data is simply a collection of sentences and value of other non-word variables in the language model.

In the first and second experiment, artificial language data is used to make evaluation of model learning techniques more obvious. When we know the language model that constitute the sentences in the dataset, comparing it with the result of model learning will give a certain measure of how successful model learning techniques in revealing the language model.

To create the sentences in the dataset, random sampling is applied to generate an instance of each variable for the corresponding sentence. The random sampling follows the conditional probability table of each variable that is defined by artificial language's creator. As we have expected, the more sentences are generated, the more convergent the distribution of the generated dataset to the defined conditional probability tables.

For this experiment, the generation of sentences in dataset is explained in the following Box 14. The algorithm in Box 14 incorporates 2 assumptions that simplify the generation of sentences. These are:

- The word variable in language model does not influence the generation of instance of other variables.
- All variables except word variable are non-temporal. This means that for each sentence there is only one instance generated for each sentence and it is duplicated for each word in the sentence.

1. Check whether the instance of all variables except word variable is already generated
    a. If it is, then go to step 4
    b. If it is not, then go to step 2
2. Go through each variable except word variable.
   For each variable, if the instance is not yet generated and the instances of the corresponding variable's parents are already generated, then generate the instance of this variable by means of the conditional probability table.
3. Go back to step 1
4. Generate the instance of word variable until the end of sentence is reached.

Box 14 How the sentences in artificial language are generated

The word is picked according to how the artificial language's creator defines the role of previous words in a sentence and its parents in the model. These influences are defined in the conditional probability table. The end of sentence is reached when the word variable generates a dummy word "end_of_sentence".

5.2.2 REAL CONVERSATIONAL DATA

Whilst the artificial language is used to make evaluation and interpretation of the results easier, the real conversational data is used in this experiment for reason of exploration. The result of the model learning for real conversational data will be a language model that can be incorporated as part of Automated Speech Recognition model.

For this experiment, the real conversational data is taken from The Spoken Dutch Corpus project (Dutch Language Union, 2001). The dataset contains approximately 150.000 unique words spread through more than million sentences. In total there are nearly 9 million words in the dataset.

Besides words in the sentence, there are other variables annotated in each sentence in the dataset of The Spoken Dutch Corpus project. Some examples of these variables that are available are:

- Gender of the speaker
- Language spoken (Flemish or Dutch)
- Education level of the speaker
- Occupation of the speaker
- Context of the conversation (Scripted or unscripted; topic of the conversation)
- Interactions between the speakers (No interaction, some interaction or full interaction)

To make the experiment more efficient, only words that occur more than 10 times in the dataset are used. This leaves us with 20.000 unique words cutting the dataset to 8 million words.

5.3 EXPERIMENT 1

In this first experiment, the effectiveness of each metric (MDL, BSe, Perplexity and Modified Perplexity) and the efficiency of each algorithm (Genetic Algorithm and Particle Swarm Optimization) will be analyzed based on how it performs on the artificial language.

First of all, 15 Bayesian networks ranging from 2 to 4 variables are chosen to test these metrics and algorithm. These Bayesian networks are depicted in Figure 25.

Each of these Bayesian networks can be considered as the language model of the artificial language. From this point on we refer to one of 15 Bayesian network by the number that the Bayesian network assigned in Figure 25.

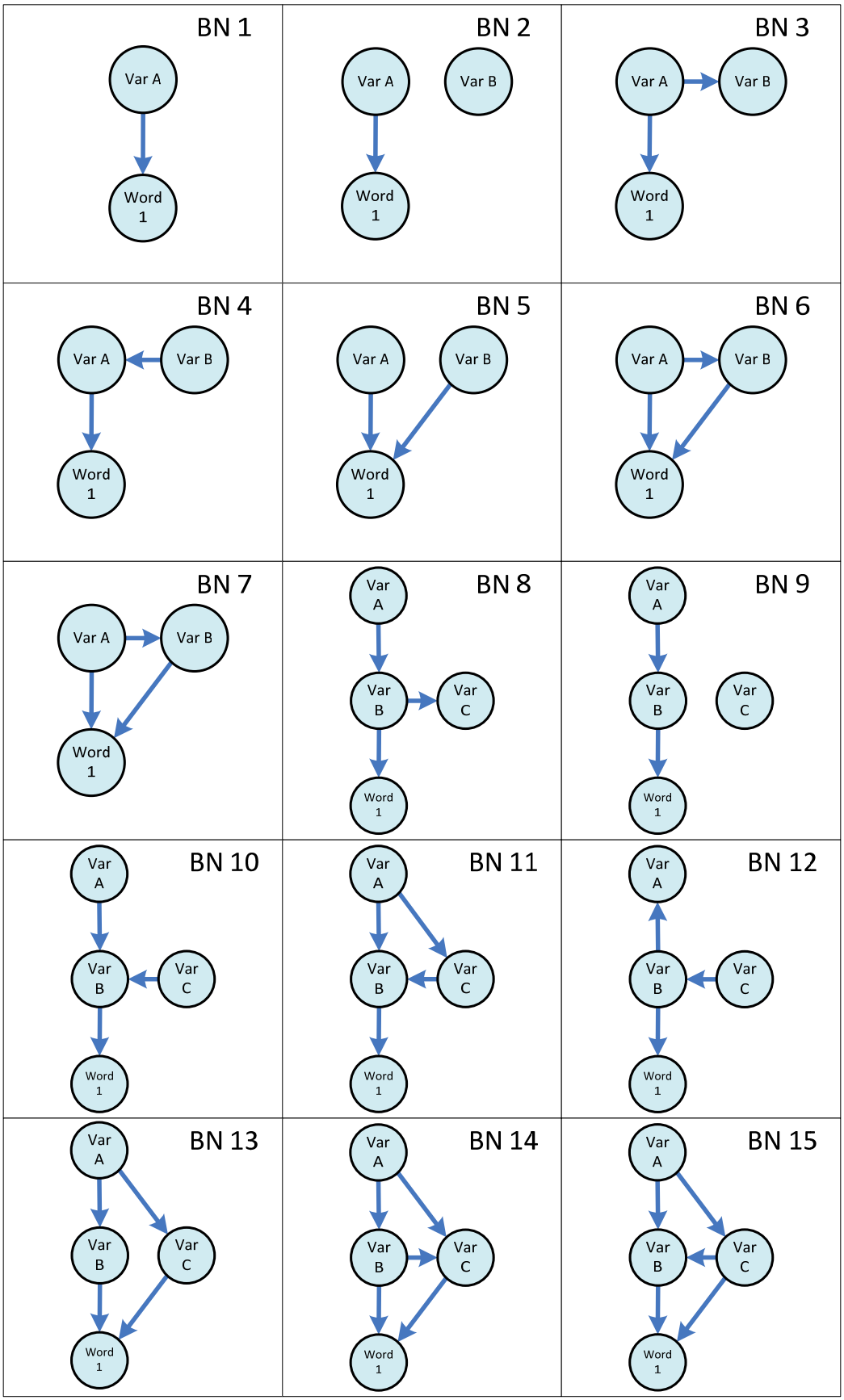


Figure 25 All Bayesian Networks that will be used in this experiment

5.3.1 ANALYZING THE METRICS

To test how the metrics perform, each metric is combined with the Genetic Algorithm to find the most optimal Bayesian Network according to each of those metrics. In this experiment, we make sure that the Genetic Algorithm always gives the most optimal solution. For Bayesian networks with 2 to 4 variables, it is still manageable to determine whether the result of the Genetic Algorithm is always optimal and the empirical data from our tests show that in this case Genetic Algorithm gives the most optimal Bayesian Network.

For each one of 15 Bayesian networks, a set of sentences is created and those sets of sentences are used as observational data to learn the Bayesian Network. The outcome will be analyzed with the original Bayesian network in mind. There are a few factors to look at when comparing those two Bayesian network:

1. How many edges does the original Bayesian network have?
2. How many edges does the learnt Bayesian network have?
3. How many edges are preserved in the learnt Bayesian network?
4. How many preserved edges have the same direction?

Obviously, the learnt Bayesian network is considered to have completely successful when factor 1 is equal to factor 2 and both factor 3 and 4 are equal to factor 1.

To test the influence of the size of the observational data in determining the metric, for each artificial language 9 sets of data varying on the amount of sentences in it are generated. These 9 datasets are datasets that contain 10, 50, 100, 500, 1000, 5000, 10000, 50000 and 100000 sentences. For each of these datasets, the combination of a metric and the Genetic Algorithm is run.

All results of this experiment can be seen in Appendix B. To analyze the characteristics of each metric, we pick out the result for the artificial language from BN 1. The graphs showing the results for this artificial language using MDL metric, BSe metric, Perplexity metric and Modified Perplexity metric are depicted in Figure 26, Figure 27, Figure 28 and Figure 29 respectively.

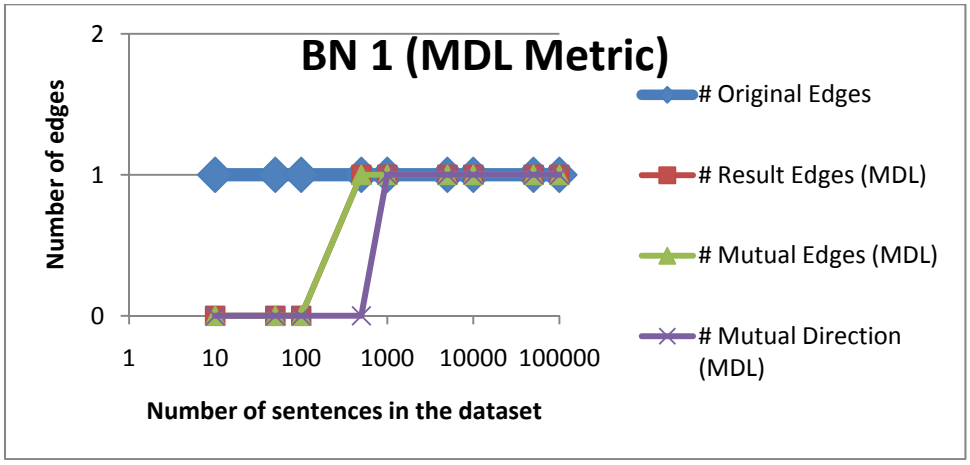


Figure 26 The result for BN 1 using MDL metric

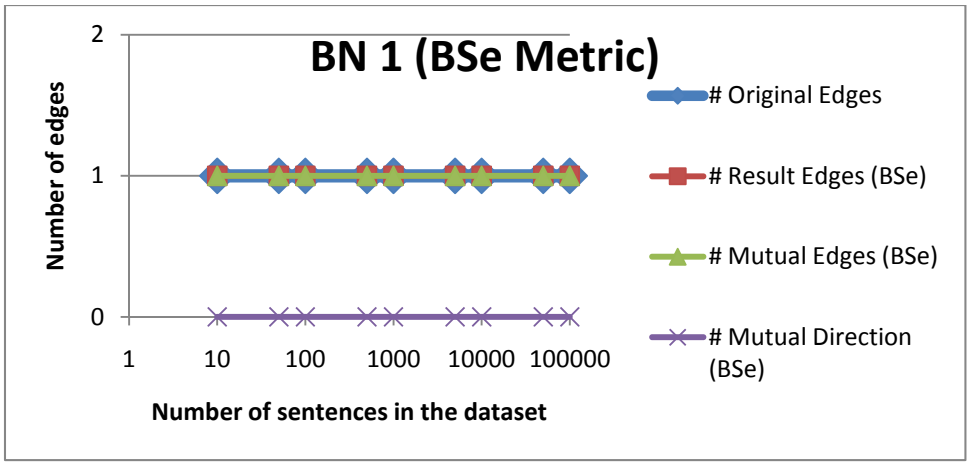


Figure 27 The result for BN 1 using BSe metric

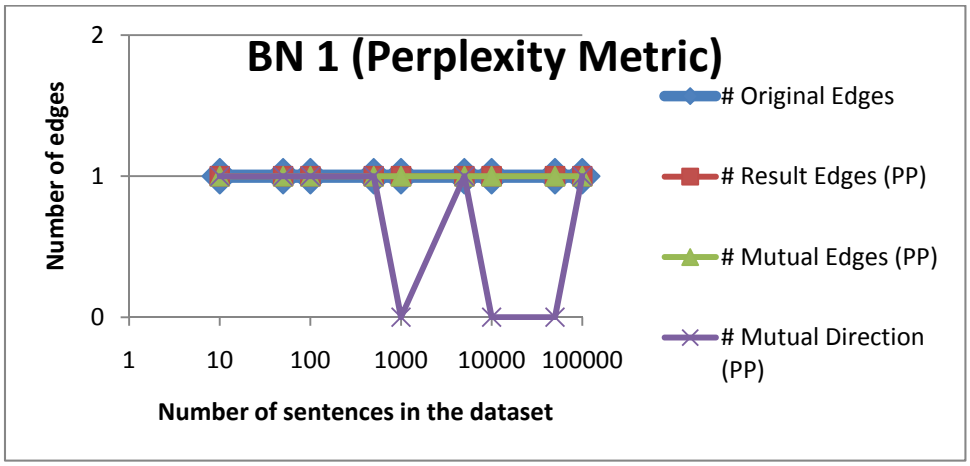


Figure 28 The result for BN 1 using Perplexity metric

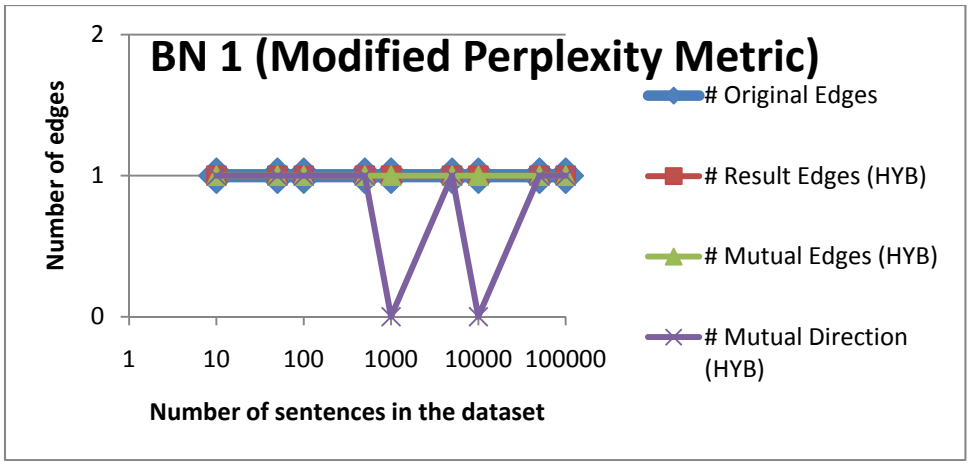


Figure 29 The result for BN 1 using Modified Perplexity metric

The underlying Bayesian network in this case only has two variables which should be pretty simple to learn. This also shows in the result that given a sufficient amount of sentences (we will discuss this later), each metric can see the dependency between those two variables. However, the metric does not always succeed in learning the direction of that dependency. By further examining the score of each metric for this case, it also shows that:

- The MDL metric and Perplexity give the same score for equivalent Bayesian Networks (for the definition of equivalent see (Verma & Pearl, 1990)). This means that when the direction of BN 1 is inversed, the score is still the same. This explains the fluctuation of the preserved direction in the results for MDL metric and Perplexity
- In contrast with MDL and Perplexity, BSe and Modified Perplexity do not assign the same score to each equivalent Bayesian network. From the result in this case, we see that the BSe metric always gets the direction wrong and Modified Perplexity gets the direction right most of time.

Another interesting result to discuss here are the results for the artificial language from BN 2 and BN 12. The graphs for BN 2 using MDL, BSe, Perplexity and Modified Perplexity are depicted in Figure 30, Figure 31, Figure 32 and Figure 33 respectively, while Figure 34, Figure 35, Figure 36 and Figure 37 show the graph for BN 12.

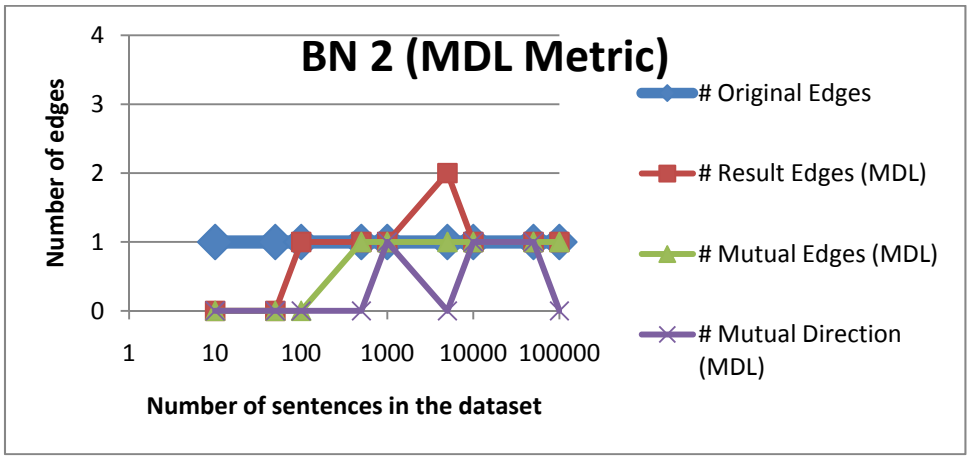


Figure 30 The result for BN 2 using MDL metric

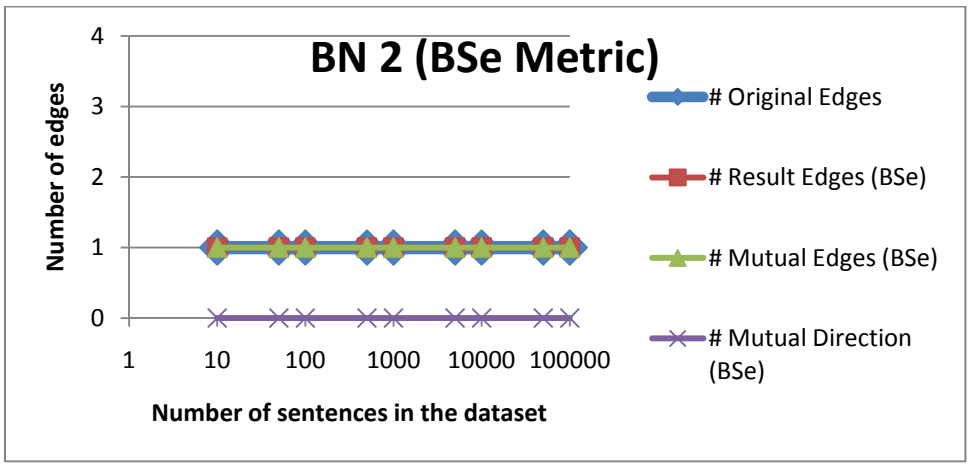


Figure 31 The result for BN 2 using BSe metric

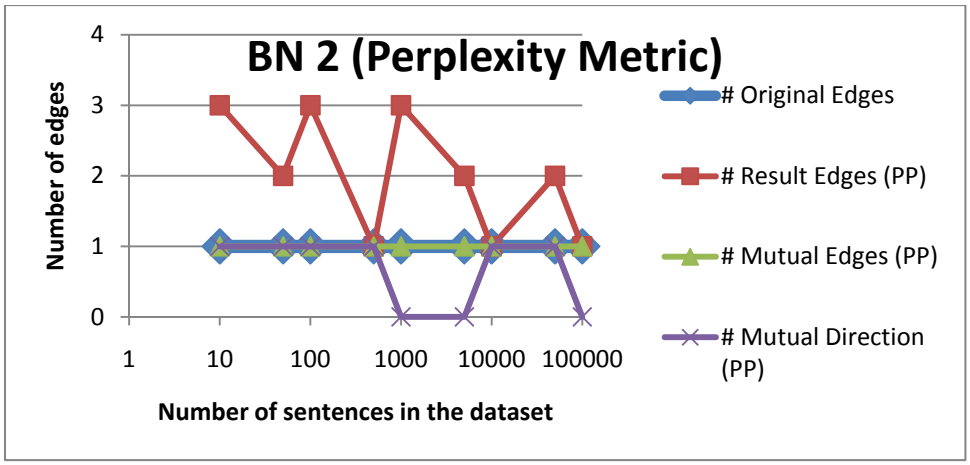


Figure 32 The result for BN 2 using Perplexity metric

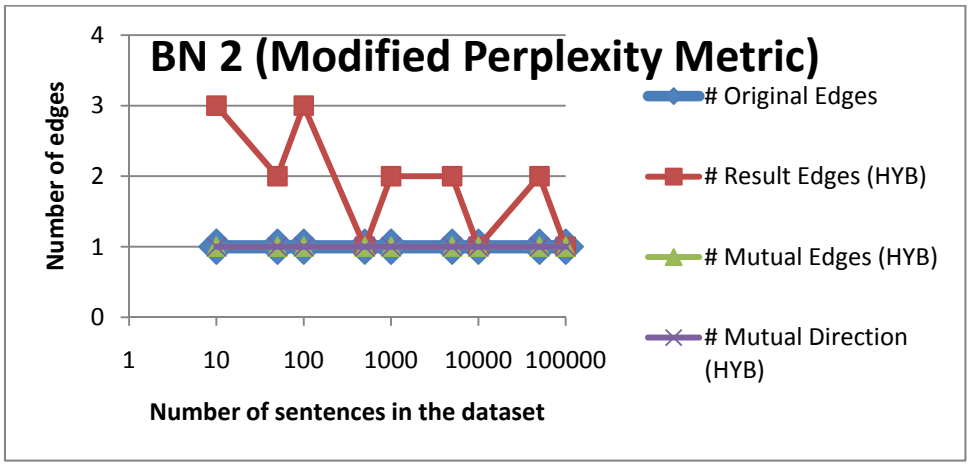


Figure 33 The result for BN 2 using Modified Perplexity metric

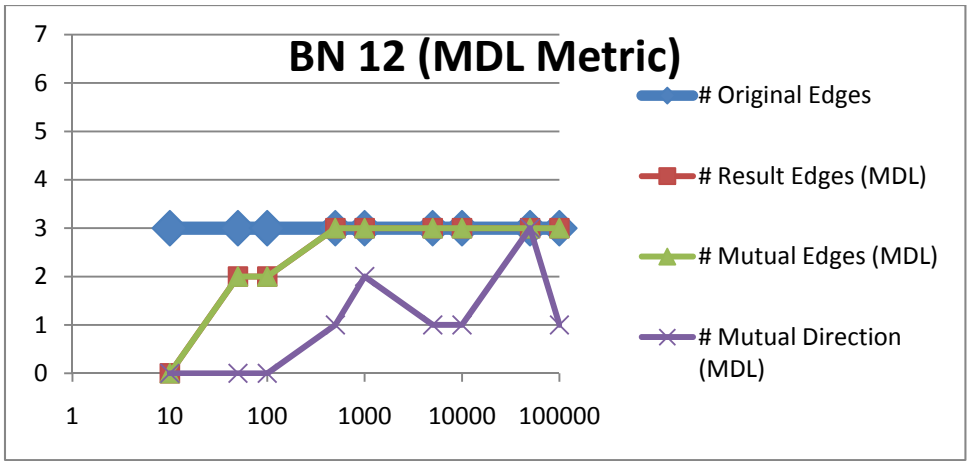


Figure 34 The result for BN 12 using MDL metric

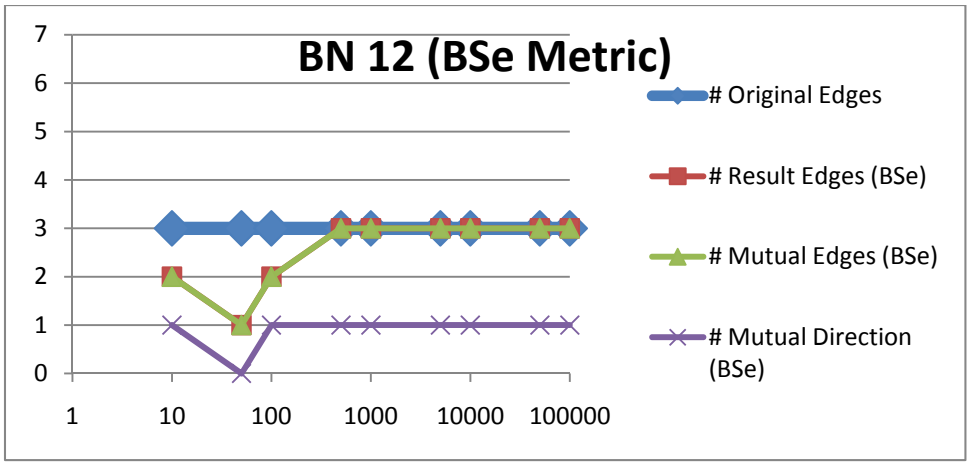


Figure 35 The result for BN 12 using BSe metric

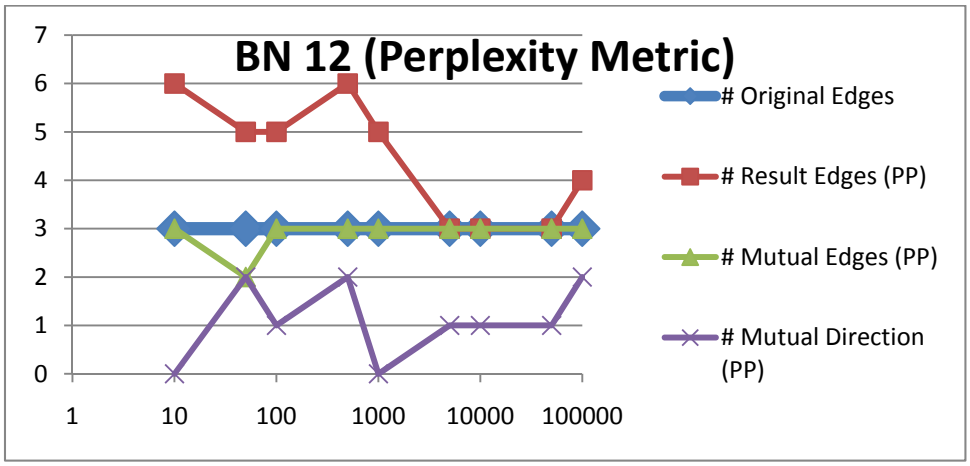


Figure 36 The result for BN 12 using Perplexity metric

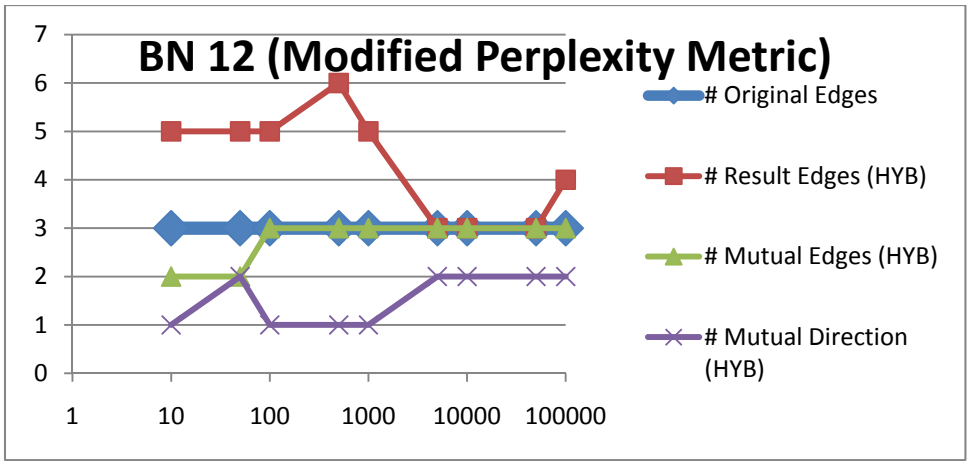


Figure 37 The result for BN 12 using Modified Perplexity metric

Analyzing these result (and from another artificial language) we can see an interesting trend. When given insufficient data, MDL and BSe are very conservative. MDL and BSe tend to give less edges compared to the

original model when they suffer from data insufficiency. The explanation of this trend lies in how MDL and BSe metric are constructed. These two metrics incorporate the complexity of the Bayesian Network in order to punish complex networks. This is not the case with Perplexity and Modified Perplexity. On the contrary, these two metrics give a complex network when there is insufficient data. For Perplexity and Modified Perplexity insufficient data means that the data is undersampled and therefore the structure in the data does not show. Thus, the best way to explain the data when knowledge of the structure is lacking according to Perplexity and Modified Perplexity is to give a Bayesian Network where there are many dependencies between variables.

On the other side when given a sufficiently large dataset, the amount of edges converges to the amount of edges in original Bayesian network. So, when there is sufficient amount of data, all metric will succeed in learning the Bayesian network close to the original Bayesian network

Confirming the aforementioned analysis of BN 1, when it comes to learning the direction of the edges in the original model, all metrics still give unsatisfying results. This can be seen clearly in experiment for BN 12 where there is a large number of possible edges in the resulting Bayesian Network.

We also point out, because of the artificial language's nature that is produced by random sampling, that the result can deviate with a margin of 1 edge when multiple trials on the same Bayesian Network are performed. This can be seen in Figure 38 and Figure 39. In Figure 38, BN 5 is used as the original Bayesian network, while in Figure 39 BN 9 is the original Bayesian Network. Because in BN 9 there are more edges possible, it can be seen that MDL and BSe which are conservative in their result are more stable than Perplexity and Modified Perplexity.

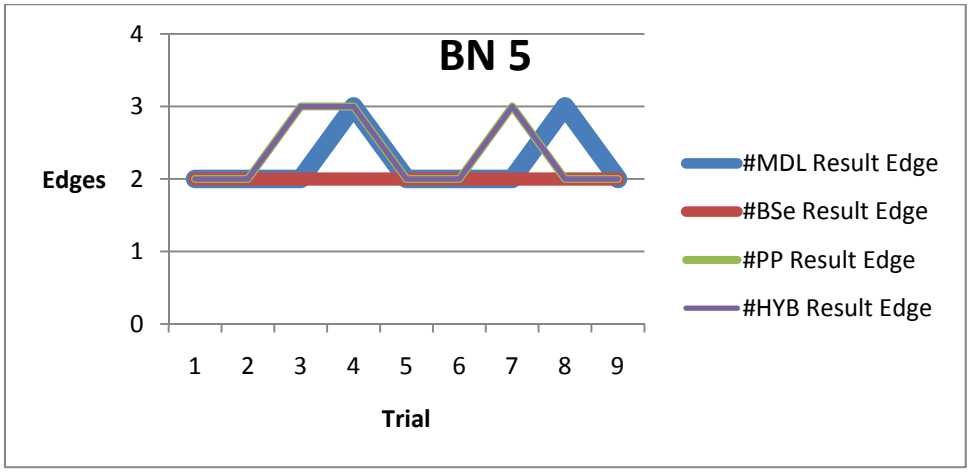


Figure 38 Multiple trials using BN 5

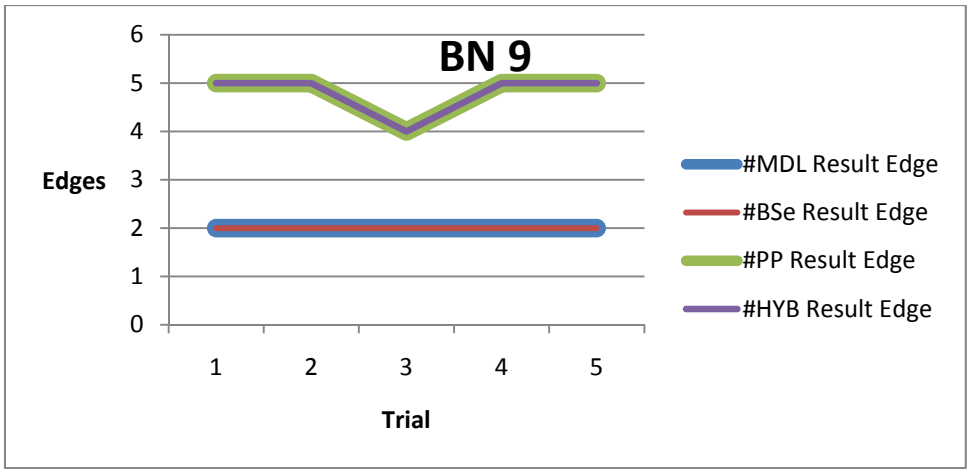


Figure 39 Multiple trials using BN 9

It is also interesting to see what happens when one variable, in this case the word variable, has more and more states. From Figure 40, Figure 41, Figure 42 and Figure 43, it is seen that the effect of increasing the number of states in a variable has a reverse effect of increasing the number of sentences. It can be seen that for Perplexity and Modified Perplexity there are more and more edges recognized when the amount of states in the word variable is increased. MDL and BSe are more stable. Because increasing the number of states has a reverse effect compared to increasing the number of sentences, it is fair to conclude that the ratio of all possible states and number of sentences influences the result of model learning. The lower the ratio, the closer the result will be to the original model.

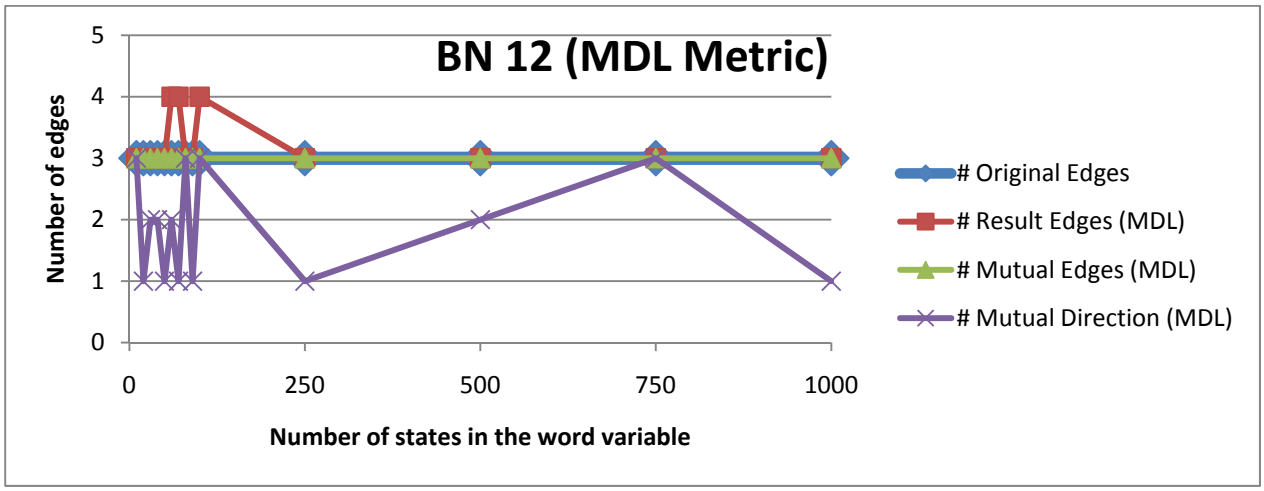


Figure 40 The result of increasing number of states in word variable gradually for BN 12 using MDL metric

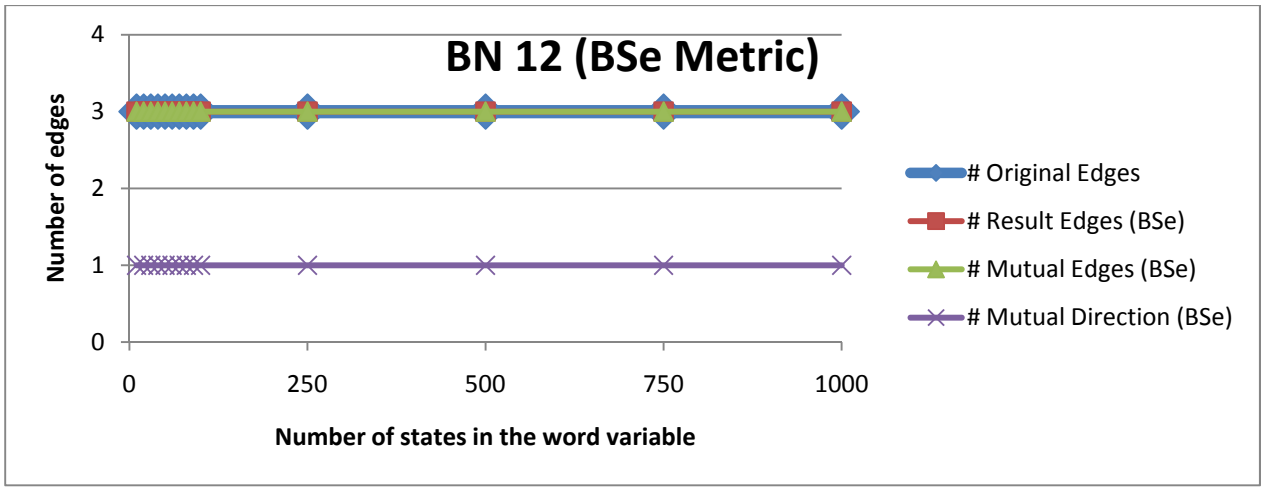


Figure 41 The result of increasing number of states in word variable gradually for BN 12 using BSe metric

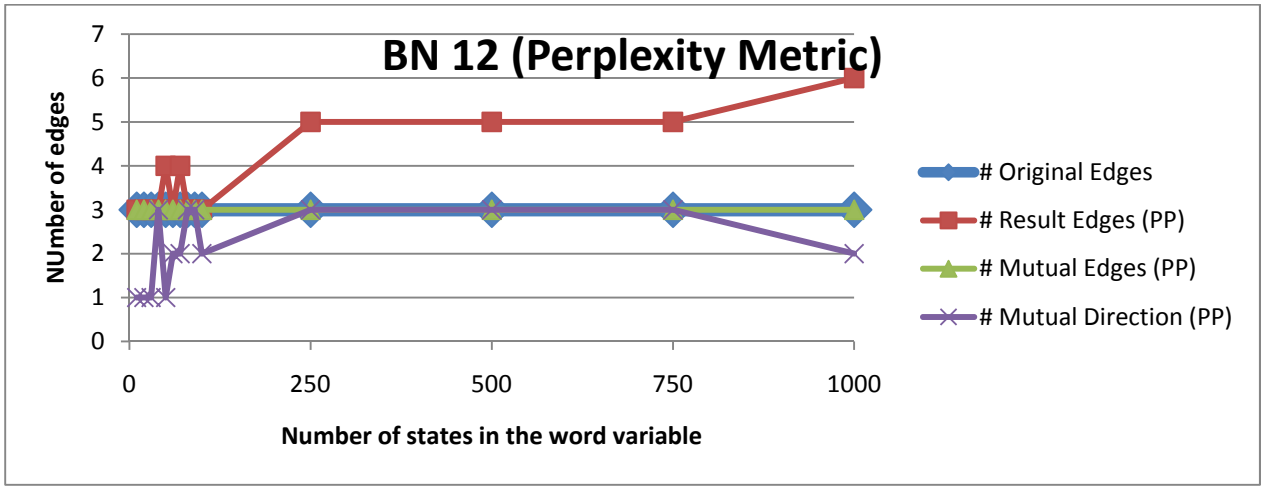


Figure 42 The result of increasing number of states in word variable gradually for BN 12 using Perplexity metric

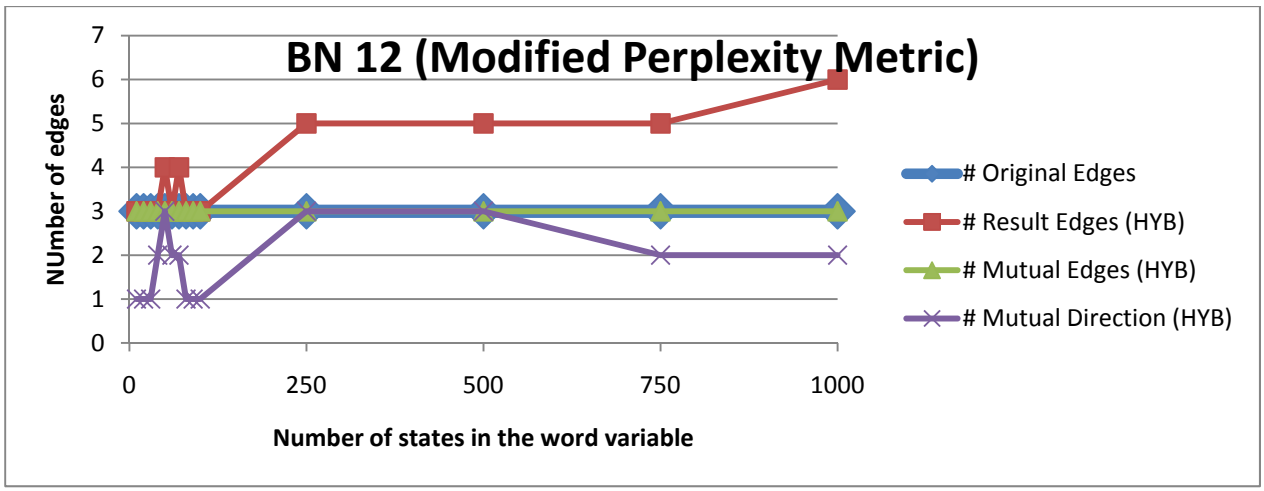


Figure 43 The result of increasing number of states in word variable gradually for BN 12 using Modified Perplexity metric

5.3.2 ANALYZING THE ALGORITHMS

In these experiments, there are two algorithms that are tested: Particle Swarm Optimization and the Genetic Algorithm. By applying these with the same dataset and the same metric, we would like to know how these algorithms perform compared to each other.

Figure 44 is a graph that shows the most optimal score for each iteration in Genetic Algorithm and Particle Swarm Optimization. This graph is made from the experiment using MDL metric for dataset originated from BN 15. This graph is chosen arbitrary, since the graphs from other datasets also follow the same pattern as this one. From Figure 44, it can be seen that Particle Swarm Optimization takes less iterations to get to the optimal result than the Genetic Algorithm. It can be seen that the Genetic Algorithm takes small steps regarding the score to go to the optimum, whereas one iteration in Particle Swarm Optimization can make a large difference in score.

It is tempting to conclude that Particle Swarm Optimization is more efficient than the Genetic Algorithm. However, if we analyze the details of Particle Swarm Optimization, given that there are N particles at work, Particle Swarm Optimization explores N Bayesian Networks in one iteration. When N is a high number, which it almost always is (in this thesis N is 40), Particle Swarm Optimization has done more exploration than the Genetic Algorithm in one iteration. In one iteration, the Genetic Algorithm is designed to only create 2 offsprings from crossover and a few from mutations. It is then logical that the Genetic Algorithm takes more iteration than Particle Swarm Optimization. From these experiments, we cannot conclude which algorithm is more efficient than the other.

However, concerning the optimality of the result, the results of the experiments show that both algorithms always get to the most optimal result. Based on the first experiment, there is no proof that one algorithm is favorable to another.

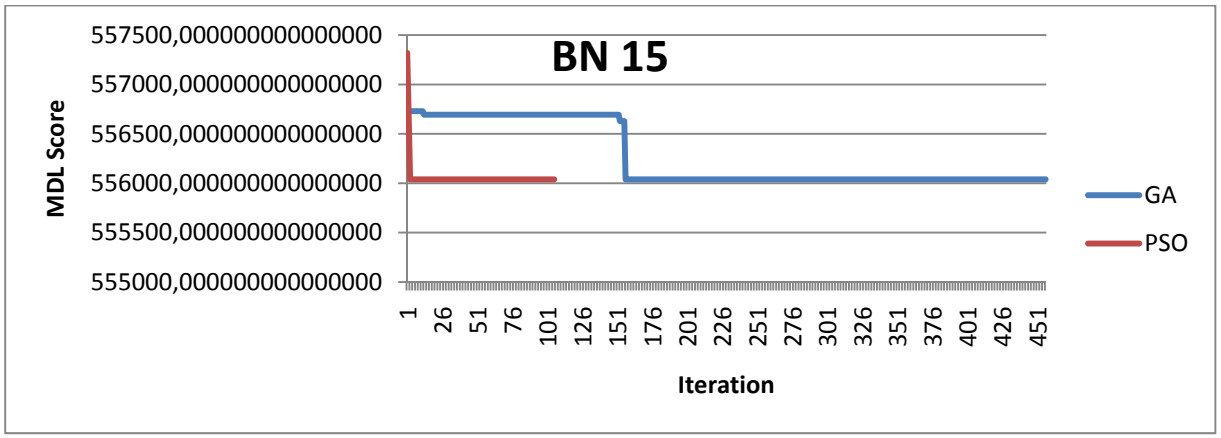


Figure 44 Comparison between the effeciency of Genetic Algorithm and Particle Swarm Optimization

5.3.3 CONCLUSION

After analyzing the results of the experiment 1, we can see that each one of four metric has its own characteristics. The characteristics of these four metrics are summed up in Table 9

Table 9 Characteristics of the metrics

Metric	Characteristics
MDL metric	• Give the same score for equivalent Bayesian Networks • Tend to give less edges compared to the original model when they suffer from data insufficiency
BSe metric	• Give different score to two unidentical equivalent Bayesian Networks • Tend to give less edges compared to the original model when they suffer from data insufficiency
Perplexity metric	• Give the same score for equivalent Bayesian Networks • Tend to give more edges compared to the original model when they suffer from data insufficiency
Modified Perplexity metric	• Give different score to two unidentical equivalent Bayesian Networks • Tend to give more edges compared to the original model when they suffer from data insufficiency

The data insufficiency itself can be identified by calculating the ratio of all possible states and number of sentences influences the result of model learning. The lower the ratio is, the more sufficient the data is. When the ratio is high, the effect of the result of the model learning will depend on the characteristics of the applied metric.

On the other side, two algorithms in which the experiments are done do not show much difference. Genetic Algorithm and Particle Swarm Optimization always produce the most optimal result during the experiments. Particle Swarm Optimization takes less iteration to come up with the most optimal result, but one iteration in Particle Swarm Optimization explore more Bayesian networks than Genetic Algorithm.

However, the amount of the edge's direction that is recognized in the results of this experiment is unsatisfying. More prior knowledge is needed to be incorporated for a better result regarding to the directions of the edges. This can be done by putting constraints in the model that the algorithm explores.

5.4 EXPERIMENT 2

Analyzing the results from the first experiment, there is a need to put constraints on the resulting models to make the directions of dependency more accurate. These constraints are based on valid assumptions of the general aspects concerning language modeling.

1. Future events can not influence present outcomes. This implies that the variable at timestep t can not cause any variable at timestep t-1 or lower.
2. The process is stationary. This implies that A dependency in a timestep t is also valid in other timesteps.
3. Word variables can not cause non-word variables.

These constraints are inserted into the algorithm for each offspring created in the Genetic Algorithm or each new position reached by a particle in Particle Swarm Optimization. With these constraints, the same experiments as in the previous section with 9 sets of data varying on the amount of sentences are run. Because the constraints also include cases for learning Dynamic Bayesian Network, experiment with artificial language that is produced from Dynamic Bayesian Network with 2 timeslices are performed.

5.4.1 EXPERIMENT WITH BAYESIAN NETWORK

The results of the experiment for BN 1 can be seen in Figure 45, Figure 46, Figure 47 and Figure 48. The difference between these results with the results for the one without any constraints is in the absence of the randomness when it comes to the direction. It can be seen in the graphs that when the edge is recognized, the direction is also set in a correct way. This is different than the results of the first experiment when especially for the Perplexity and the Modified Perplexity metrics the direction of the edges is never constant when varying the number of sentences.

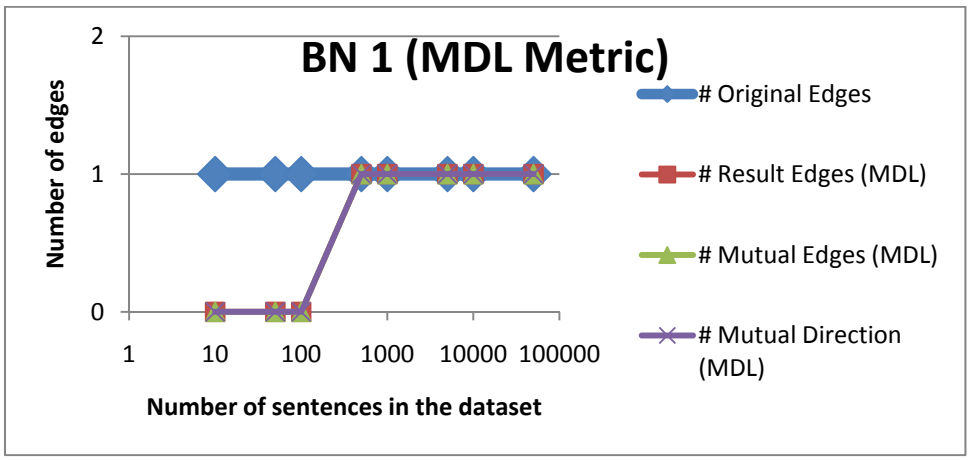


Figure 45 The result for BN 1 using MDL metric

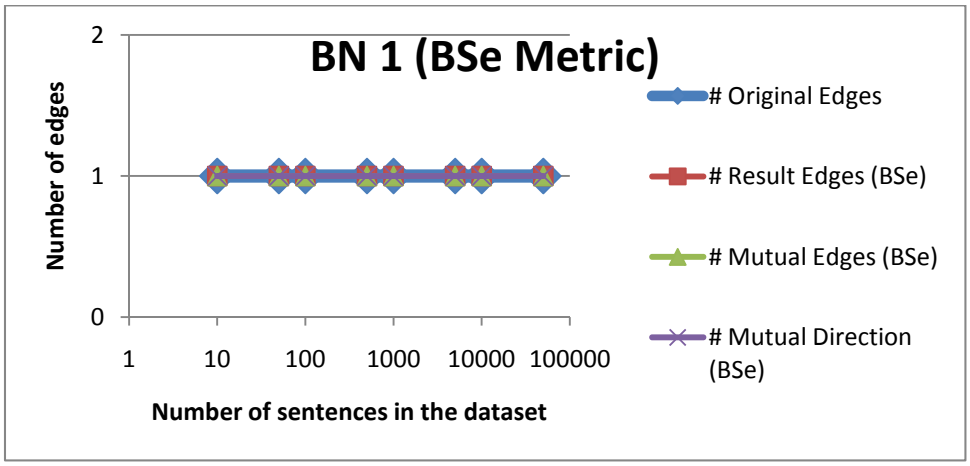


Figure 46 The result for BN 1 using BSe metric

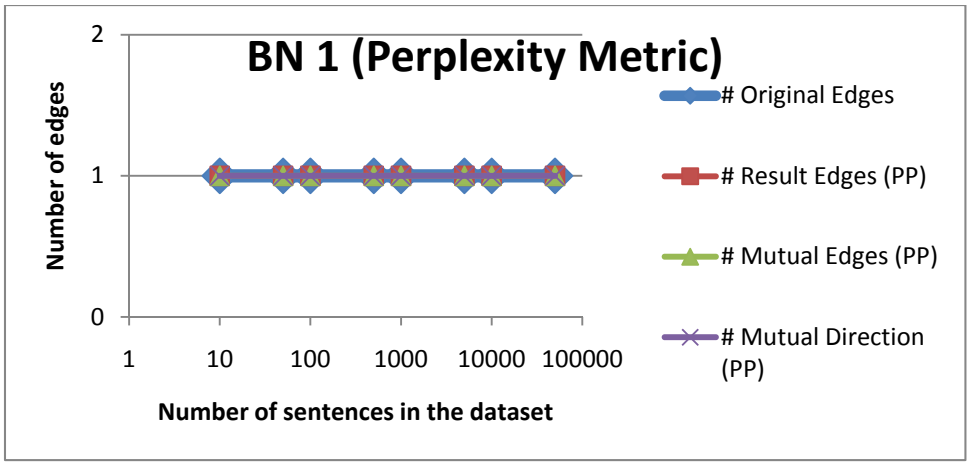


Figure 47 The result for BN 1 using Perplexity metric

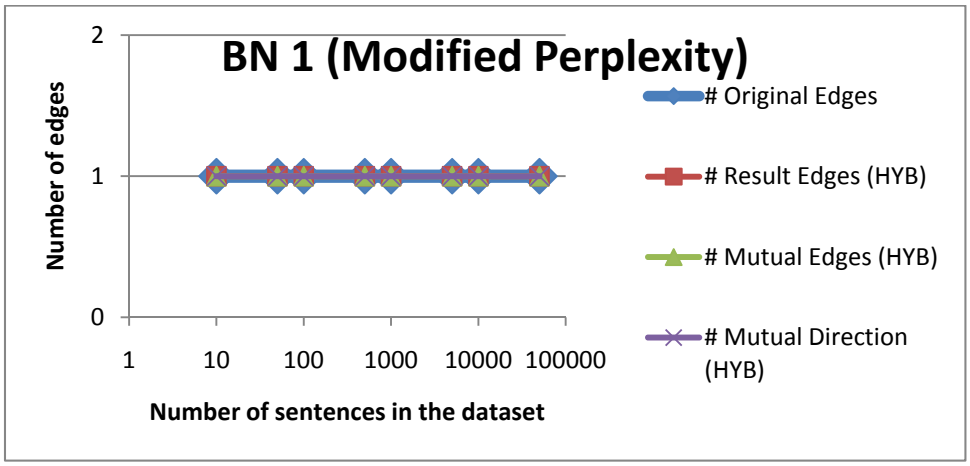


Figure 48 The result for BN 1 using Modified Perplexity metric

However, BN 1 is a trivial network. So, let's examine the results of BN 2 and BN 12 where there are more possible edges that can be connected. From the results of BN 2 in Figure 49, Figure 50, Figure 51 and Figure 52, the same trend can be seen. If the original edge is recognized, then the direction is also recognized. The results of BN 12 in Figure 53, Figure 54, Figure 55 and Figure 56 also perform slightly better with regard to the recognized directions. This improvement is a direct consequence of applying the constraints to the algorithms. The algorithm is now aware of BNs that should not be the end result of the algorithms. However, as can be seen in the result of BN 12 for the MDL metric, in a bigger network these constraints do not solve the problem of choosing the best network from networks that have the same scores. For 50000 sentences, the graph shows that only one edge has a correct direction, whereas for fewer sentences more edges with correct directions are recognized. This is because for other recognized edges, when their directions are inverted, it will give the same score. So, in this case the randomness still plays a role. However, because of the third constraint, this randomness is not essential anymore, because this randomness only occurs in non-word variables. Ergo, the resulting Bayesian network can still be applied for language modeling. The randomness in non-word variables is not essential because as stated in (Verma & Pearl, 1990), the parameters in the variables when the scores are equivalent will also be equivalent.

Another noticeable phenomenon in the result is how poor the performance of BSe metric when it comes to recognizing the direction of the edges compared to other metrics. Because BSe metric does not give same score to equivalent Bayesian Network, this means that BSe metric gives the highest optimal score to the Bayesian Network with the wrong edge directions.

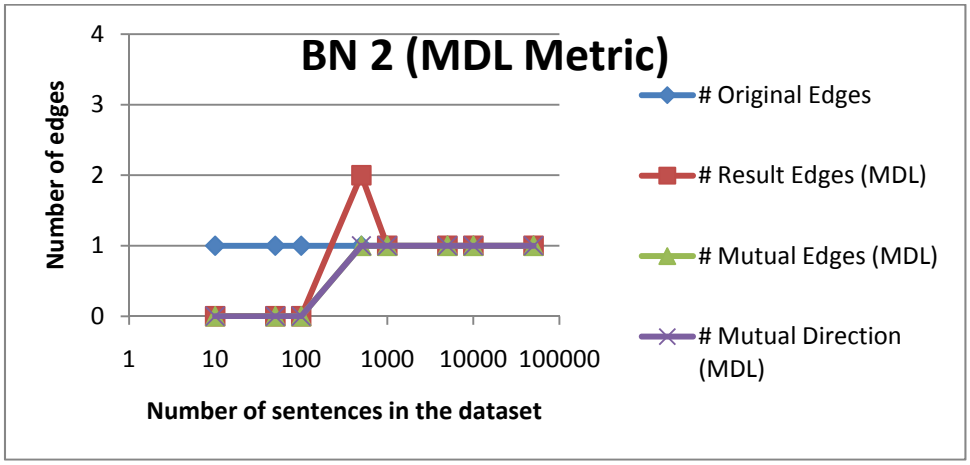


Figure 49 The result for BN 2 using MDL metric

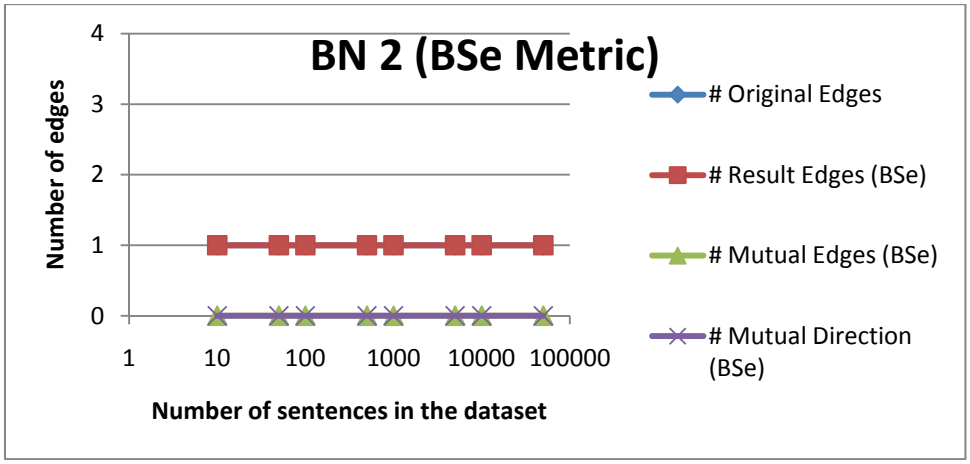


Figure 50 The result for BN 2 using BSe metric

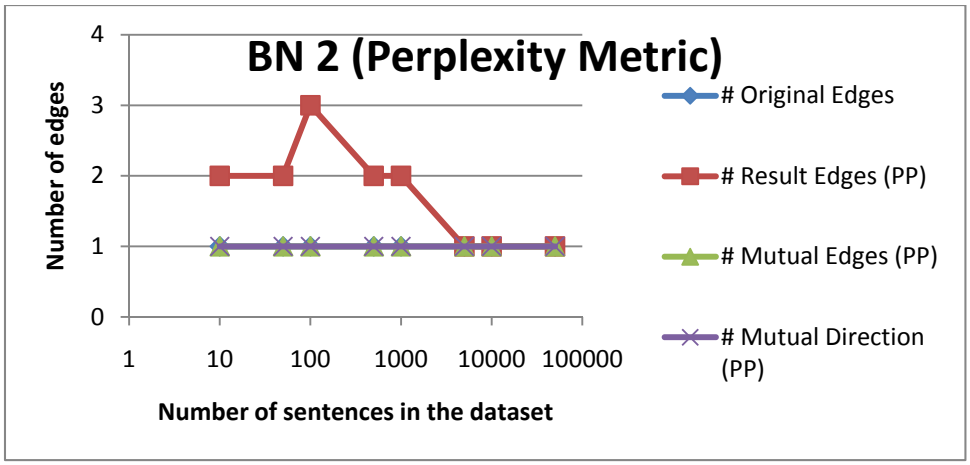


Figure 51 The result for BN 2 using Perplexity metric

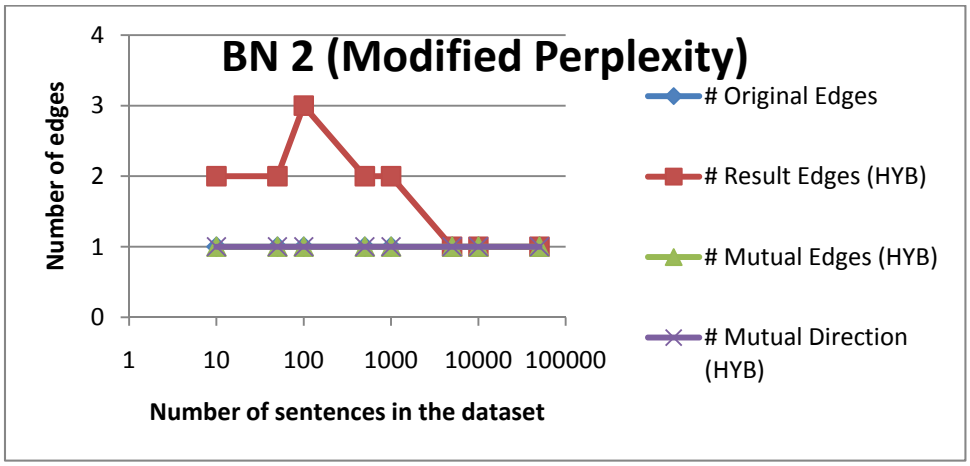


Figure 52 The result for BN 2 using Modified Perplexity metric

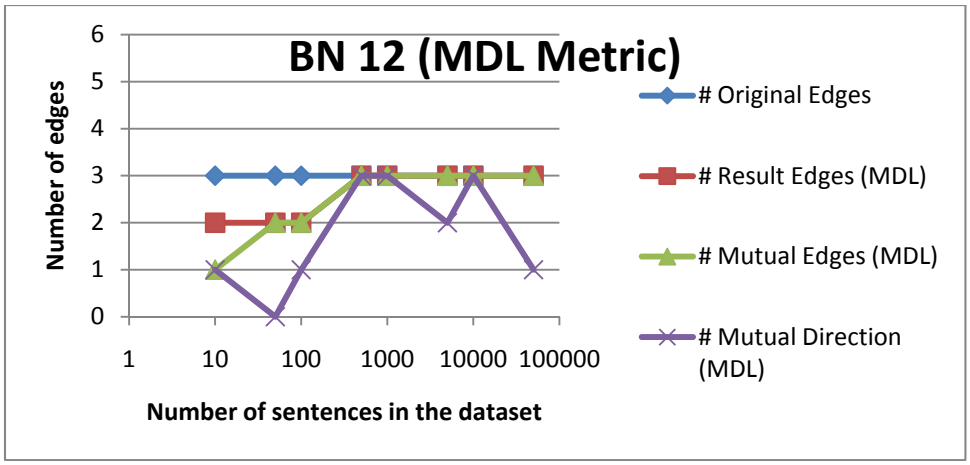


Figure 53 The result for BN 12 using MDL metric

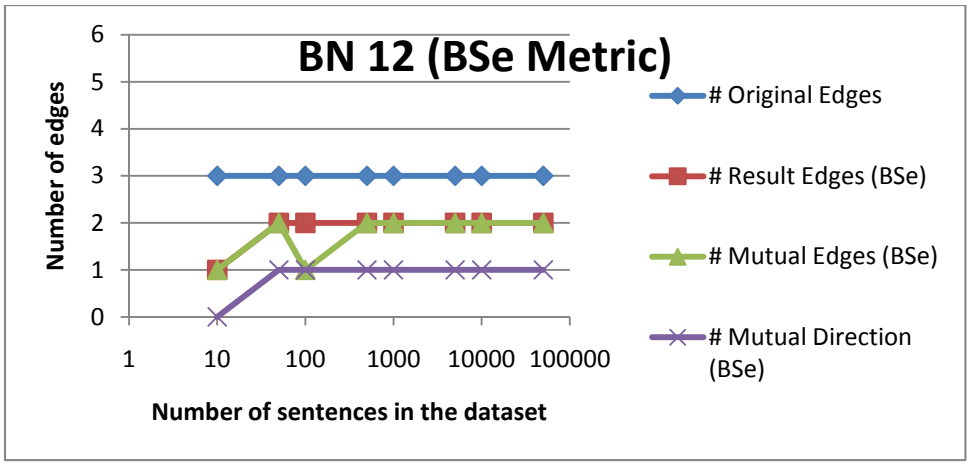


Figure 54 The result for BN 12 using BSe metric

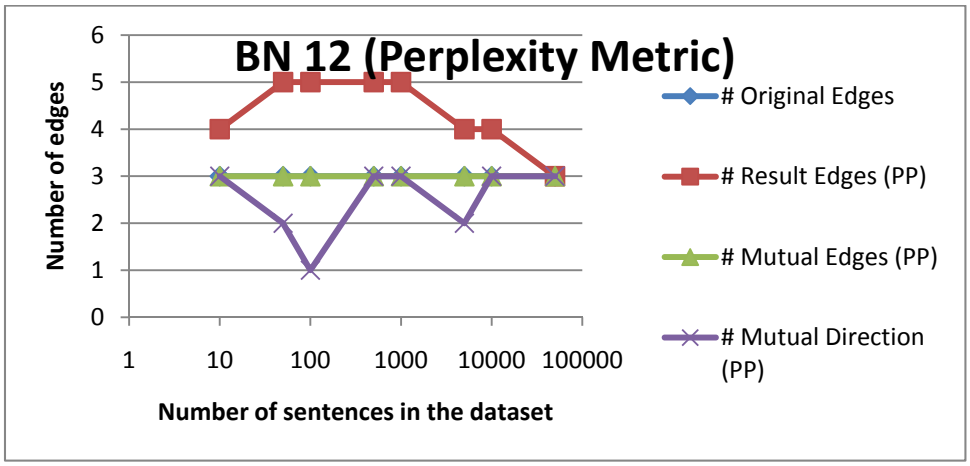


Figure 55 The result for BN 12 using Perplexity metric

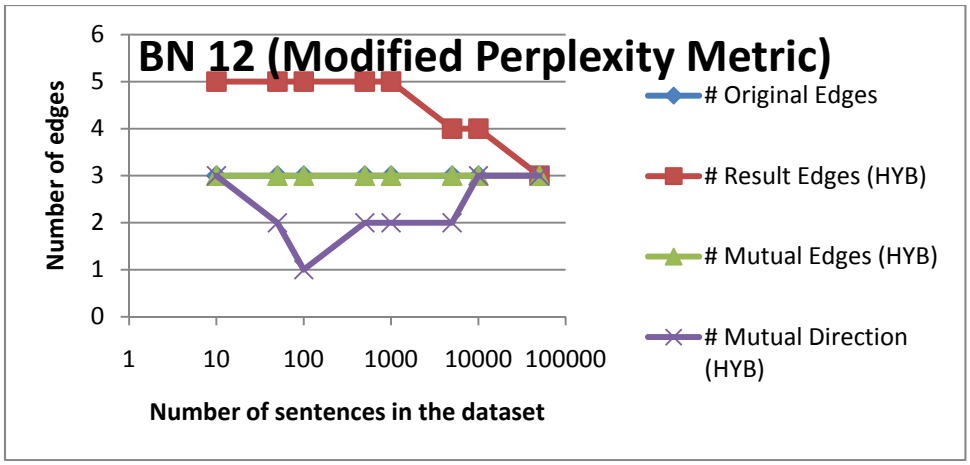


Figure 56 The result for BN 12 using Modified Perplexity metric

5.4.2 EXPERIMENT WITH DYNAMIC BAYESIAN NETWORK

Since two of the three constraints have to do with situations where temporal variables are in play, an artificial language that is based on a bigram is created. Creating an artificial language that is modeled by a bigram means that Dynamic Bayesian Network with 2 time slices is applied to model this language model. However, as stated in (Friedman, Murphy, & Russell, 1998) the model learning aspect itself does not change for Dynamic Bayesian Networks. As seen from the eyes of model learning, a transformation from unigram to bigram is a mere extension of Bayesian Network from the one with n variables to another with $2n$ variables.

Doubling the number of variables in a Bayesian Network also means that the number of possible state combinations will grow to the power of two. Based on the conclusion of first experiment, the effect on the result of model learning is similar to effect of decreasing number of sentence in dataset. Running experiments while varying the number of sentences is therefore not necessary to observe this effect.

Instead, a smaller but more evident experiment is run. In this experiment datasets for each bigram version of BN 2, BN 3 and BN 4 are created. Bigram version of BN 2, BN 3 and BN 4 are depicted in Figure 57, Figure 58 and Figure 59.

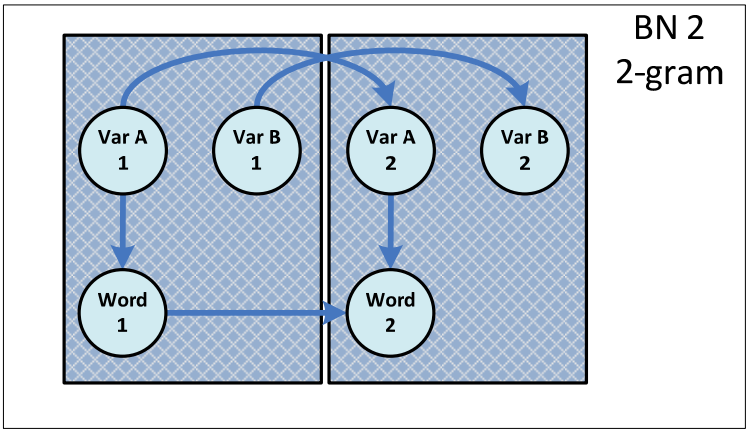


Figure 57 The bigram version of BN 2

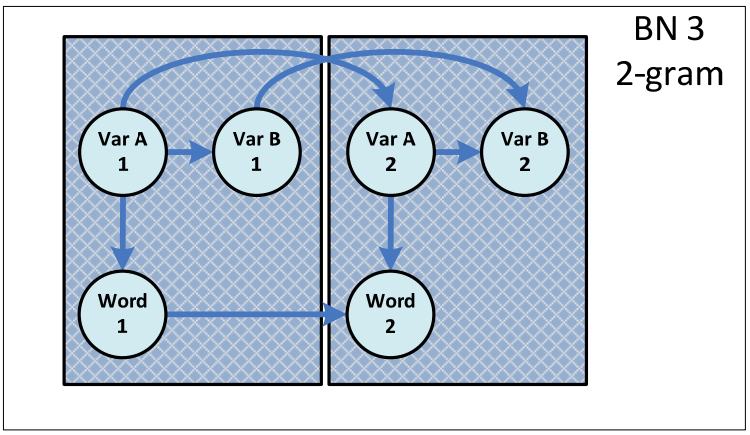


Figure 58 The bigram version of BN 3

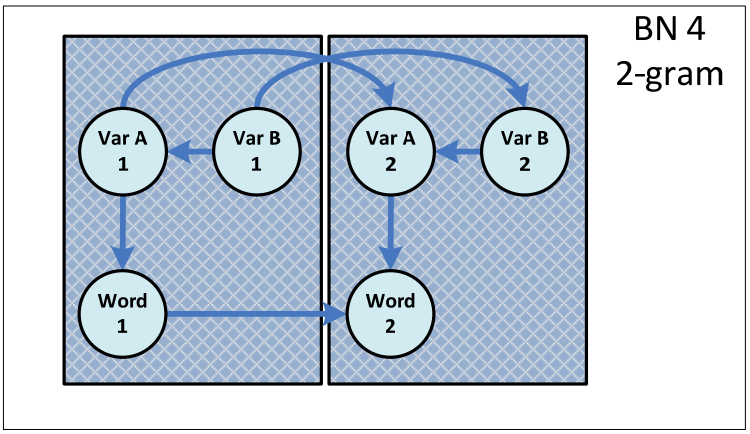


Figure 59 The bigram version of BN 4

Each dataset contains 20000 sentences and 6000 extra sentences for the test dataset. For each of these datasets, experiments are run for all metrics and using the Genetic Algorithm. The results of these experiments can be seen in Figure 60, Figure 61 and Figure 62.

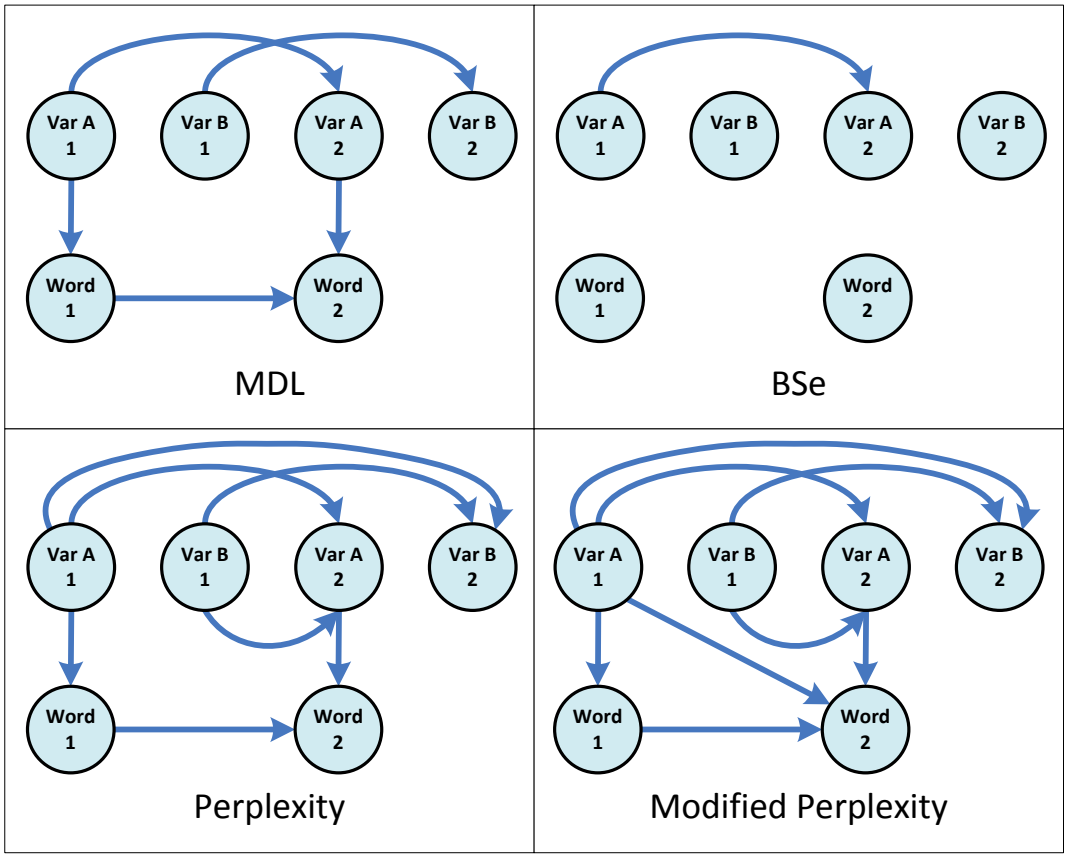


Figure 60 The result of learning the bigram version of BN 2

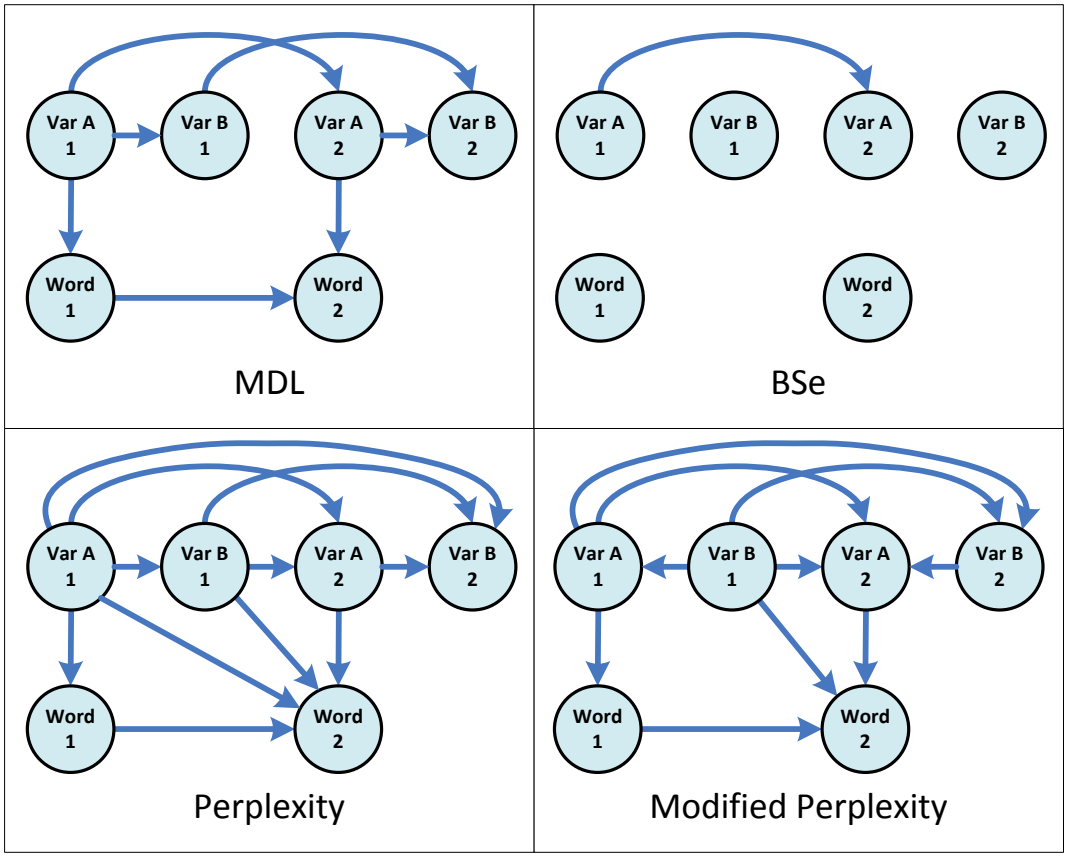


Figure 61 The result of learning the bigram version of BN 3

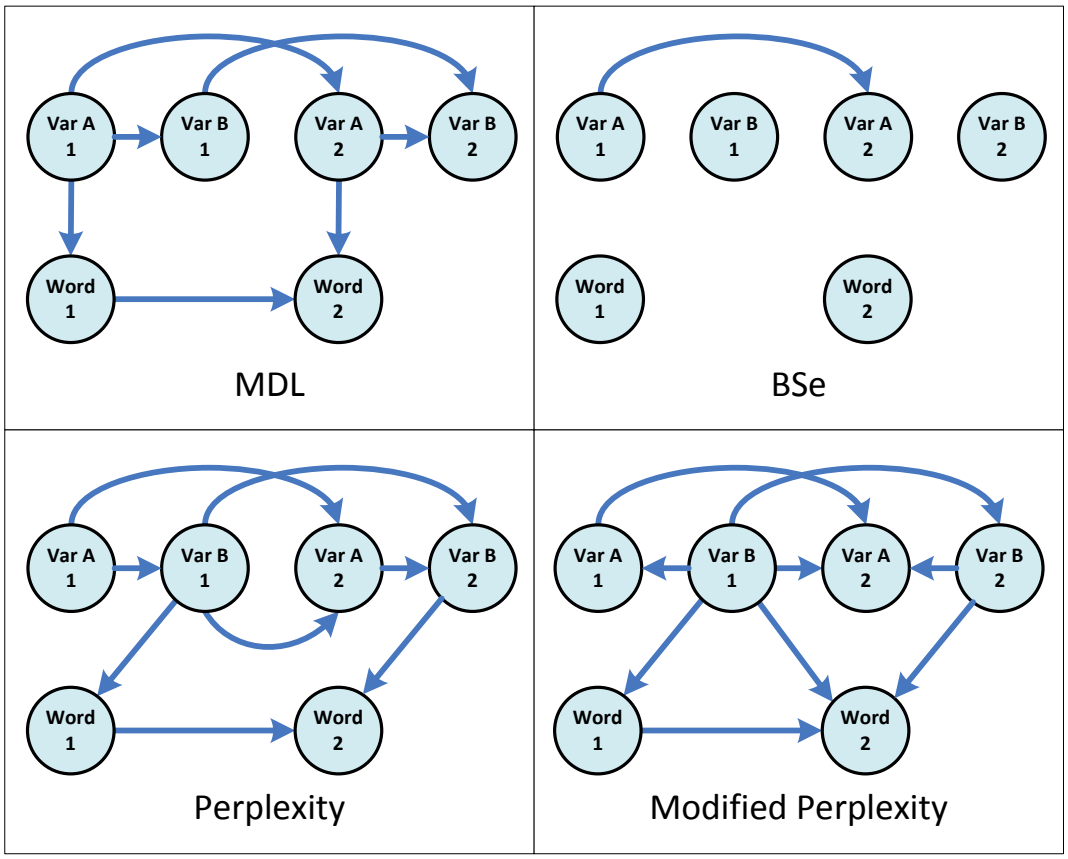


Figure 62 The result of learning the bigram version of BN 4

From the results, we can see that the MDL metric can learn all original dependencies and all directions except 2 edges in BN 4 correctly. This shows that even in very complex models that contain 6 variables the MDL metric can still give accurate result.

The BSe metric is very conservative when handling complex models. It only recognizes one dependency for all datasets and it omits the dependency between words. The latter makes the resulting Bayesian Network not very useful because it actually does not model the language model as a bigram. The reason why the BSe metric omits dependencies between words is the number of states of word variables which is always much larger than the number of states of other variables. The BSe metric punishes the increase in the number of states very hard for increasing the complexity of the Bayesian Network.

On the other side, the results of Perplexity and Modified Perplexity are always more complex than the original model. This is caused by the nature of Perplexity and Modified Perplexity that always look for a better fit, even when as a consequence the complexity of the Bayesian Network increases.

5.4.3 CONCLUSION

The result of experiment 2 shows that adding the constraints in the algorithm improve the resulting Bayesian network of the model learning. The direction of the edges in the resulting Bayesian network is better recognized. However, the improvement is less drastic for the results of BSe metrics.

When it comes to MDL metric and Perplexity metric that give the same score to equivalent Bayesian Network, the constraints do not eliminate the randomness of the direction in the result. This is however not a bad thing since the randomness can only occur in non-word variables, so it will still be applicable as a language model.

The result of the experiment with Dynamic Bayesian Network confirms the characteristics of each metric concluded from experiment 1. BSe Metric gives a very minimalistic Dynamic Bayesian Network, while Perplexity and Modified Perplexity Metric prefers a more complex Dynamic Bayesian Network compared to the original model. Of all metrics, MDL Metric performs the best by learning BN 2 and BN 3 correctly and recognizing all edges in BN 4 while having wrong direction for 2 edges in the result.

5.5 EXPERIMENT 3

Experiment 1 and 2 show by means of artificial language that it is possible to learn the language model from observational data. Now, the question remains: how the model learning techniques will perform when the data is real conversational data. In experiment 3, we will find the answer to this question.

5.5.1 DESIGN

In this experiment, the data from the Spoken Dutch Corpus project such as explained in Section 5.2.2 is used. From this data, 5 variables are chosen for this experiment. These are:

1. The Word variable: it contains all the words in the vocabulary. Only words that occur more than 10 times in the Spoken Dutch Corpus are used. After selecting only words that occur more than 10 times, the number of words comes down to 22188 words. This includes two dummy words that mark the beginning and the end of a sentence.
2. The Gender variable: this variable contains the gender of the speaker. This variable has two states: MALE and FEMALE.

3. The Language variable: this variable contains in which language the speaker is talking. In Spoken Dutch Corpus project, there are two languages: Dutch and Flemish. However, these two languages are related to each other, because Flemish can be seen as a variety of Dutch. Therefore there is a big overlap between words used in Dutch and in Flemish.
4. The Conversation Type variable: based on the nature of the conversation (e.g. lecture, telephone dialogues, discussion, etc), whether it's scripted or not, whether it is broadcasted (via tv or radio) or not and its domain (private or public), 21 types of conversations are identified.
5. The Education Level variable: the states of this variable describe the education level of the speaker. There are 4 states for this variable: LOW, MEDIUM, HIGH and UNKNOWN.

From experiment 1 and experiment 2, it can be concluded that model learning with MDL metric comes the closest to the original underlying model. It is also showed that model learning with MDL metric when dealing with data insufficiency doesn't always come up with model that is either too trivial or too complex. Since data sufficiency is always the case when learning the model by means of real conversational data, MDL metric is chosen for this experiment.

Experiment 1 also shows that when it comes to efficiency and optimality, neither Genetic Algorithm nor Particle Swarm Optimization is more superior from another. For experiment 3, Genetic Algorithm is chosen for a practical reason. One iteration in Genetic Algorithm takes fewer operations than in Particle Swarm Optimization which means that it takes less time to compute one iteration in Genetic Algorithm. By taking a snapshot of the state of the algorithm in a iteration, it is possible to stop the algorithm in the middle of the process when it's necessary and pick it up where it left off when the algorithm is run again. When computation time of one iteration is shorter, stopping the algorithm can be done more flexibly.

5.5.2 ANALYSIS

Having chosen MDL and GA as respectively, metric and algorithm of the model learning, we run the experiment to find the most optimal language model for a real conversational data. The resulting language model of this experiment can be seen in Figure 63.

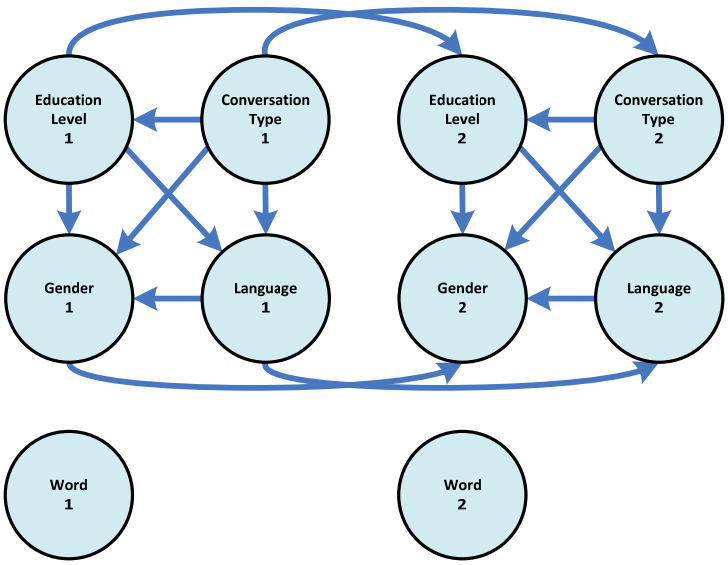


Figure 63 The resulting language model (First trial)

As we notice from Figure 63, the resulting language model does not see any role of non-word variables in influencing the choice of words in a sentence. Furthermore, it doesn't even incorporate bigram model into this

language model. The resulting language model claims that the previous word doesn't influence the choice of the next word. This resulting language model is obviously of no worth for Automated Speech Recognition systems since the Word variable does not make use of the information and knowledge from other variables.

The reason why the model learning comes up with a language model that isolates the word variables from other variables is caused by the trade-off between complexity and data fit in MDL metric. Putting an edge between the word variable with any other variable is punished with a high complexity score because the word variable has a high number of states. When a variable has a high number of states, the number of parameters in conditional probability table will also be extensive. The number of parameters is the main factor in the complexity score of MDL metric. On the other side, the data fit gained doesn't match the high complexity score. This is why the most optimal language model according to MDL metric doesn't have an edge that connects the word variable with other variables.

To accomplish a more useful model, one more constraint is added to the algorithm. This constraint obliges all language models that the algorithm explores to incorporate N-gram model. This means that each word variable in the model must be a parent of other word variables in the following $N-1$ time slices. Having this constraint added into the algorithm, similar experiment is run. The resulting language model is depicted in Figure 64.

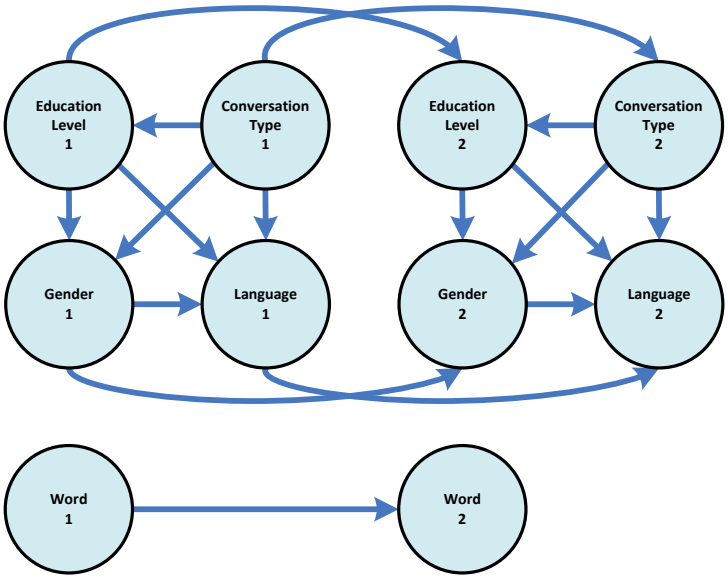


Figure 64 The resulting language model (Second trial)

As expected, the resulting language model incorporates N-gram model. This is a slight improvement compared to the result from the previous result without N-gram constraint. However, because of the high complexity cost explained above, the non word variables still omit connection with the word variable. Notice that compared to the previous result without N-gram constraint, when word variables are removed both remaining models are equivalent. This shows that when it comes to the structure of the non-words variables, this structure is the most optimal.

Compared to a plain bigram model, the resulting language model with added n-gram constraint does not add anything valuable to improve ASR systems. In order to come up with a better model learning result, the non-word variables and word variables should not be separated from each other. To enforce this, another constraint is added. This constraint obliges all models that the algorithm explores to have at least one connection between word variable and non-word variables. This is implemented in the algorithm by picking a random pair of a word variable and a non-word variable in the same time slice and connects them by putting

an edge that is pointing to the word variable. However, this is only done when there is not even one edge that connects word variable and non-word variables. After adding this constraint, another experiment is run. The resulting language model after adding another constraint can be seen in Figure 65.

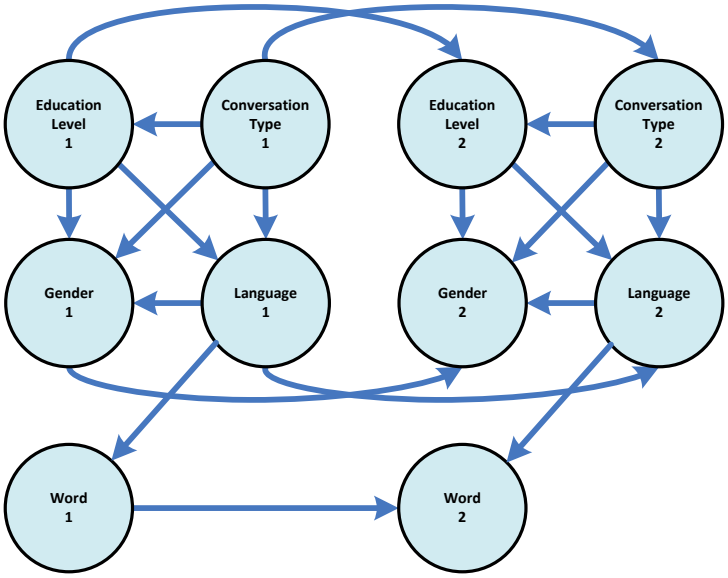


Figure 65 The resulting language model (Third trial)

The resulting language model this time connects the word variables and non-word variables by making the word variables the parents of the Language variable. The connection between the word variables and non-word variables is minimal because adding another edge between the word variable and other non-word variable will increase the complexity of the model. Even though the connection is minimal, this language model is better in predicting choice of words than a plain bigram model, since the information and knowledge from non-word variables are put in good use now. This is confirmed by the Perplexity score (Wiggers, 2008) of the bigram model and the resulting language model. Calculation of the perplexity score of the bigram model results in 179.427, while the perplexity score of the resulting language model gives 43.0703 as a result. As explained in Section 4.1.3, the lower the perplexity score is, the better the language model is. This means that the resulting language model is a better language model compared to the bigram model.

Before we go to the conclusion of this experiment, there is still one remark about the speed of running the experiment 3 to be made. The experiment using real conversational data takes a lot of computation time even after a lot of optimization in the implementation is done to speed it up. The large amount of computation time it takes is caused by the following reasons:

- The dataset is large. This causes querying the dataset to slow down.
- The word variable has a large number of states. This causes the calculation of the metric to take longer because it has to examine every possible combination of states in the model.
- The model contains a large number of variables. This causes the number of iterations that the algorithm takes before it comes to the most optimal model to increase.

5.5.3 CONCLUSION

The experiment 3 shows that the constraints added in experiment 2 to improve the performance of model learning do not suffice. As shown by the first result of experiment 3, the word variables are left isolated in the resulting model. This is caused by the number of states that the word variables have. Making a word variable a child of other variable will increase the complexity cost in the metric score and thus according to the metric this

model is less optimal. This is a consequence of the inherent property of the MDL metric since its score is a sum up of complexity cost and the data fit. In a dataset that is insufficient and when there are a large number of states in a variable, the score of the MDL metric will be dominated by complexity cost. So, it is also can be concluded that the result of the model learning with MDL metric in this case is simpler than how the original model should be.

In order to learn a better model, two additional constraints are added:

1. The model that the algorithm explores must incorporate N-gram model
2. The model that the algorithm explores must have at least one edge between non word variable and word variable.

By adding these constraints, the resulting model becomes more useful and better compared to plain N-gram model that only consist of word variables. This is also shown by the perplexity score of the resulting model.

However, the bottleneck in model learning is the computation time. Even though, in the implementation of experiment 3 a lot of work is done to speed up the computation time, it still takes a long time before the model learning comes up with the resulting model. This is caused by the large dataset, the large number of states in word variable and a large number of variables in the model.

6 CONCLUSION & FUTURE WORK

6.1 EVALUATION OF MODEL LEARNING

The goal of this thesis is to develop mechanism in learning the Dynamic Bayesian Network model for Automated Speech Recognition system. From the literature, it appears that learning a Dynamic Bayesian Network is no different than learning a Bayesian Network with a doubled amount of variables. In this thesis, the model learning mechanism is decomposed into two components. These are metric and search algorithm. A metric is the quantitative measure of how optimal the model is. Having this metric makes it possible to compare one model to another. There is however a large number of possible models which makes it more difficult to find the most optimal model. The number of possible models increases exponentially as the number of variables in the model also increases. This is where the search algorithm steps in. The search algorithm defines the process of finding the most optimal models. To avoid exploring all possible models, the search algorithms in this thesis are adapted from local search algorithms.

With focus on language model, a list of metrics that can be applied for model learning is set up. These are

- MDL metric
- BSe metric
- Perplexity metric
- Modified Perplexity metric.

In this thesis, the details of implementation of each metric are explained thoroughly. The operation that is performed frequently when calculating the metric is querying database. When the database is large, querying database will take a lot of computation time. In this thesis, it is also discussed how to implement the database and how its queried in such way that it can speed up the whole metric calculation.

Each of these metrics has its own characteristic when it is applied to learn the most optimal model. The characteristics are concerning the behavior when it's in the following situation:

- How does it score two unidentical equivalent Bayesian Network? (The equivalence of Bayesian Network is explained in (Verma & Pearl, 1990))
- What kind of network does it learn when it has to deal with data insufficiency?

The MDL metric and the Perplexity metric give an equal score to two unidentical equivalent Bayesian Network, while the BSe metric and Modified Perplexity metric give a different score. When two unidentical equivalent Bayesian Networks get the same score, it means that there may be randomness in the direction of the edges in the resulting Bayesian Network.

When the data insufficiency occurs, The MDL metric and the BSe metric will learn a model that is minimal compared to the original model. In other words, the resulting Bayesian Network for the MDL metric and BSe metric will contain fewer edges compared to the original model. On the contrary, the Perplexity metric and Modified Perplexity metric gives a result that is more complex than the original model when it has to deal with data insufficiency. How the metric deals with data insufficiency is essential in interpreting the result of the model learning especially for language model. This is because it is infeasible to collect sufficient data for language model since there are a large number of variables and each variable has a large number of states.

From the experiments, it appears that the model learning using the MDL metric perform better than other metrics. The result using the MDL metric is never too complex compared to the original model and most edges in the model are recognized. The MDL metric also performs well when the real conversational dataset is used. With help of the constraints taken from prior knowledge, model learning with the MDL metric produces a

model that is better than a plain N-gram model. However, in the score of MDL metric the complexity cost is dominant compared to the gain from the data fit. This is caused by the large number of states in the variables of the language model. This creates an imbalance between complexity and the data fit of the model. The consequence is that the result of model learning using the MDL metric in this case becomes more simple than the original model is.

Furthermore, two algorithms are implemented in this thesis as the choices of search algorithms, namely Genetic Algorithm and Particle Swarm Optimization. The most important aspect in implementing these algorithms is how the Bayesian Network is encoded in this algorithm. Bayesian Network requires the structure of the network to be directed and acyclic. The encoding should then cope with the directivity and acyclicity of the Bayesian network. In Chapter 4, these encodings and other implementation details are explained.

The experiments in this thesis show that both algorithms can find the most optimal model. Both algorithms differ in how many iteration it takes to the most optimal model. Genetic Algorithm takes more iterations than Particle Swarm Optimization to come up with the most optimal model. However, it can not be concluded that Particle Swarm Optimization is more efficient than Genetic Algorithm, since Particle Swarm Optimization explores more models in one iteration than Genetic Algorithm. Ergo, the choice of the search algorithm is arbitrary since it will not influence the performance of the model learning as a whole.

Through the experiments, we also find out that by picking the most suitable metric and search algorithm is not enough to learn the language model. In this thesis, a list of constraints is incorporated into the search algorithm in order to add prior knowledge into the whole model learning mechanism. These constraints are:

1. Future events can not influence present outcomes. This implies that the variable at timestep t can not cause any variable at timestep $t-1$ or lower.
2. The process is stationary. This implies that A dependency in a timestep t is also valid in other timesteps.
3. Word variables can not cause non-word variables
4. The model must incorporate N-gram model
5. The model must have at least one edge between non word variable and word variable.

These constraints are imposed to all models that are explored by the search algorithm. By incorporating these constraints, the number of possible models that are needed to be explored is reduced. These constraints also help the model learning to come up with a useful model that can be applied as a language model. For example, it is not useful to come up with a model whose word variable is isolated from other variables; it is also not useful to have a model whose variable take variable from subsequent timeslice as its parent; etc. In the experiments, we also see an improvement in the direction of the edges that are recognized correctly because of these constraints.

The bottleneck of the model learning to find the language model for ASR is the computation time. Even after a optimizing the calculation of the metric, it still take a long time to come up with the most optimal model. This is caused by the following reasons:

- The dataset for language model is large
- One or more variables in the model have a large number of states
- There are a lot of variables in the model.

It is expected that if the model extends or a larger dataset is used, then the computation time will also increase exponentially.

6.2 FUTURE WORK

Even though the experiments show that model learning techniques can come up with a sensible language model, from the evaluation of the model learning it can also be concluded that there is still room for improvements in the model learning techniques. The recommended future works in this chapter will range from improving the optimality of model learning to incorporating more linguistic knowledge in the model learning.

In this thesis, the focus is on learning the language model for Automated Speech Recognition systems. However, an Automated Speech Recognition system also needs the acoustic model. The biggest difference between the language model and the acoustic model is on the composition of the variables. The variables in acoustic model are a mix of discrete and continuous variables whereas a language model is most likely to be composed of discrete variables only. There are at least two options in tackling this problem. The first is assuming that all variables except the acoustic feature variable are discrete variables; the model for discrete variables can be learnt in a similar way as the language model is learnt. Figure 66 shows how the acoustic feature variables are combined with other discrete variables. In Figure 66, a discrete variable Mixture which regulates the weights in a mixture of Gaussians is the connector between other discrete variables and the acoustic feature variable. The mixture variable is a hidden variable. To learn the model of a Bayesian network with a hidden variable, modifications in the techniques are necessary. These modifications are discussed in (Friedman, 1997) and (Chickering & Heckerman, 1997).

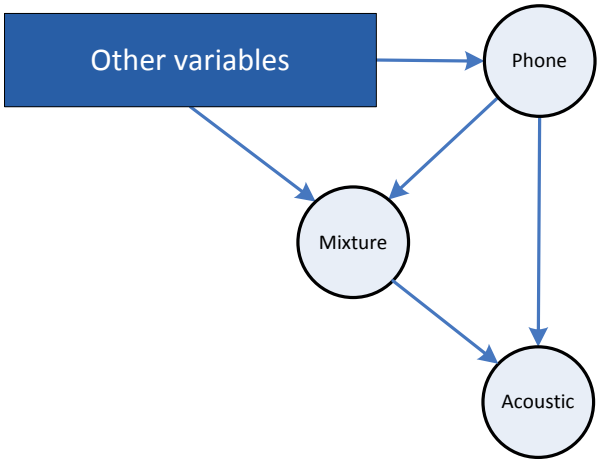


Figure 66 The acoustic model with mixture of Gaussian as the distribution of acoustical features

Instead of decomposing the acoustic model into a discrete variables' model and a continuous variables' model, a new metric that handles a mix of discrete and continuous variables can be used. However, literature that discusses such metrics is scarce. When such metric is found, learning the acoustic model differs from learning the language model only in applied metric.

Research that focuses on acoustic models will answer the question whether the abovementioned two options are feasible. It will also reveal the difficulties in doing so and how effective the model learning techniques for acoustic models will be.

In the last experiment we see that the result of model learning does not have many edges in its model. This is caused by how the MDL metric is defined. The MDL metric is a trade-off between complexity and data-fit. However, in the field of Speech Recognition where each variable can have a large amount of states, this trade-off is not quite fair. The price of going to a more complex model is very high whilst the gain in data-fit, even though it improves a lot, is quite small compared to the price of complexity.

As we have discussed in Chapter 3, the number of states in Speech Recognition is inherently high. This causes the number of parameters to rise exponentially as the number of edges in the Bayesian Network increases. Since the punishment for complexity in MDL becomes higher as the number of parameters increases and the gain in data-fit does not increase in the same magnitude, the complexity in MDL will overpower the data-fit.

To deal with this imbalance, further research in compressing the parameters and the structure of Bayesian network is needed. It implies that without the decrease or with a minimal decrease (desirably negligible decrease) of data-fit, the amount of parameters and the representation of structure are condensed. In (Friedman & Goldzsmidt, 1996), the authors propose a modification of the MDL score by learning the local structure in a conditional probability table. The essence of this idea is grouping the parameters in the conditional probability table that are equivalent to each other into one parameter. In this manner, the number of parameters can be restricted without compromising in the data-fit. The grouping can be performed with either of two compression techniques: default tables or decision trees.

By applying this modified MDL score, the weight in complexity in the MDL score can be reduced. It is then expected that the balance between complexity and data-fit can be restored. To confirm this, experiments to test the effect of modified MDL score are necessary.

Furthermore, it is also an inherent property of model learning for speech recognition that how large the dataset may be, it will still be a sparse dataset. When learning conventional language models such as bigram and trigram, smoothing techniques are applied to make the distribution in the dataset more uniform. There are various kinds of smoothing techniques, namely, discounting techniques, interpolation and back-off (Jurafsky & Martin, 2009). By taking lessons from these smoothing techniques, similar techniques can be incorporated in model learning techniques to get around the problem of sparse dataset.

Besides the needed improvements in the model learning techniques itself, the model learning techniques also still has to deal with an extensive amount of calculation time. This is caused by the calculation of the metric that necessitates a massive amount of search queries in the dataset. It takes even more time when the dataset itself is large. To make model learning techniques for speech recognition more feasible, finding other approaches to speed up the model learning techniques is essential.

Since the model learning techniques procedurally are dominated by search queries, speeding up search queries in the dataset can make it much faster. A few suggestions in order to achieve this:

- Because the algorithm in model learning is based on comparing one model to another, it is not necessary for the search query to give an exact number as an answer as long as the answer gives the correct proportion of the dataset. Instead of searching the whole dataset to come up with the answer of the search query, the answer can be found by doing sampling in the dataset
- In the field of speech recognition, it is plausible to assume that there are more possible combinations of states than the amount of data in the dataset. By modifying the implementation of the search query in order to take account this assumption into account (e.g. other data representation, more indexing, etc), the result can be found faster.

Besides making the search query faster, speeding up the model learning techniques can be done by making the algorithms in model learning more efficient. This can be done by incorporating prior knowledge in the algorithm. The prior knowledge tells the algorithm what kind of models is expected in the result. This makes the algorithm aware of the models that are less probable so that the algorithm will put more priority in scrutinizing the models that are more probable. This prior knowledge can be elicited from experts or other sources of knowledge. In Section 2.3.3, examples of how prior knowledge can be incorporated in model learning are discussed.

6.3 CLOSING

In this thesis, we show that the learning language model using the observational data is possible. To complete the ASR system, the acoustic model is also needed to be learned. How the acoustic model is learned is however not different than the mechanism that we have proposed. When the mechanism to learn the acoustic model is also outlined by other research, it is then possible to come up with a model for ASR system automatically with the observation data as the input.

When this is possible, it is not necessary anymore for designers of ASR system to do a special research to find the most optimal model manually. Instead of finding the most optimal model, the only things that the designers of ASR system have to do are picking the variables in the model and collecting the data for those variables. Since the designer needs to elicit knowledge from experts and translate it to a valid model when designing the model manually, this takes a lot of time. By modeling it automatically, it will speed up the implementation of ASR system, especially when there are a large number of variables in the model and a certain model is needed for a special domain.

However, even though learning the model of ASR system automatically is quicker compared to designing the model of ASR system manually, the experiments in this thesis show that learning the model automatically is a compute-intensive operation. This means that it will take a lot of computation time to learn a model. As the model extends, the computation time will increase. In order to make model learning more useful, reducing the computation time without influencing the resulting model is required.

7 REFERENCES

Bilmes, J., & Bartels, C. (2005). Graphical Model Architectures for Speech Recognition. *IEEE Signal Processing Magazine , 22* (5), 89- 100.

Bøttcher, S. G. (2004). *Learning Bayesian Networks with Mixed Variables.* Aalborg: Aalborg University.

Buntine, W. (1991). Theory Refinement on Bayesian Network. *Proceedings of the seventh conference (1991) on Uncertainty in artificial intelligence* (pp. 52-60). Los Angeles: Morgan Kaufmann.

Chickering, D. M., & Heckerman, D. (1997). Efficient Approximations for the Marginal Likelihood of Bayesian Networks with Hidden Variables. *Machine Learning* , 181-212.

Chris, W. S., & Freeman, P. R. (1987). Estimation and inference by compact coding. *Journal of the Royal Statistical Society. Series B (Methodological) , 49* (3), 240-265.

Cooper, G. F., & Herskovits, E. (1992). A Bayesian Method for the Induction of Probabilistic Networks from Data. *Machine Learning , 9* (4), 309--347.

Daoudi, K., Fohr, D., & Antoine, C. (2003). Dynamic Bayesian networks for multi-band automatic speech recognition. *New Computational Paradigms for Acoustic Modeling in Speech Recognition , 17* (2-3), 263-285.

Davis, K. H., Biddulph, R., & Balashek, S. (1952). Automatic Recognition of Spoken Digits. *The Journal of the Acoustical Society of America* , 637-642.

Dawes, B. (2009, November 3). *Boost Libraries*. Taken from Boost: www.boost.org/doc/libs/release/libs/libraries.htm

DeGroot, M. H. (1970). *Optimal Statistical Decisions.* New York: McGraw-Hill.

Du, T., Zhang, S. S., & Wang, Z. (2005). Efficient Learning Bayesian Networks Using PSO. In *Computational Intelligence and Security* (pp. 151-156). Berlin: Springer Berlin / Heidelberg.

Dutch Language Union. (2001, April 1). *Het Corpus Gesproken Nederlands*. Taken from Het Corpus Gesproken Nederlands: http://lands.let.ru.nl/cgn/ehome.htm

Engelbrecht, A. P. (2007). *Computational Intelligence: An Introduction.* Chichester: Wiley InterScience.

Forsberg, M. (2003). *Why is Speech Recognition Difficult?* Chalmers: Chalmers University of Technology.

Friedman, N. (1997). Learning Belief Networks in the Presence of Missing Values and Hidden Variables. *Proceedings of the Fourteenth International Conference on Machine Learning* (pp. 125-133). San Fransisco: Morgan Kaufmann Publishers, Inc.

Friedman, N., & Goldzsmidt, M. (1996). Learning Bayesian Networks with Local Structure. In M. I. Jordan, *Learning in Graphical Models* (pp. 252--262). Cambridge: MIT Press.

Friedman, N., Murphy, K., & Russell, S. (1998). Learning the Structure of Dynamic Probabilistic Network. *Uncertainty in Artificial Intelligence*, (pp. 139-147). Madison.

Geiger, D., & Heckerman, D. (1994). *Learning Gaussian Networks.* Redmond: Morgan Kaufmann Publishers.

Gillispie, S. B., & Perlman, M. D. (2001). Enumerating Markov Equivalence Classes of Acyclic Digraph. *Uncertainty in Artificial Intelligence* (pp. 171-177). San Francisco: Morgan Kaufmann.

Heckerman, D., Geiger, D., & Chickering, D. M. (1995). Learning Bayesian Networks: The Combination of Knowledge and Statistical Data. *Machine Learning , 20* (3), 197-243.

John, P. F., Fong, C., Wong, S. H., Spitz, J. R., & Leung, H. C. (1995). PhoneBook: A Phonetically-Rich Isolated Word Telephone-Speech Database. *International Conference on Acoustics, Speech, and Signal Processing* (pp. 101-104). Detroit: IEEE.

Johnson, D. B. (1975). Finding All the Elementary Circuits of a Directed Graph. *SIAM Journal on Computing* , 77-84.

Juang, B. H., & Rabiner, L. R. (2005). Automated Speech Recognition - A Brief History of the Technology Development. In K. Brown, *Encyclopedia of Language and Linguistics* (pp. 806-819). Elsevier.

Jurafsky, D., & Martin, J. H. (2009). *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition.* New Jersey: Pearson Education, Inc.

Kennedy, J., & Eberhart, R. (1995). Particle Swarm Optimization. *IEEE International Conference on Neural Networks* (pp. 1942-1948). Perth: IEEE.

Korb, K. B. (2004). *Bayesian Artificial Intelligence.* Boca Raton: Chapman and Hall/CRC.

Lam, W., & Bacchus, F. (1994). Learning Bayesian Belief Networks: An Approach Based on the MDL Principle. *Computational Intelligence , 3*, 269-293.

Larrañaga, P., Kuijpers, C. M., Murga, R. H., & Yurramendi, Y. (1996). Learning Bayesian Network Structures by Searching For the Best Ordering With Genetic Algorithms. *IEEE Transactions on Systems, Man and Cybernetics* , 487-493.

Murphy, K. P. (2002). *Dynamic Bayesian Networks: Representation, Inference and Learning.* Berkeley: University of California.

Neiland, J. R., & Korb, K. B. (1999). The Evolution of Causal Models: A Comparison of Bayesian Metrics and Structure Priors. *Lecture Notes in Computer Science , 1574*, 433-438.

Rabiner, L. R. (1989). A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. *Proceedings of the IEEE* (pp. 257-286). IEEE.

Reeves, B., & Nass, C. (1996). *The Media Equation: How People Treat Computers, Television, and New Media Like Real People and Places.* Chicago: Center for the Study of Language and Inf.

Rissanen, J. (1978). Modeling By Shortest Data Description. *Automatica , 14*, 465-471.

Robinson, R. W. (1977). *Combinatorial Mathematics V.* Berlin: Springer-Verlag.

Sakoe, H., & Chiba, S. (1978, February). Dynamic programming algorithm optimization for spoken word recognition. *IEEE Transactions on Acoustics, Speech and Signal Processing* , 43-49.

Spirtes, P., Glymour, C., & Scheines, R. (2001). *Causation, Prediction, and Search.* The MIT Press.

Suzuki, J. (1999). Learning Bayesian belief networks based on the minimum description length principle. *IEICE Trans Inf Syst (Inst Electron Inf Commun Eng) , E82-D* (2), 356-367.

Tarjan, R. (1973). Enumeration of the Elementary Circuits of a Directed Graph. *SIAM Journal on Computing* , 211-216.

Ugur, A., & Thompson, H. (2006). The p-sized partitioning algorithm for fast computation of factorials of numbers. *The Journal of Supercomputing* , 73-82.

Verma, T., & Pearl, J. (1990). Equivalence and synthesis of causal models. *Proceedings of the Sixth Annual Conference on Uncertainty in Artificial Intelligence* (pp. 255 - 270). New York: Elsevier Science Inc.

Wallace, C. S., & Boulton, D. M. (1968). An Information Measure for Classification. *The Computer Journal , 11* (2), 185-194.

Wallace, C. S., & Korb, K. B. (1999). Learning Linear Causal Models by MML Sampling. *Causal Models and Intelligent Data Management* , 89-111.

Wang, T., & Yang, J. (2009). A heuristic method for learning Bayesian networks using discrete particle swarm optimization. *Knowledge and Information Systems* , 269-281.

Wiggers, P. (2008). *Modelling Context in Automatic Speech Recognition.* Delft: Delft University of Technology.

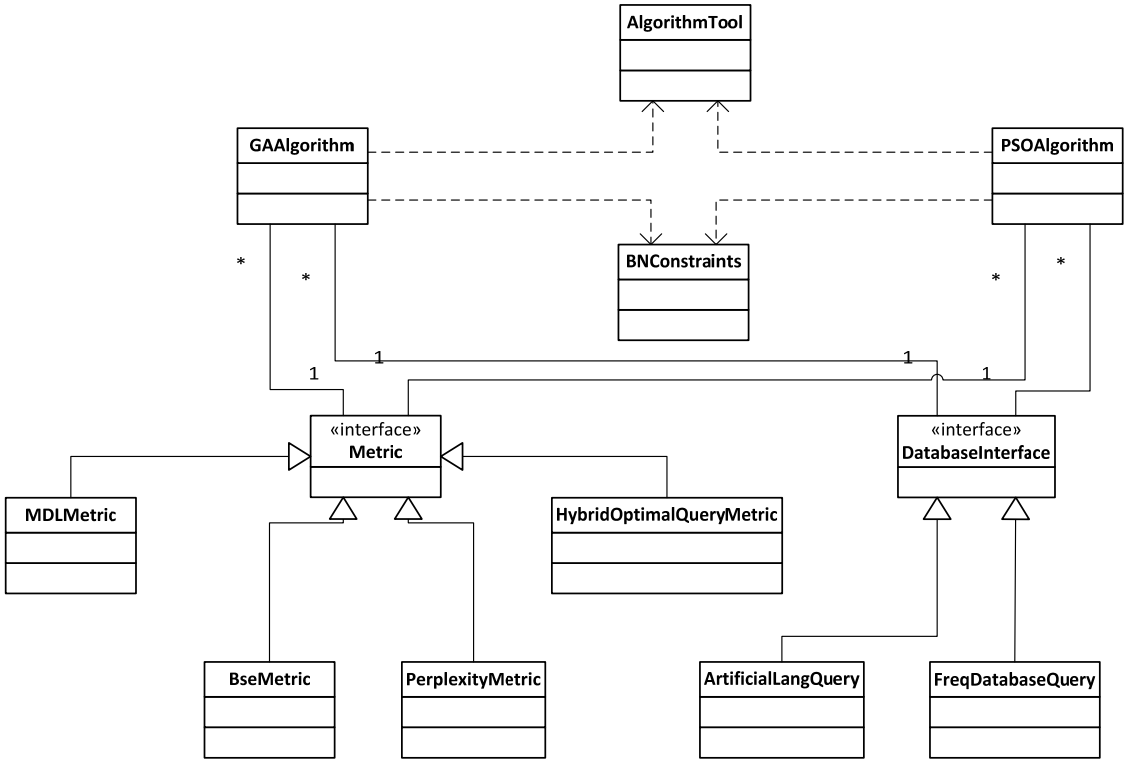
Zweig, G. G. (2003). Bayesian network structures and inference techniques for automatic speech recognition. *New Computational Paradigms for Acoustic Modeling in Speech Recognition , 17* (2-3), 173-193 .

Zweig, G. G. (1998). *Speech Recognition with Dynamic Bayesian Networks.* Berkeley: University of California.

Zweig, G., & Russell, S. (1998). Speech Recognition with Dynamic Bayesian Networks. *Proceedings of the fifteenth national/tenth conference on Artificial intelligence/Innovative applications of artificial intelligence* (pp. 173 - 180). Madison, Wisconsin: AAAI Press.

APPENDIX A: CLASS DIAGRAM

In this appendix, the class diagram of the implemented computer application in order to perform the experiments is depicted.



APPENDIX B: RESULT OF EXPERIMENTS

In this appendix, all results for each experiment in a form of graphs and the resulting networks are given.

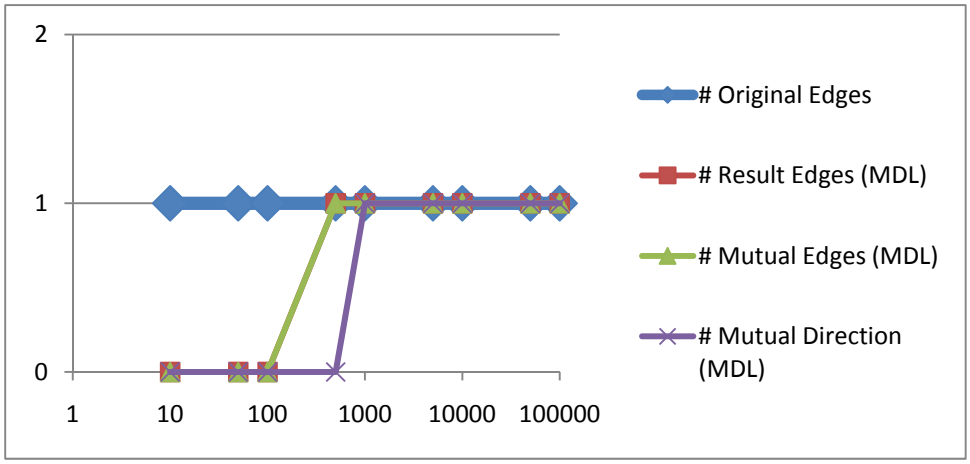
EXPERIMENT 1

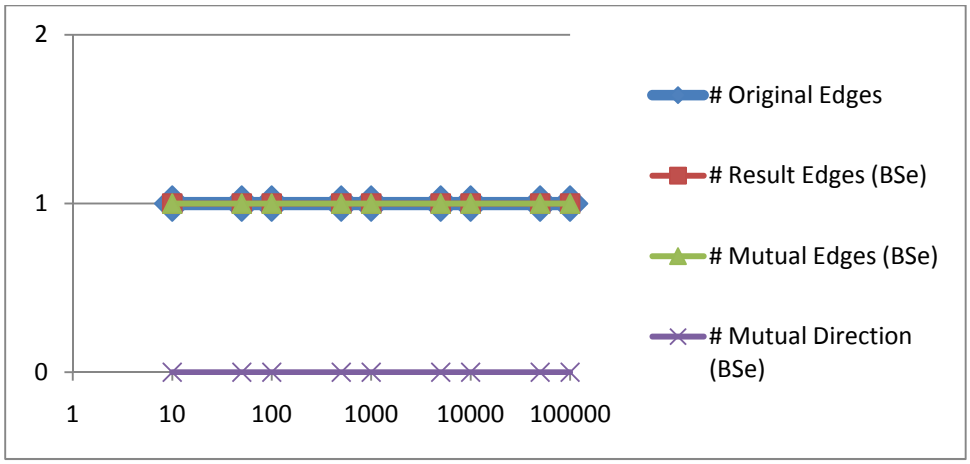
In the graphs below, the vertical axis represents the number of sentences in the database and the horizontal axis represents the number of edges.

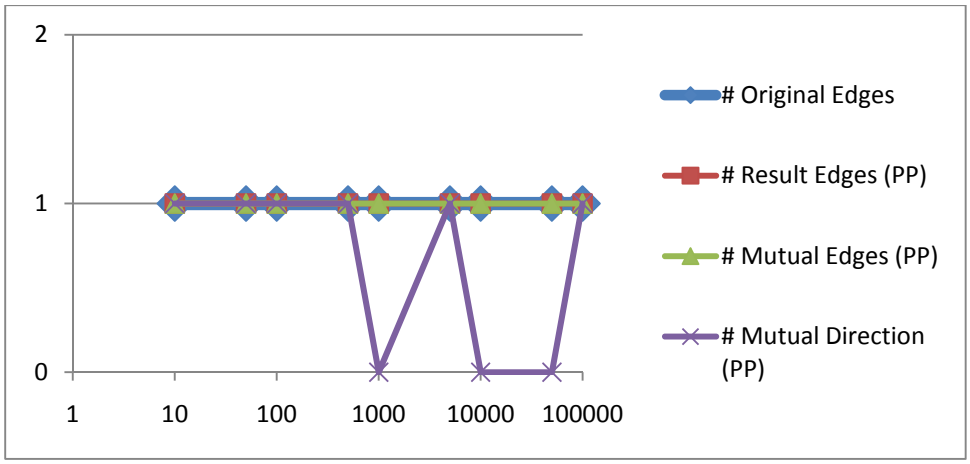
The acronyms in the graphs represent the following metrics:

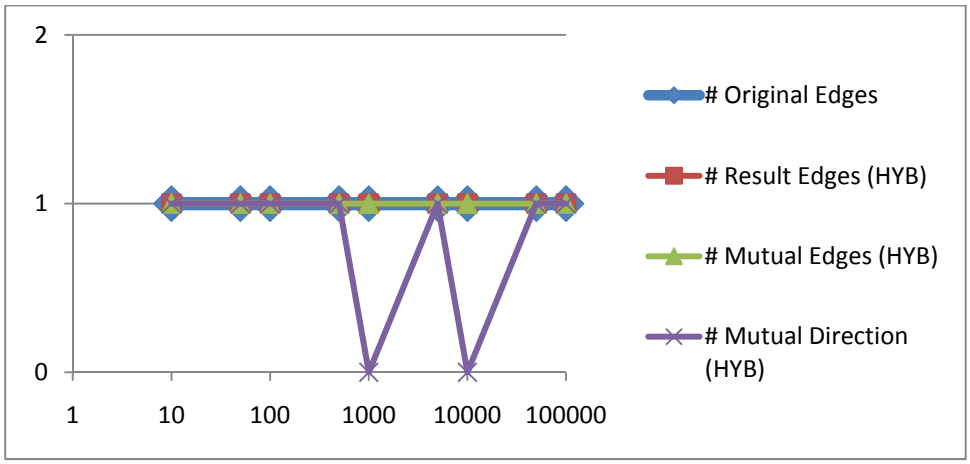
- MDL = the MDL Metric
- BSe = the BSe Metric
- PP = the Perplexity Metric
- HYB = the Modified Metric.

BN 1

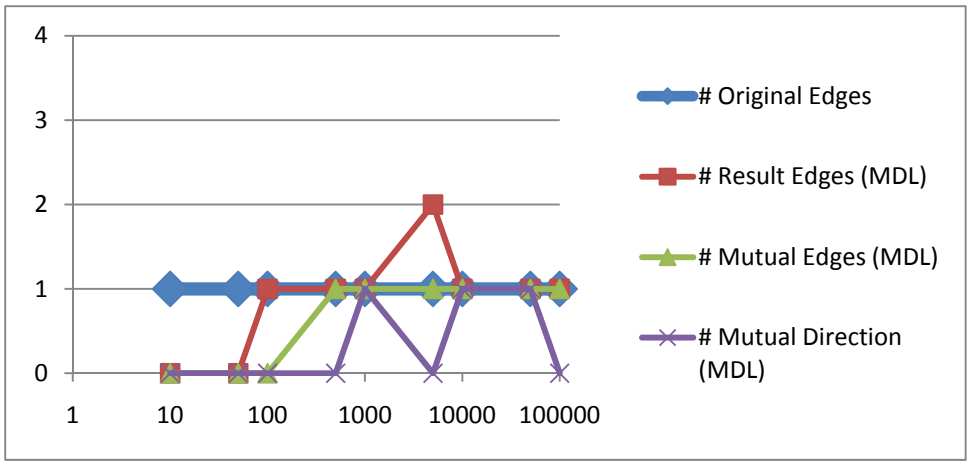


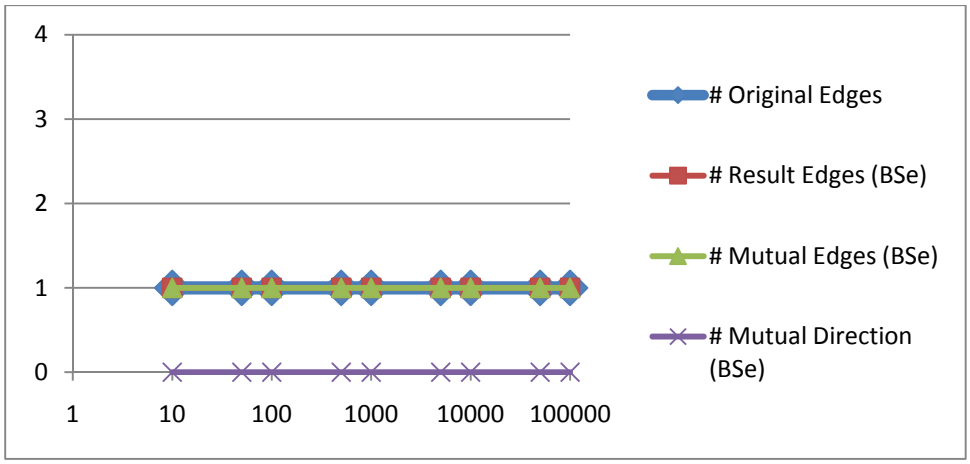


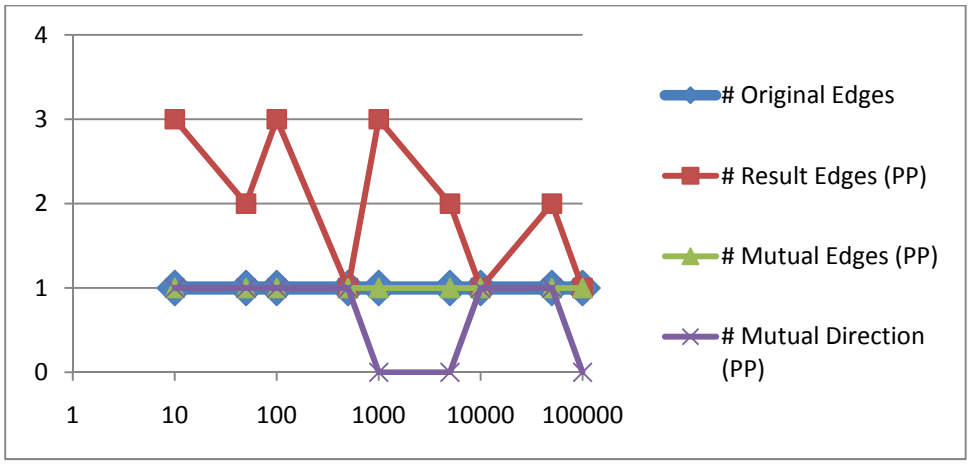

2
1
0
1
10
100
1000
10000
100000
Original Edges
Result Edges (PP)
Mutual Edges (PP)
Mutual Direction (PP)

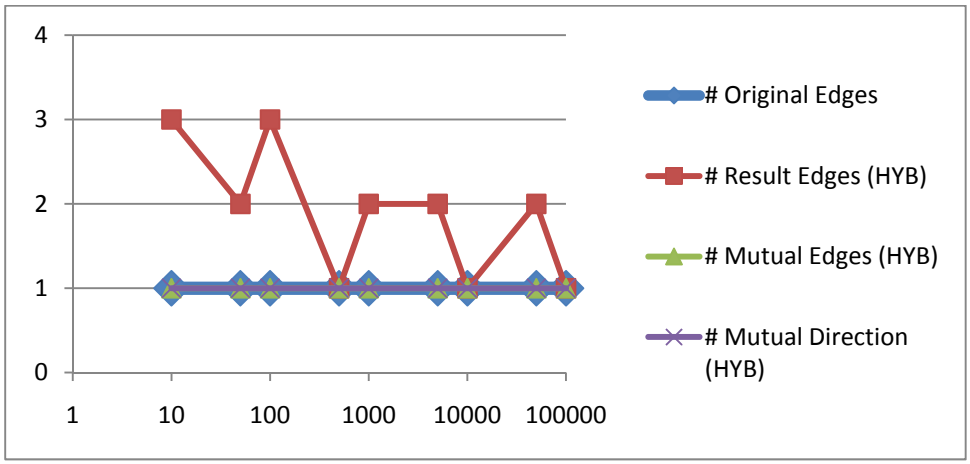

2
1
0
1
10
100
1000
10000
100000
Original Edges
Result Edges (HYB)
Mutual Edges (HYB)
Mutual Direction (HYB)

BN 2

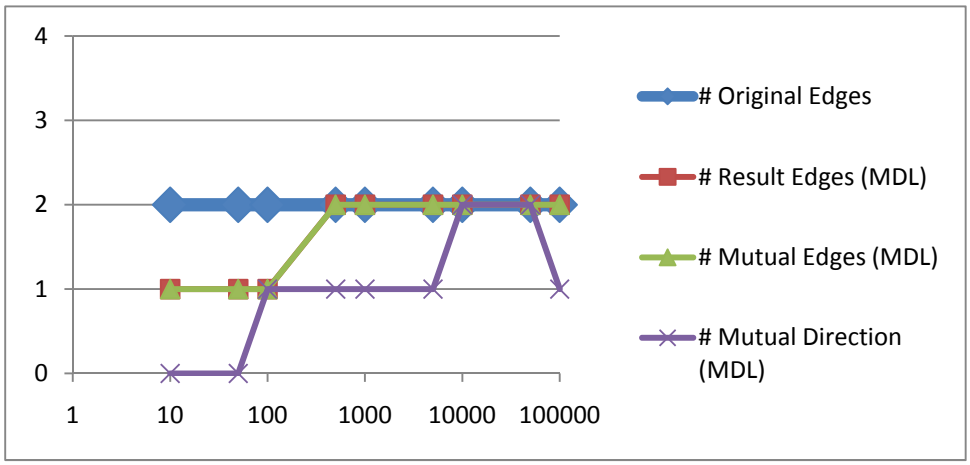

4
3
2
1
0
1
10
100
1000
10000
100000
Original Edges
Result Edges (MDL)
Mutual Edges (MDL)
Mutual Direction (MDL)

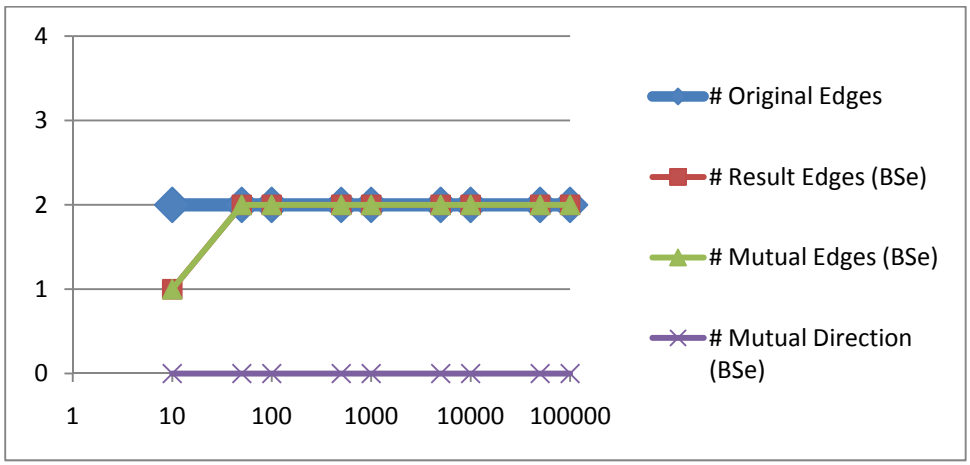

4
3
2
1
0
1
10
100
1000
10000
100000
Original Edges
Result Edges (BSe)
Mutual Edges (BSe)
Mutual Direction (BSe)

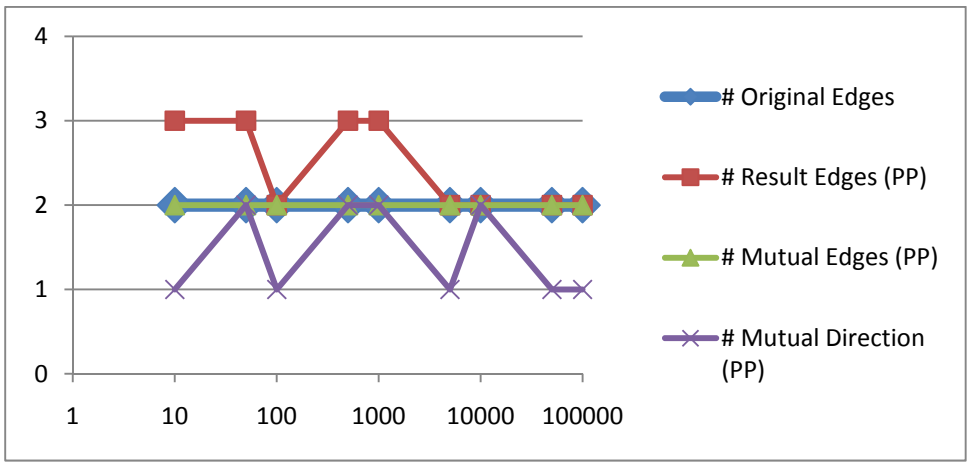

4
3
2
1
0
1
10
100
1000
10000
100000
Original Edges
Result Edges (PP)
Mutual Edges (PP)
Mutual Direction (PP)

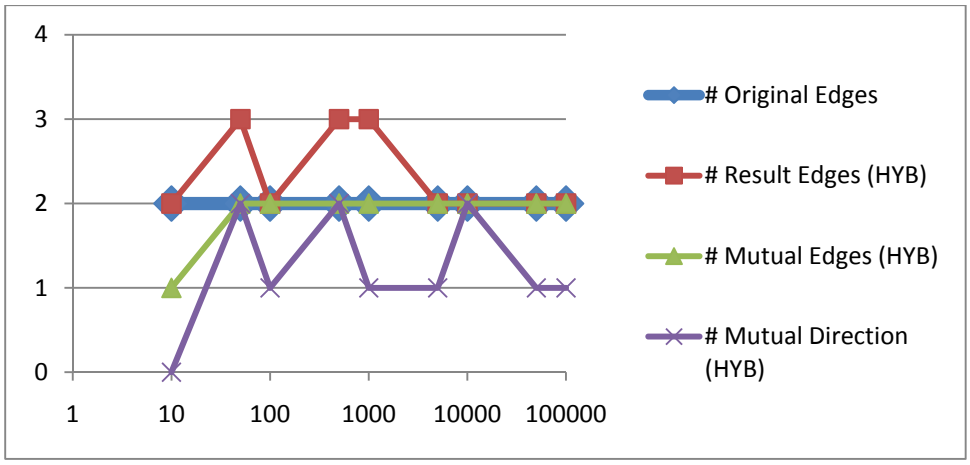

4
3
2
1
0
1
10
100
1000
10000
100000
Original Edges
Result Edges (HYB)
Mutual Edges (HYB)
Mutual Direction (HYB)

BN 3

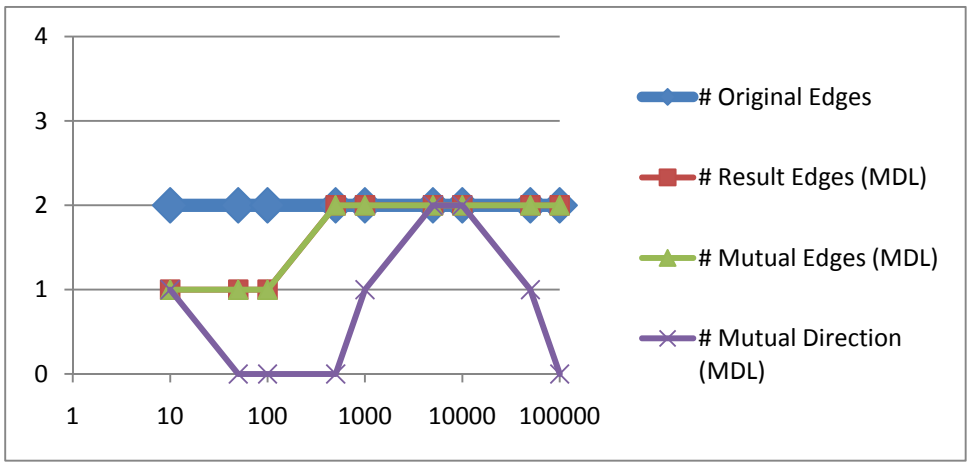

4
3
2
1
0
1
10
100
1000
10000
100000
Original Edges
Result Edges (MDL)
Mutual Edges (MDL)
Mutual Direction (MDL)

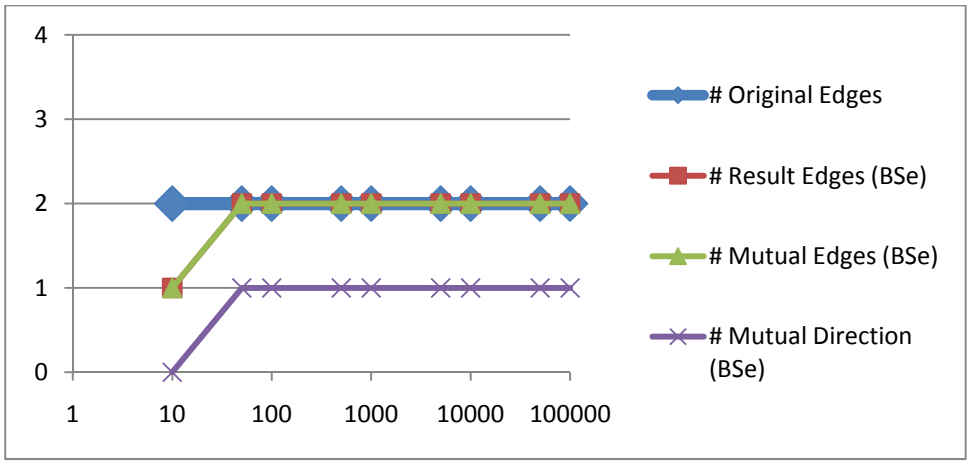

4
3
2
1
0
1
10
100
1000
10000
100000
Original Edges
Result Edges (BSe)
Mutual Edges (BSe)
Mutual Direction (BSe)

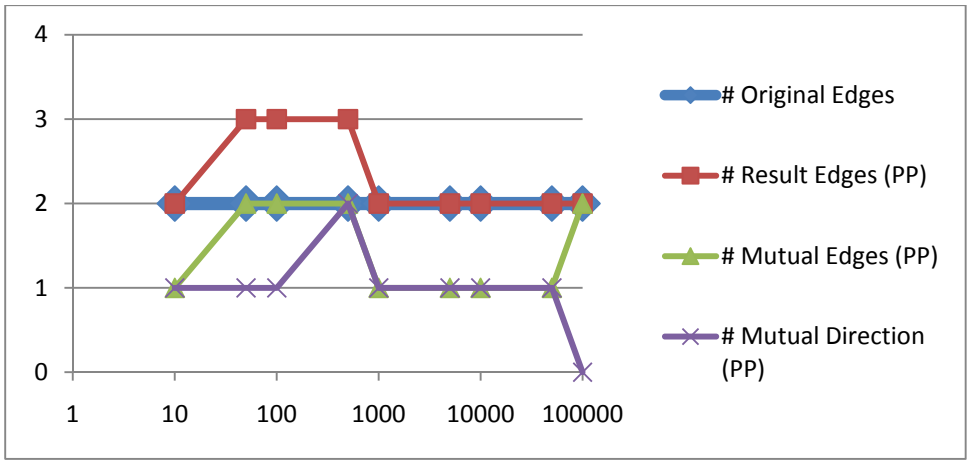

4
3
2
1
0
1
10
100
1000
10000
100000
Original Edges
Result Edges (PP)
Mutual Edges (PP)
Mutual Direction (PP)

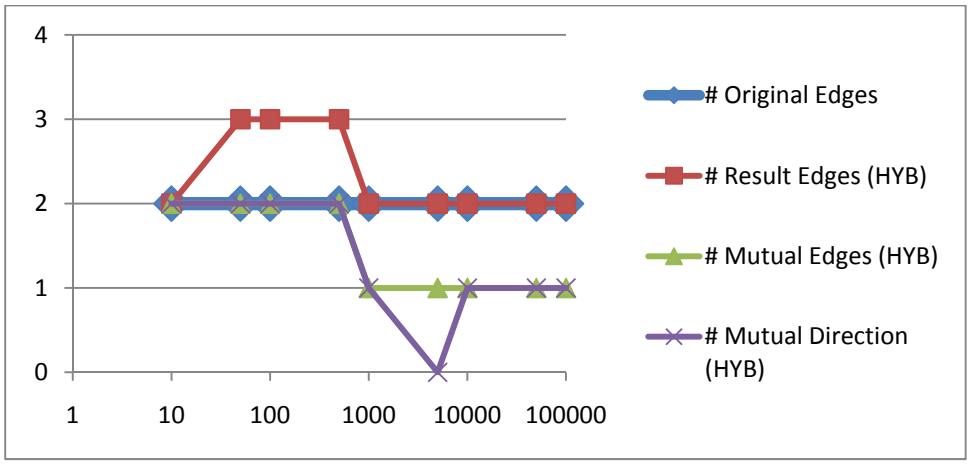

4
3
2
1
0
1
10
100
1000
10000
100000
Original Edges
Result Edges (HYB)
Mutual Edges (HYB)
Mutual Direction (HYB)

BN 4

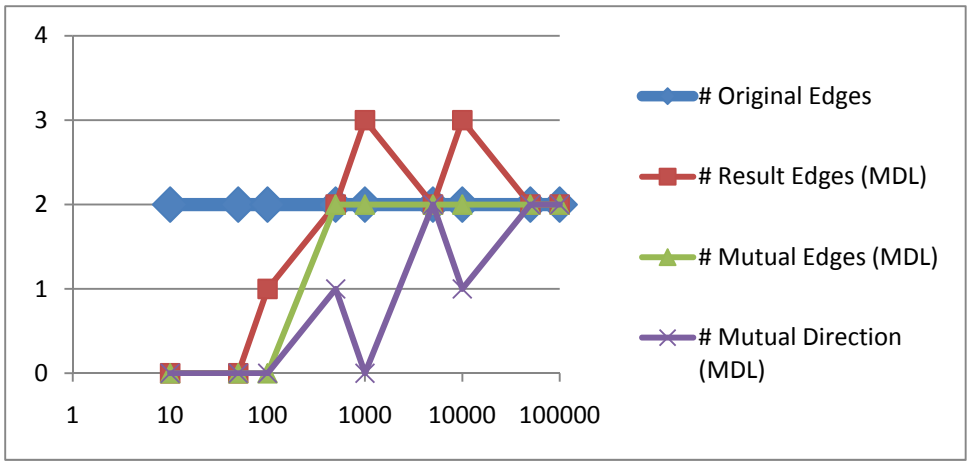

4
3
2
1
0
1
10
100
1000
10000
100000
Original Edges
Result Edges (MDL)
Mutual Edges (MDL)
Mutual Direction (MDL)

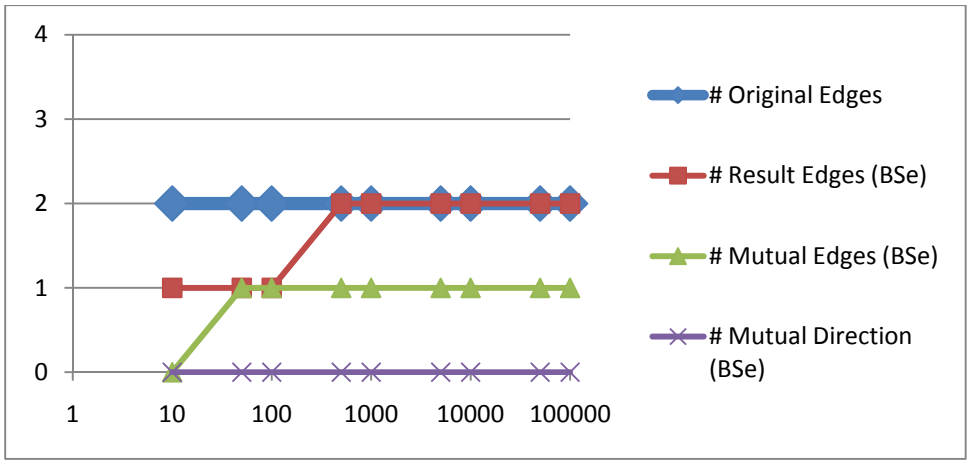

4
3
2
1
0
1
10
100
1000
10000
100000
Original Edges
Result Edges (BSe)
Mutual Edges (BSe)
Mutual Direction (BSe)

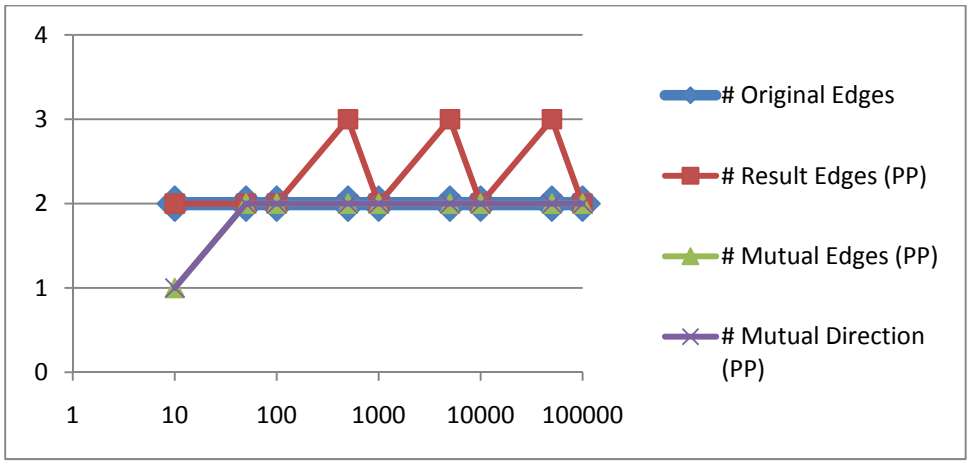

4
3
2
1
0
1
10
100
1000
10000
100000
Original Edges
Result Edges (PP)
Mutual Edges (PP)
Mutual Direction (PP)

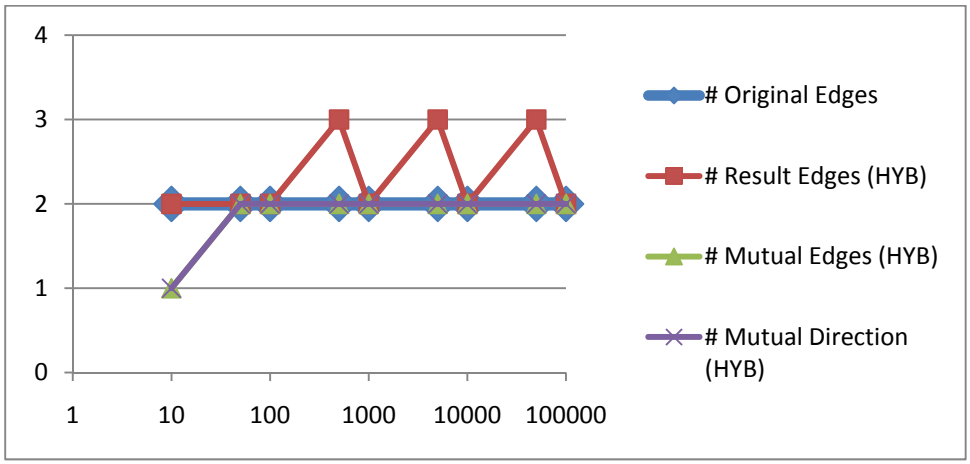

4
3
2
1
0
1
10
100
1000
10000
100000
Original Edges
Result Edges (HYB)
Mutual Edges (HYB)
Mutual Direction (HYB)

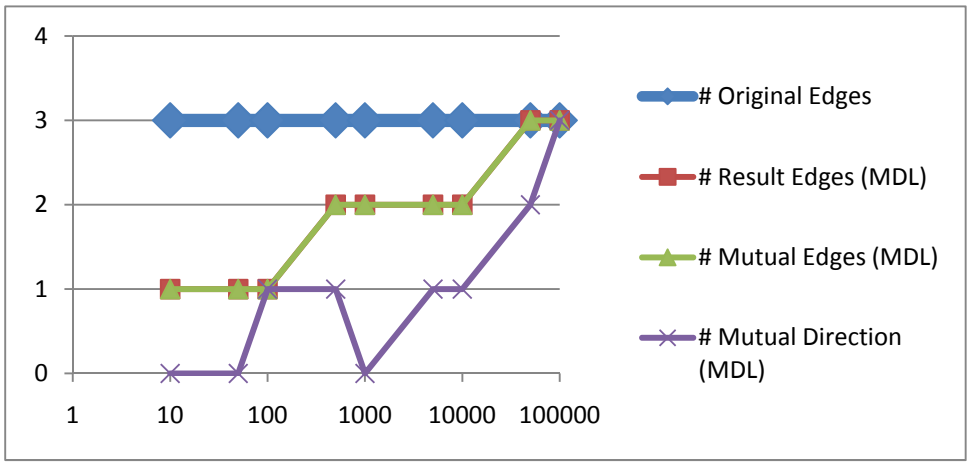
BN 5

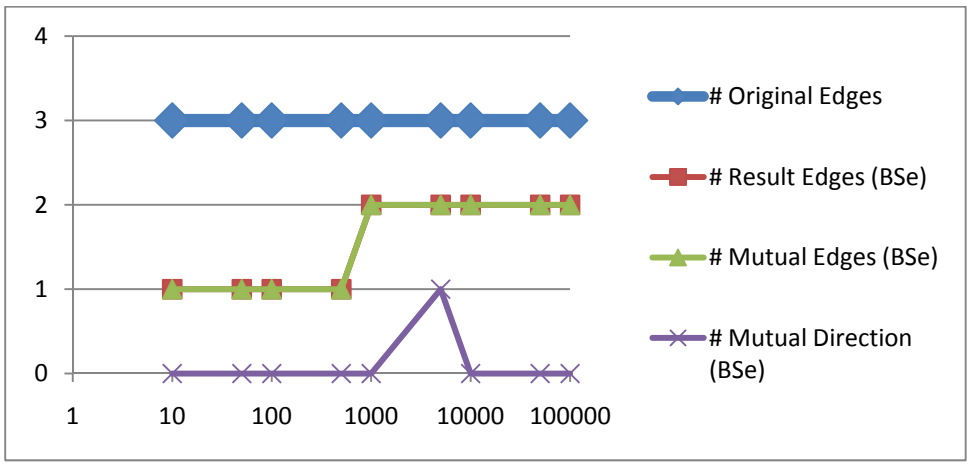

4
3
2
1
0
1
10
100
1000
10000
100000
Original Edges
Result Edges (MDL)
Mutual Edges (MDL)
Mutual Direction (MDL)

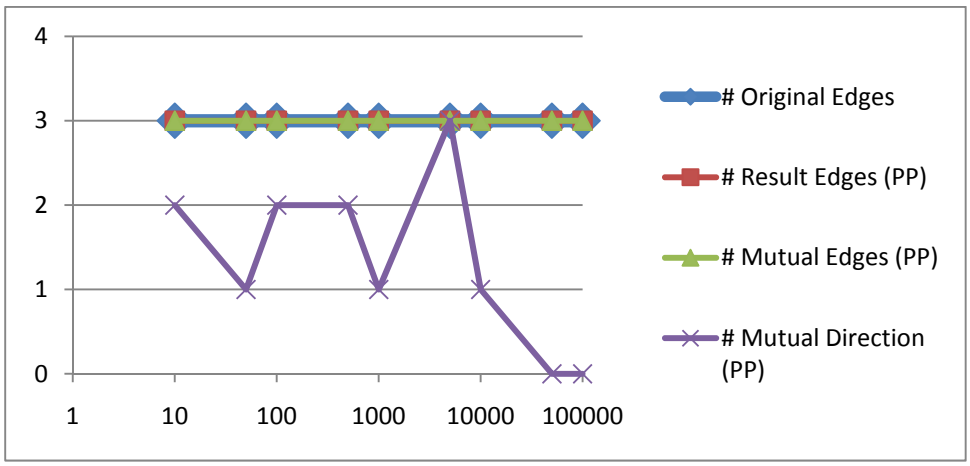

4
3
2
1
0
1
10
100
1000
10000
100000
Original Edges
Result Edges (BSe)
Mutual Edges (BSe)
Mutual Direction (BSe)

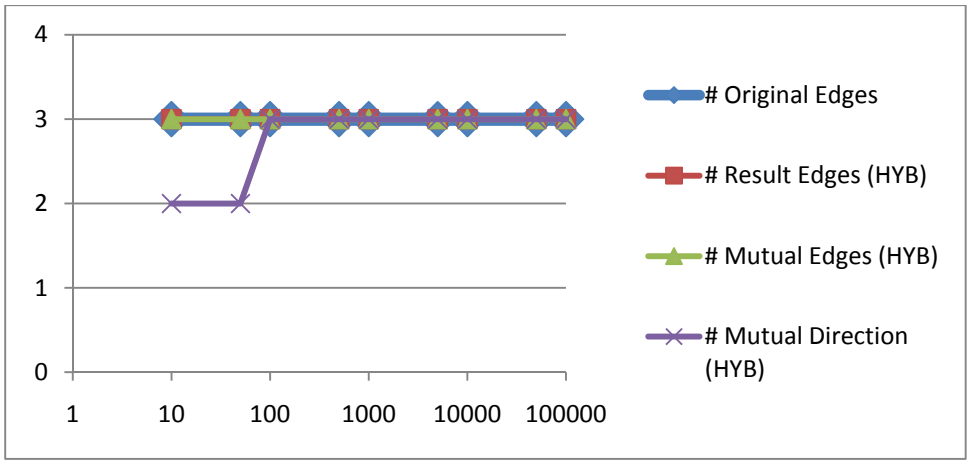

4
3
2
1
0
1
10
100
1000
10000
100000
Original Edges
Result Edges (PP)
Mutual Edges (PP)
Mutual Direction (PP)


4
3
2
1
0
1
10
100
1000
10000
100000
Original Edges
Result Edges (HYB)
Mutual Edges (HYB)
Mutual Direction (HYB)

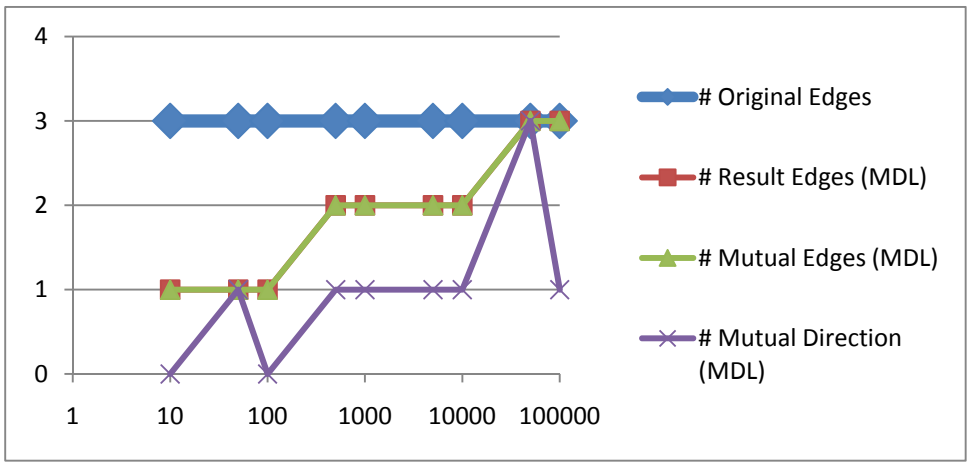

4
3
2
1
0
1
10
100
1000
10000
100000
Original Edges
Result Edges (MDL)
Mutual Edges (MDL)
Mutual Direction (MDL)

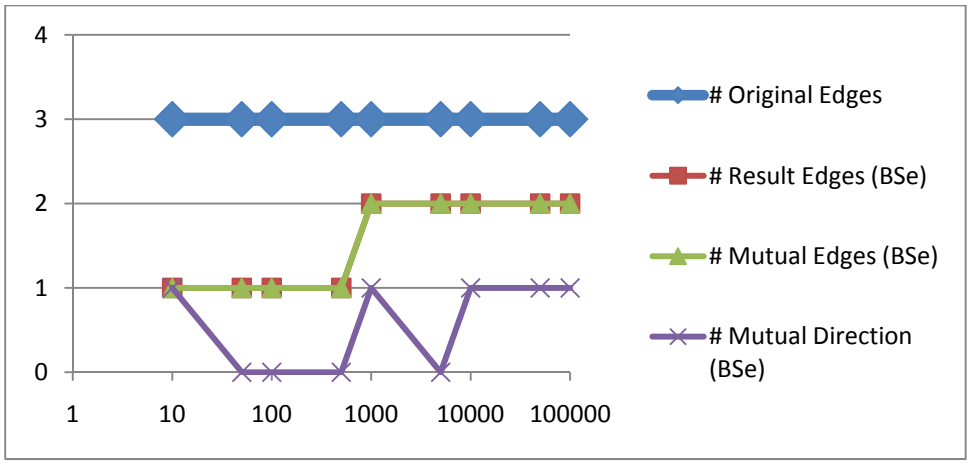

4
3
2
1
0
1
10
100
1000
10000
100000
Original Edges
Result Edges (BSe)
Mutual Edges (BSe)
Mutual Direction (BSe)

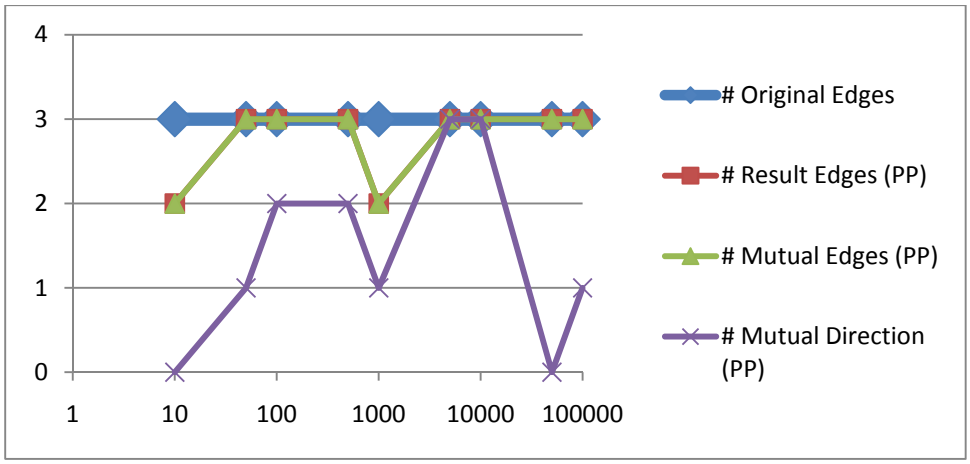

4
3
2
1
0
1
10
100
1000
10000
100000
Original Edges
Result Edges (PP)
Mutual Edges (PP)
Mutual Direction (PP)

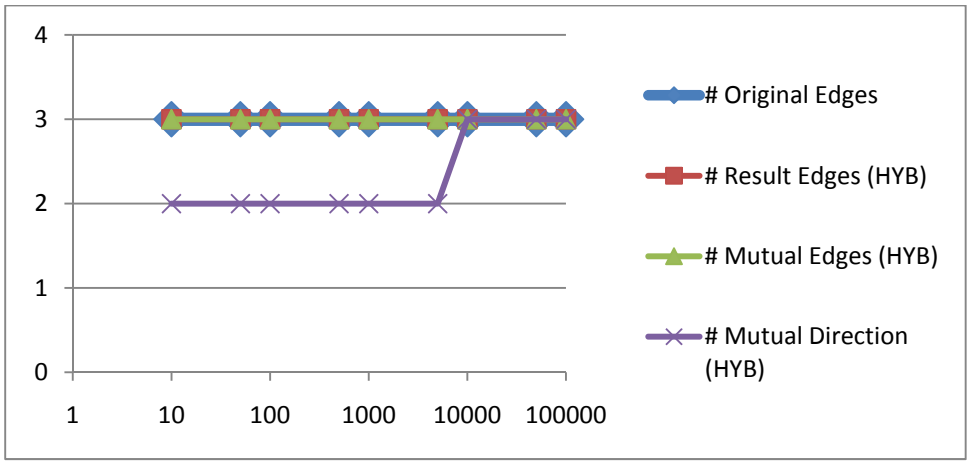

4
3
2
1
0
1
10
100
1000
10000
100000
Original Edges
Result Edges (HYB)
Mutual Edges (HYB)
Mutual Direction (HYB)

BN 7

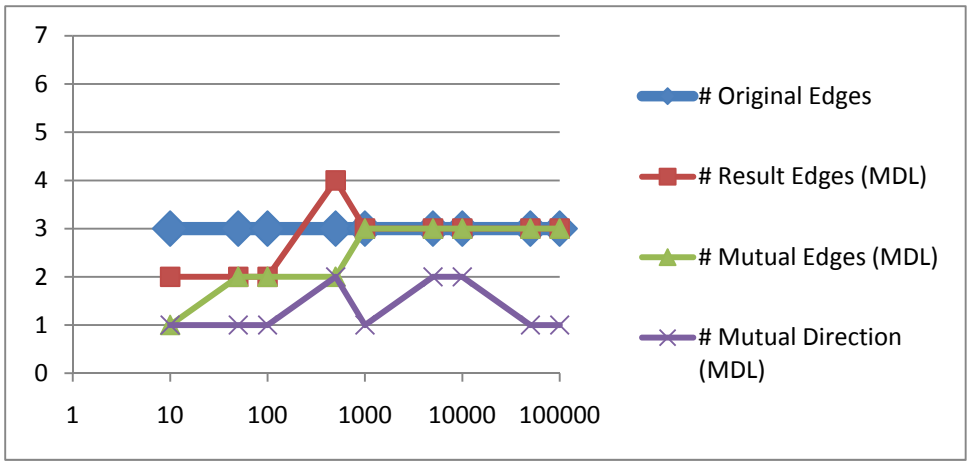

4
3
2
1
0
1
10
100
1000
10000
100000
Original Edges
Result Edges (MDL)
Mutual Edges (MDL)
Mutual Direction (MDL)

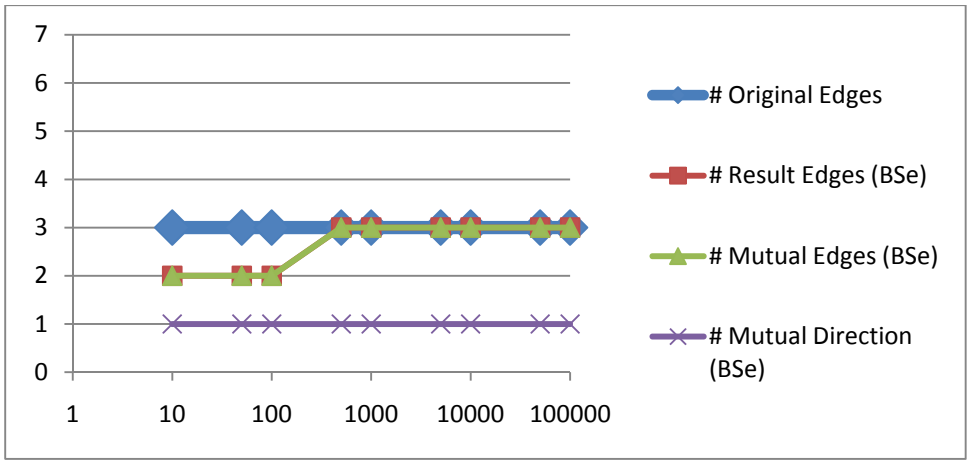

4
3
2
1
0
1
10
100
1000
10000
100000
Original Edges
Result Edges (BSe)
Mutual Edges (BSe)
Mutual Direction (BSe)

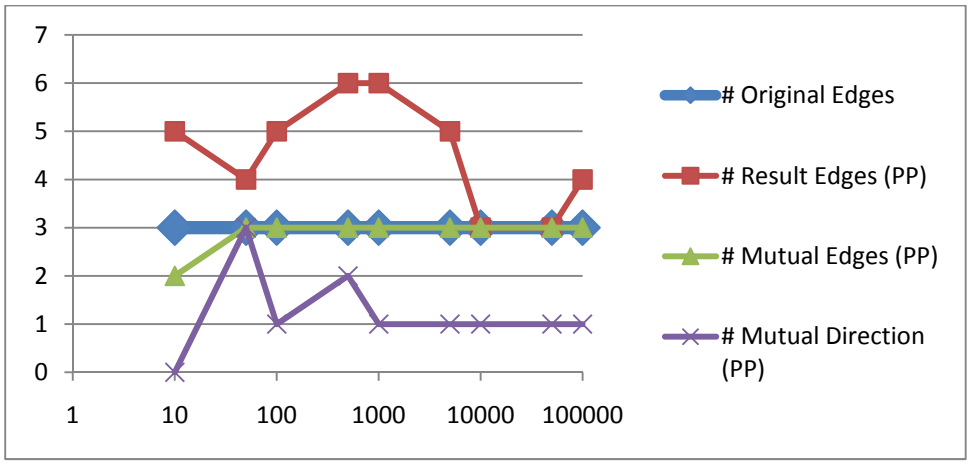

4
3
2
1
0
1
10
100
1000
10000
100000
Original Edges
Result Edges (PP)
Mutual Edges (PP)
Mutual Direction (PP)

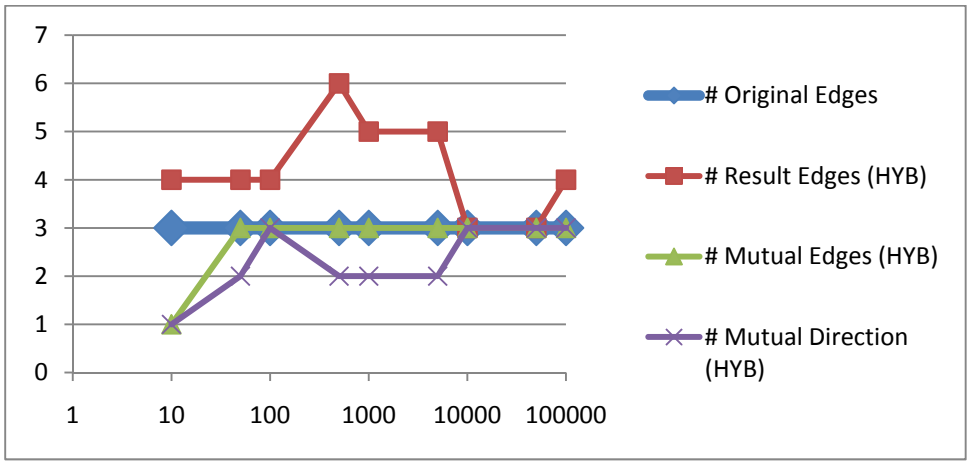

4
3
2
1
0
1
10
100
1000
10000
100000
Original Edges
Result Edges (HYB)
Mutual Edges (HYB)
Mutual Direction (HYB)

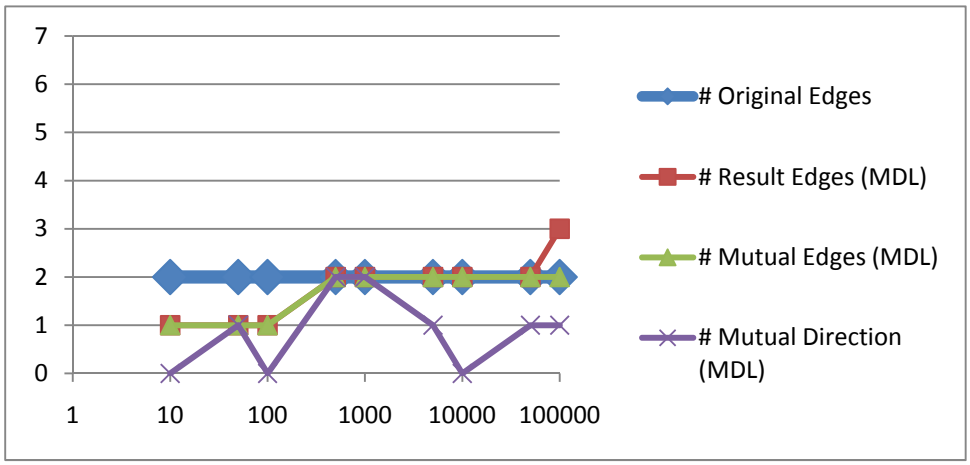
BN 8

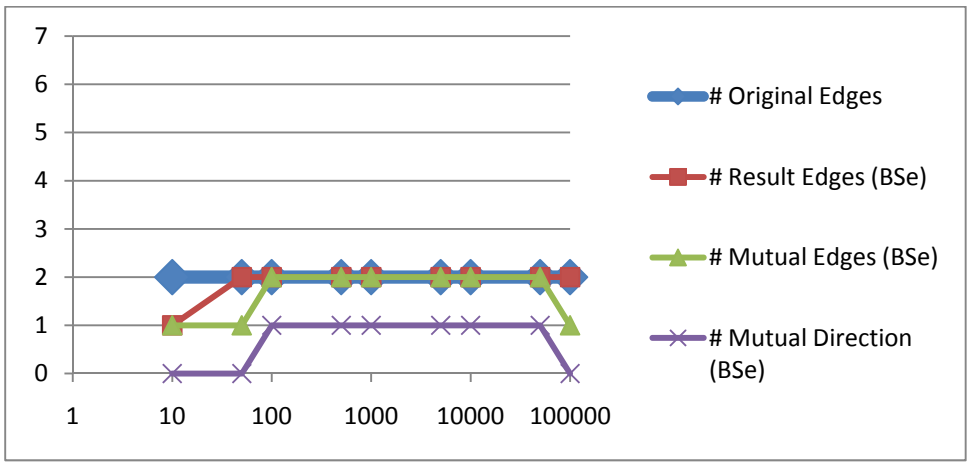

7
6
5
4
3
2
1
0
1
10
100
1000
10000
100000
Original Edges
Result Edges (MDL)
Mutual Edges (MDL)
Mutual Direction (MDL)

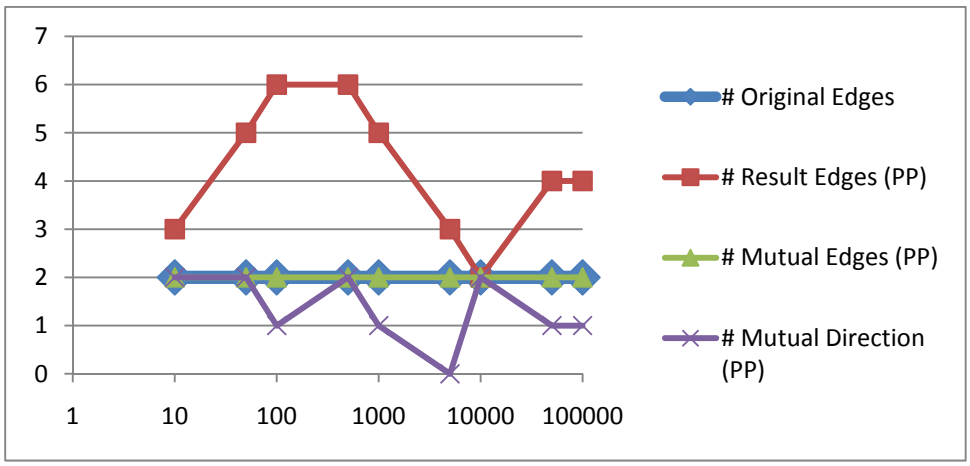

7
6
5
4
3
2
1
0
1
10
100
1000
10000
100000
Original Edges
Result Edges (BSe)
Mutual Edges (BSe)
Mutual Direction (BSe)

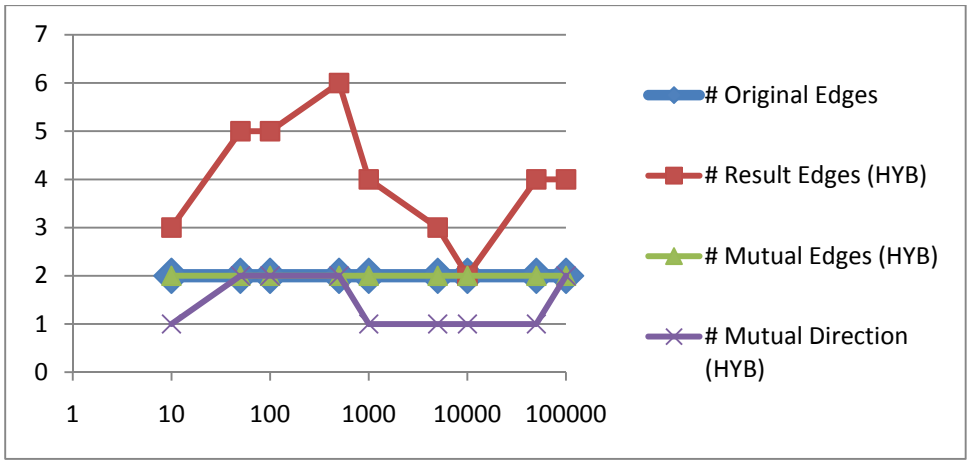

7
6
5
4
3
2
1
0
1
10
100
1000
10000
100000
Original Edges
Result Edges (PP)
Mutual Edges (PP)
Mutual Direction (PP)


7
6
5
4
3
2
1
0
1
10
100
1000
10000
100000
Original Edges
Result Edges (HYB)
Mutual Edges (HYB)
Mutual Direction (HYB)

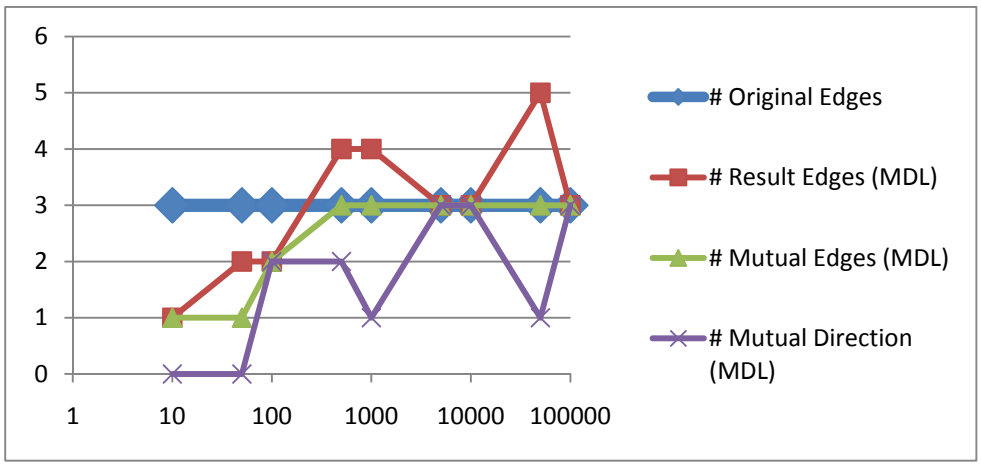

7
6
5
4
3
2
1
0
1
10
100
1000
10000
100000
Original Edges
Result Edges (MDL)
Mutual Edges (MDL)
Mutual Direction (MDL)

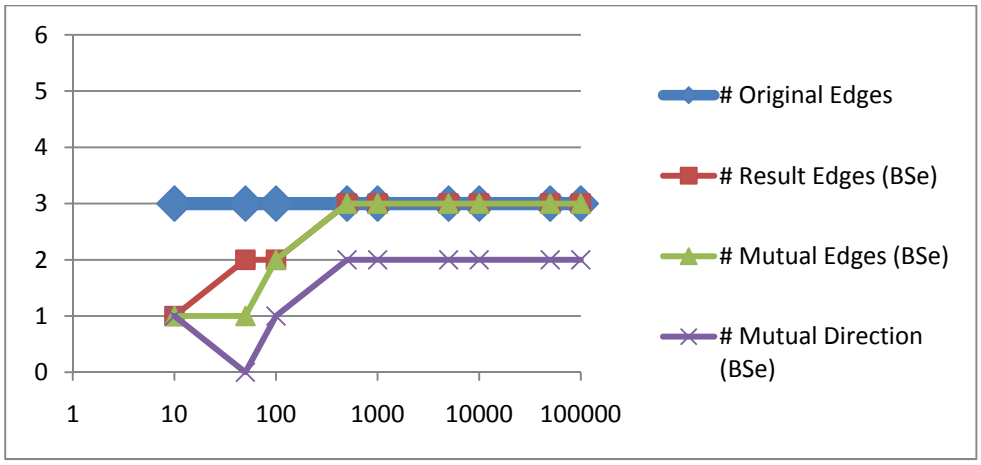

7
6
5
4
3
2
1
0
1
10
100
1000
10000
100000
Original Edges
Result Edges (BSe)
Mutual Edges (BSe)
Mutual Direction (BSe)

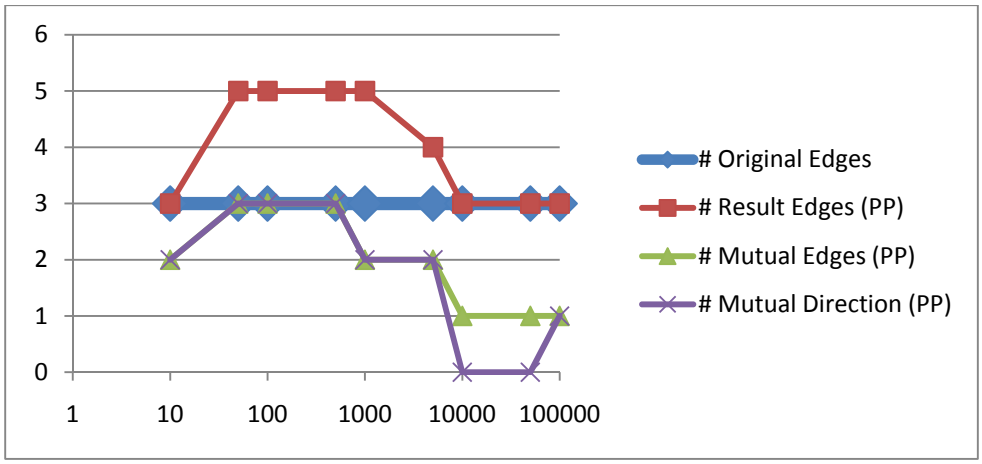

7
6
5
4
3
2
1
0
1
10
100
1000
10000
100000
Original Edges
Result Edges (PP)
Mutual Edges (PP)
Mutual Direction (PP)

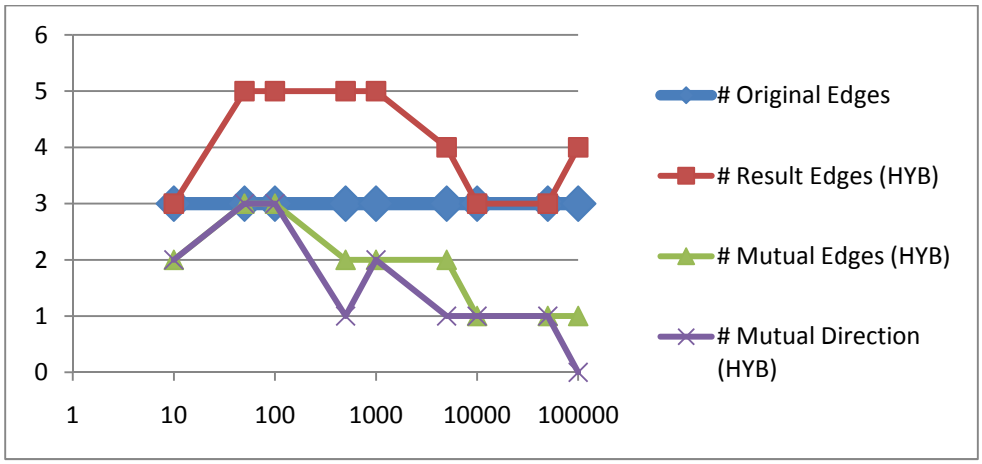

7
6
5
4
3
2
1
0
1
10
100
1000
10000
100000
Original Edges
Result Edges (HYB)
Mutual Edges (HYB)
Mutual Direction (HYB)

BN 10

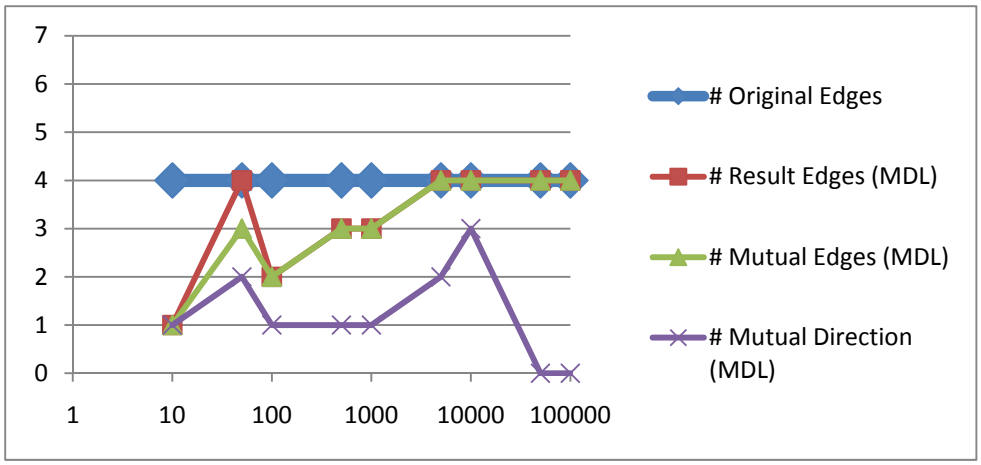

6
5
4
3
2
1
0
1
10
100
1000
10000
100000
Original Edges
Result Edges (MDL)
Mutual Edges (MDL)
Mutual Direction (MDL)

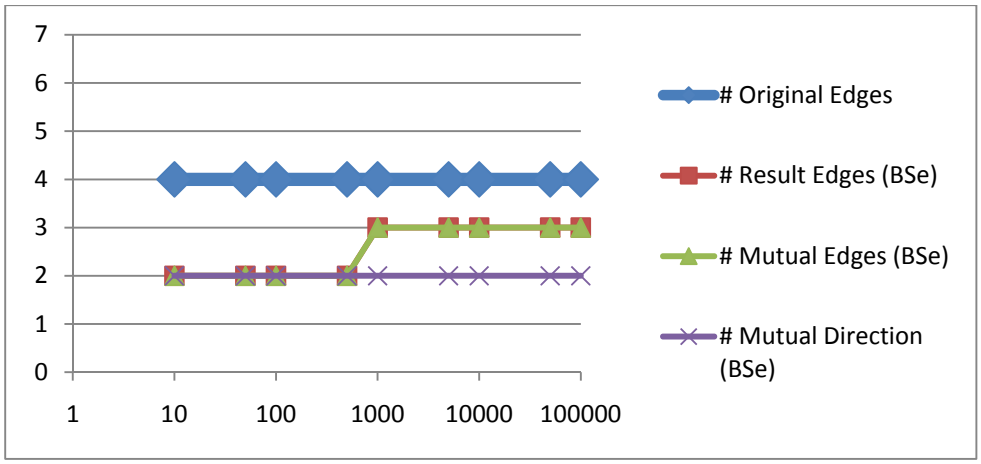

6
5
4
3
2
1
0
1
10
100
1000
10000
100000
Original Edges
Result Edges (BSe)
Mutual Edges (BSe)
Mutual Direction (BSe)

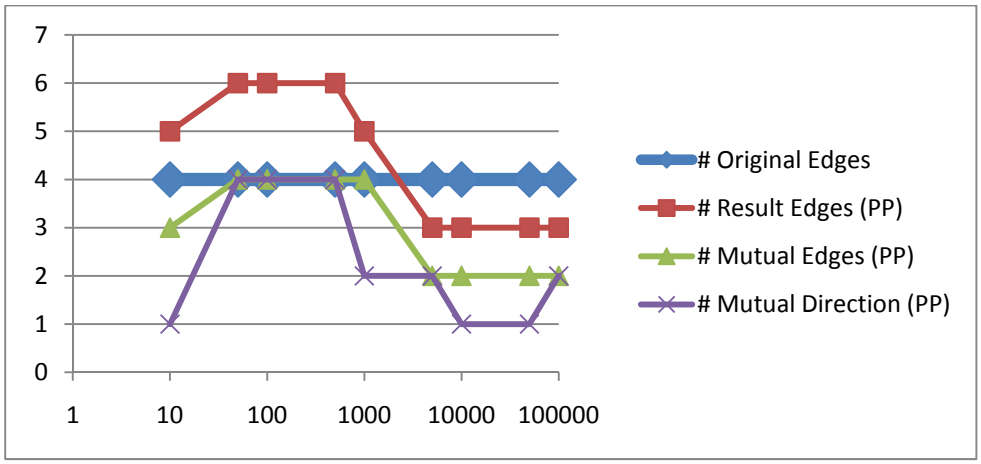

6
5
4
3
2
1
0
1
10
100
1000
10000
100000
Original Edges
Result Edges (PP)
Mutual Edges (PP)
Mutual Direction (PP)

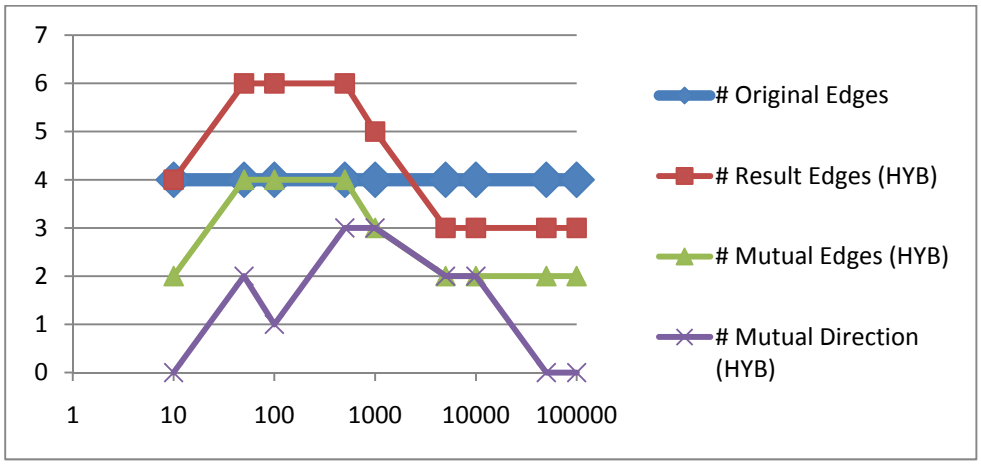

6
5
4
3
2
1
0
1
10
100
1000
10000
100000
Original Edges
Result Edges (HYB)
Mutual Edges (HYB)
Mutual Direction (HYB)

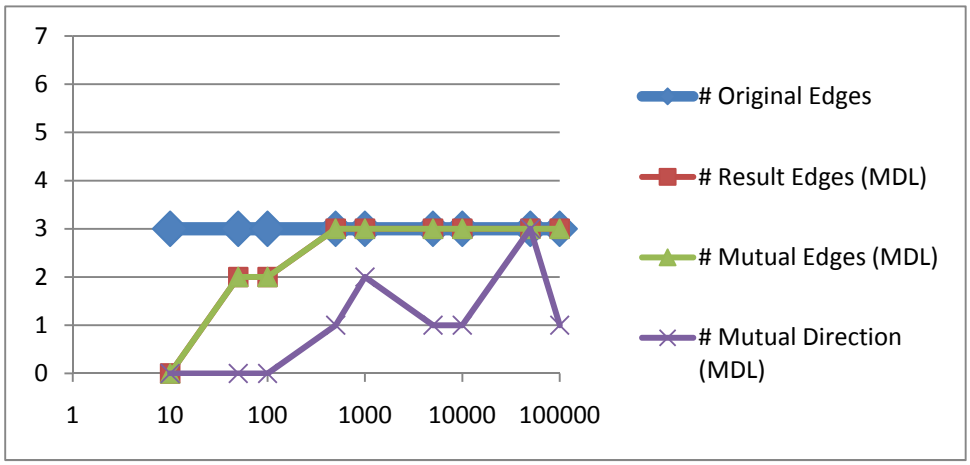
BN 11

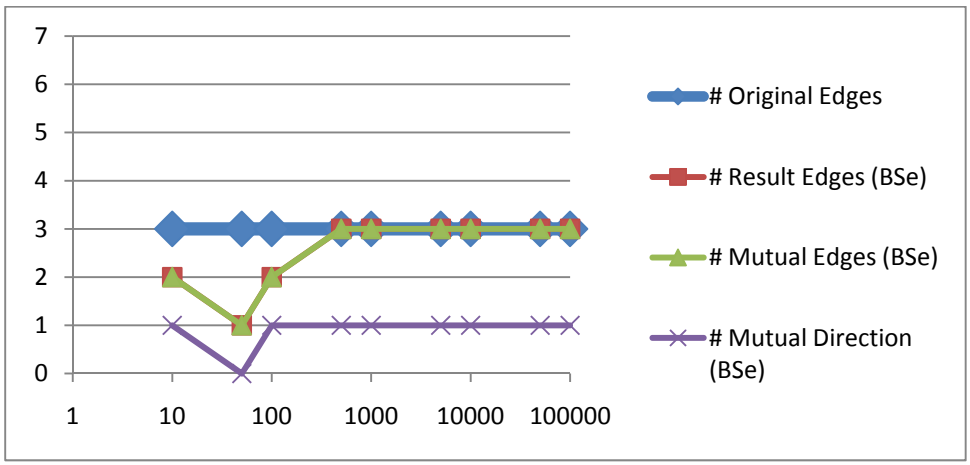

7
6
5
4
3
2
1
0
1
10
100
1000
10000
100000
Original Edges
Result Edges (MDL)
Mutual Edges (MDL)
Mutual Direction (MDL)

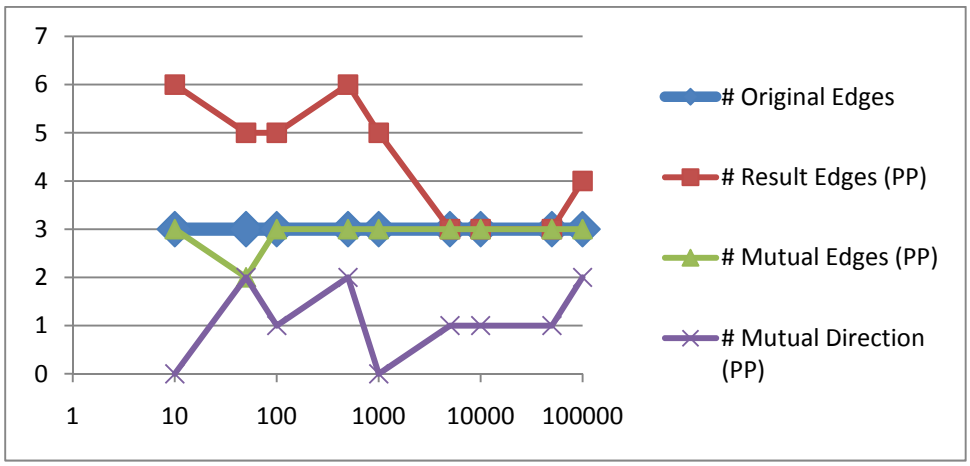

7
6
5
4
3
2
1
0
1
10
100
1000
10000
100000
Original Edges
Result Edges (BSe)
Mutual Edges (BSe)
Mutual Direction (BSe)

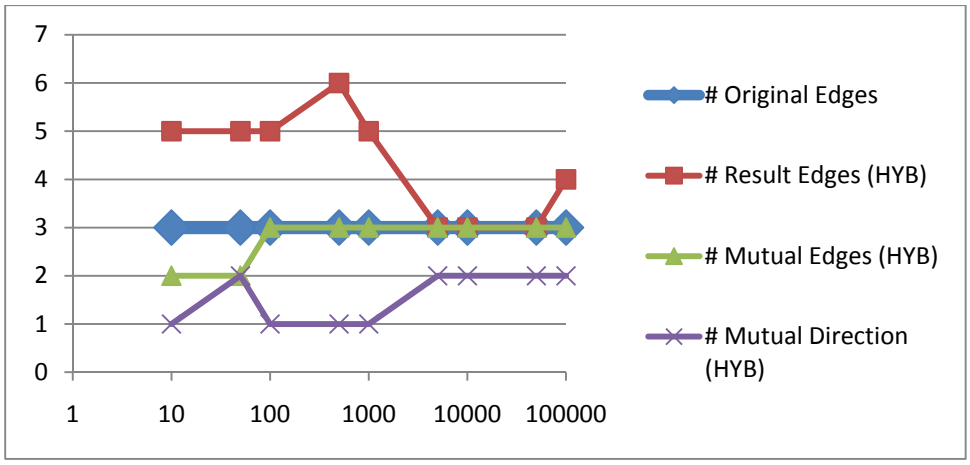

7
6
5
4
3
2
1
0
1
10
100
1000
10000
100000
Original Edges
Result Edges (PP)
Mutual Edges (PP)
Mutual Direction (PP)


7
6
5
4
3
2
1
0
1
10
100
1000
10000
100000
Original Edges
Result Edges (HYB)
Mutual Edges (HYB)
Mutual Direction (HYB)

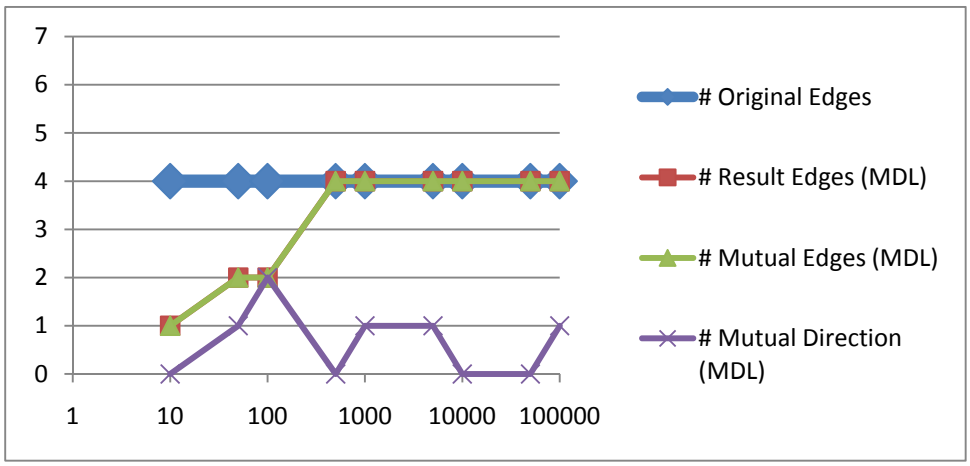

7
6
5
4
3
2
1
0
1
10
100
1000
10000
100000
Original Edges
Result Edges (MDL)
Mutual Edges (MDL)
Mutual Direction (MDL)

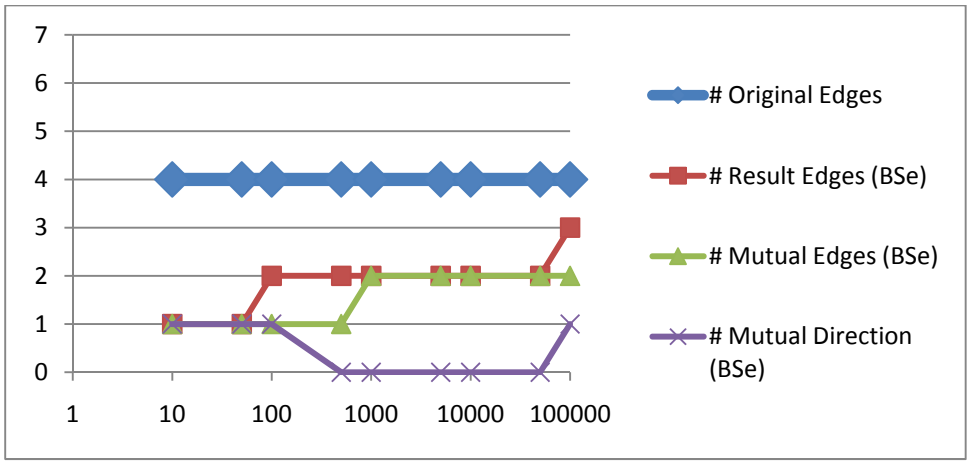

7
6
5
4
3
2
1
0
1
10
100
1000
10000
100000
Original Edges
Result Edges (BSe)
Mutual Edges (BSe)
Mutual Direction (BSe)

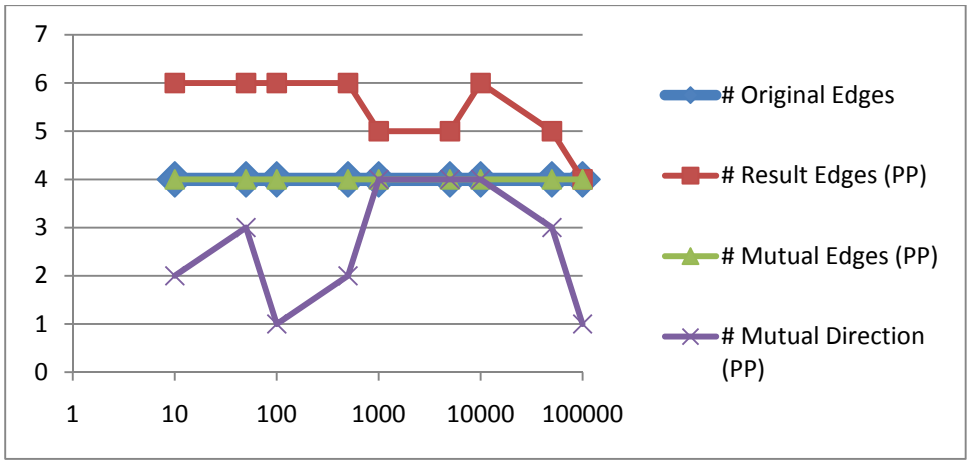

7
6
5
4
3
2
1
0
1
10
100
1000
10000
100000
Original Edges
Result Edges (PP)
Mutual Edges (PP)
Mutual Direction (PP)

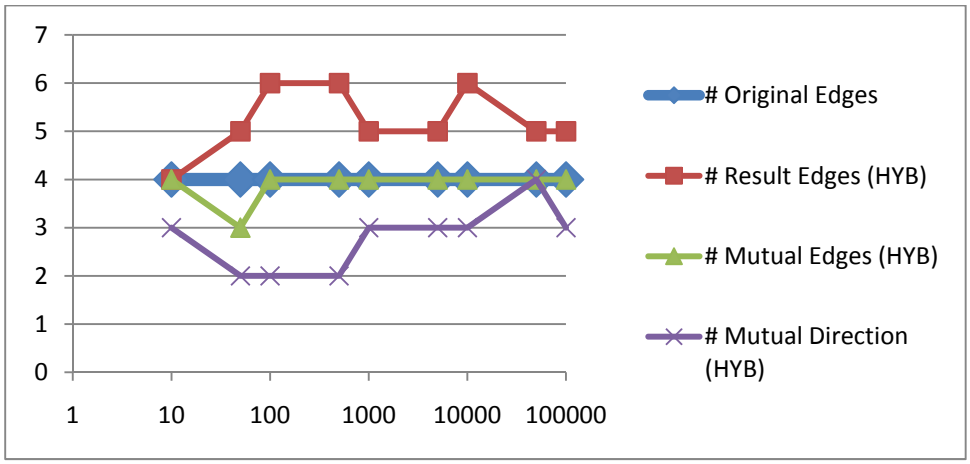

7
6
5
4
3
2
1
0
1
10
100
1000
10000
100000
Original Edges
Result Edges (HYB)
Mutual Edges (HYB)
Mutual Direction (HYB)

BN 13

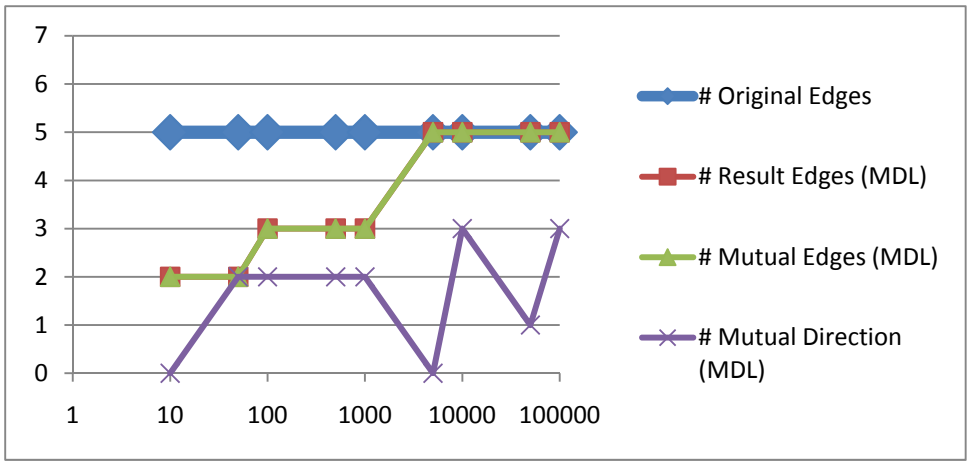

7
6
5
4
3
2
1
0
1
10
100
1000
10000
100000
Original Edges
Result Edges (MDL)
Mutual Edges (MDL)
Mutual Direction (MDL)

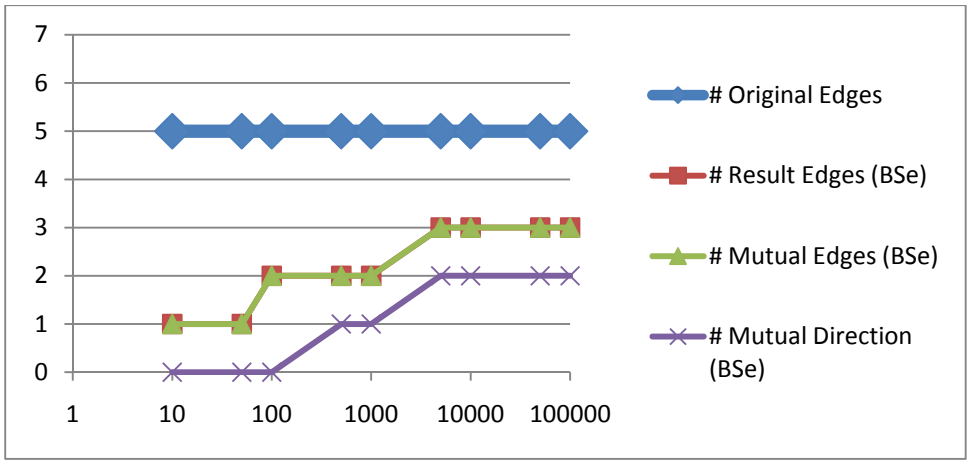

7
6
5
4
3
2
1
0
1
10
100
1000
10000
100000
Original Edges
Result Edges (BSe)
Mutual Edges (BSe)
Mutual Direction (BSe)

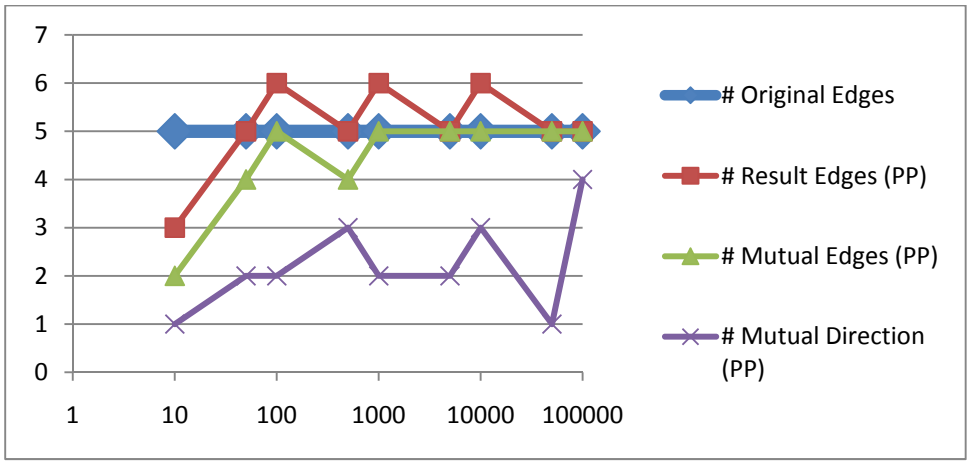

7
6
5
4
3
2
1
0
1
10
100
1000
10000
100000
Original Edges
Result Edges (PP)
Mutual Edges (PP)
Mutual Direction (PP)

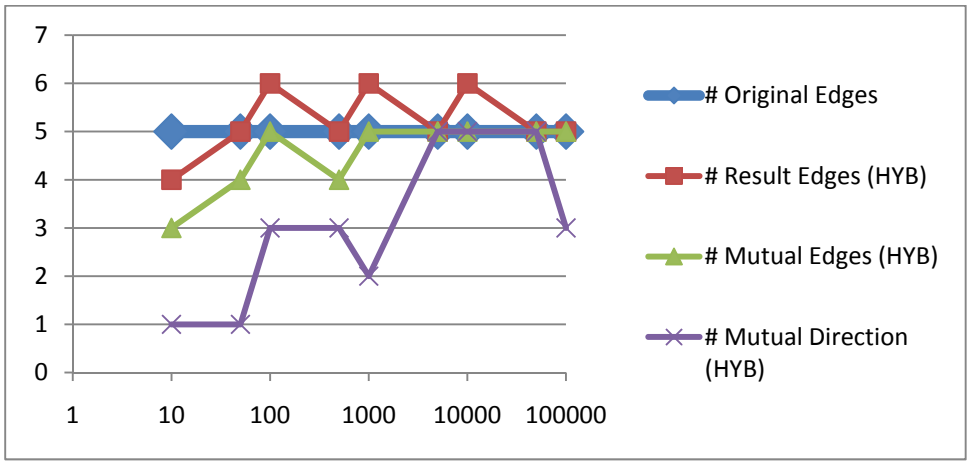

7
6
5
4
3
2
1
0
1
10
100
1000
10000
100000
Original Edges
Result Edges (HYB)
Mutual Edges (HYB)
Mutual Direction (HYB)

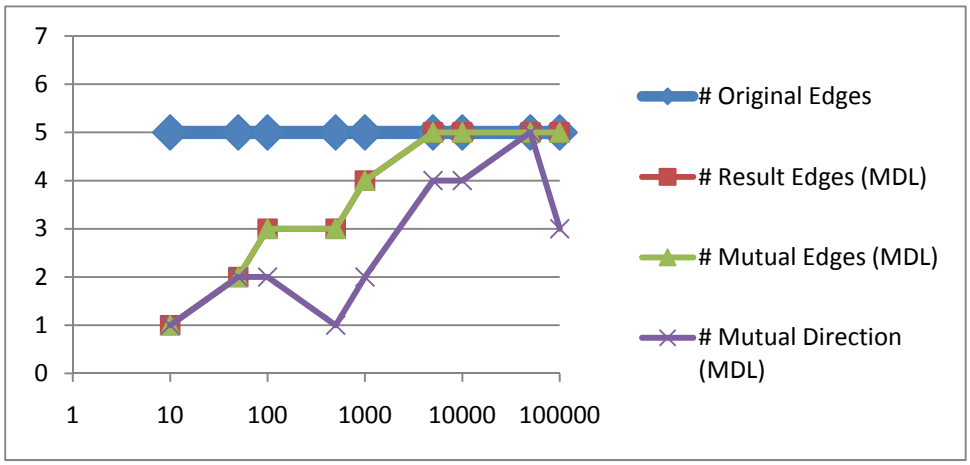
BN 14

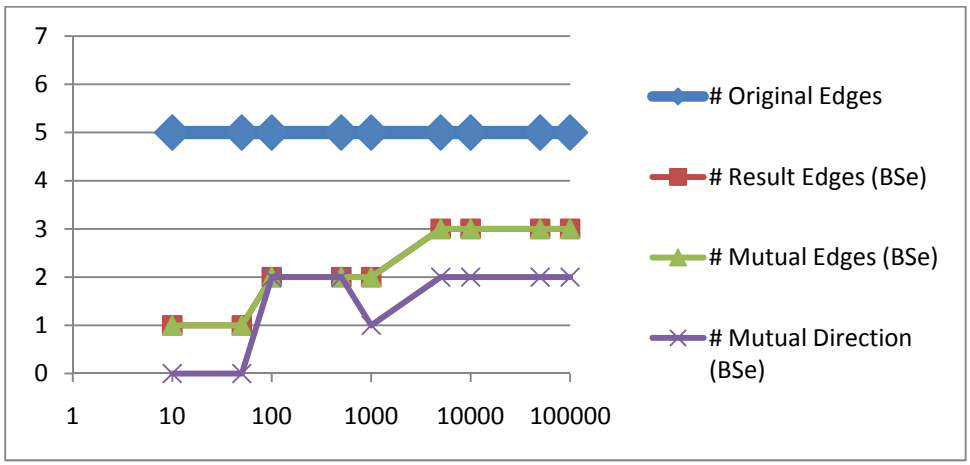

7
6
5
4
3
2
1
0
1
10
100
1000
10000
100000
Original Edges
Result Edges (MDL)
Mutual Edges (MDL)
Mutual Direction (MDL)

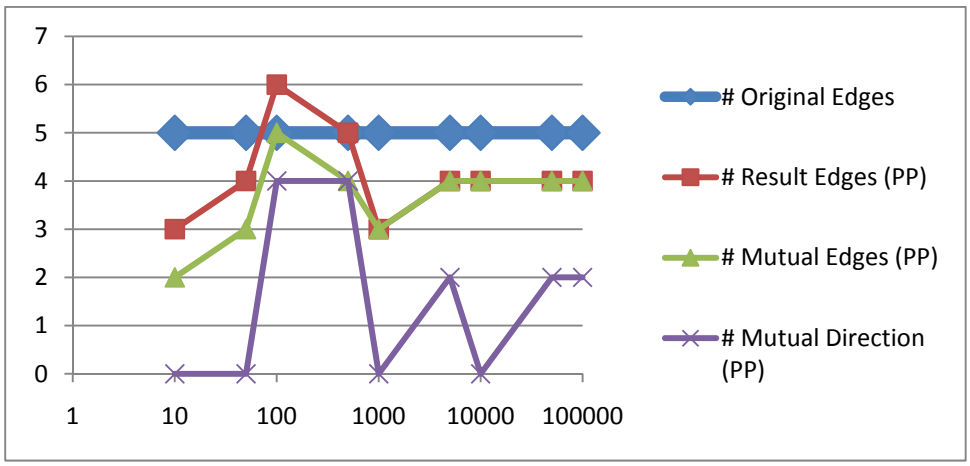

7
6
5
4
3
2
1
0
1
10
100
1000
10000
100000
Original Edges
Result Edges (BSe)
Mutual Edges (BSe)
Mutual Direction (BSe)

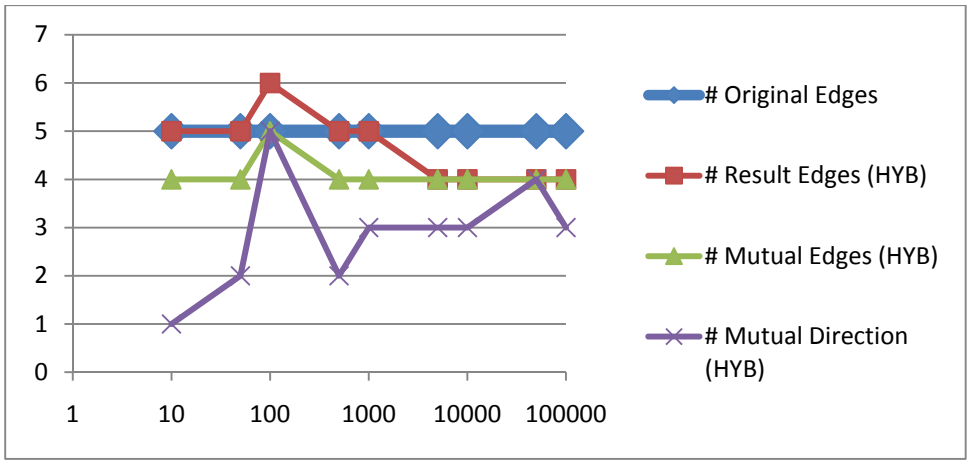

7
6
5
4
3
2
1
0
1
10
100
1000
10000
100000
Original Edges
Result Edges (PP)
Mutual Edges (PP)
Mutual Direction (PP)


7
6
5
4
3
2
1
0
1
10
100
1000
10000
100000
Original Edges
Result Edges (HYB)
Mutual Edges (HYB)
Mutual Direction (HYB)


7
6
5
4
3
2
1
0
1
10
100
1000
10000
100000
Original Edges
Result Edges (MDL)
Mutual Edges (MDL)
Mutual Direction (MDL)


7
6
5
4
3
2
1
0
1
10
100
1000
10000
100000
Original Edges
Result Edges (BSe)
Mutual Edges (BSe)
Mutual Direction (BSe)


7
6
5
4
3
2
1
0
1
10
100
1000
10000
100000
Original Edges
Result Edges (PP)
Mutual Edges (PP)
Mutual Direction (PP)



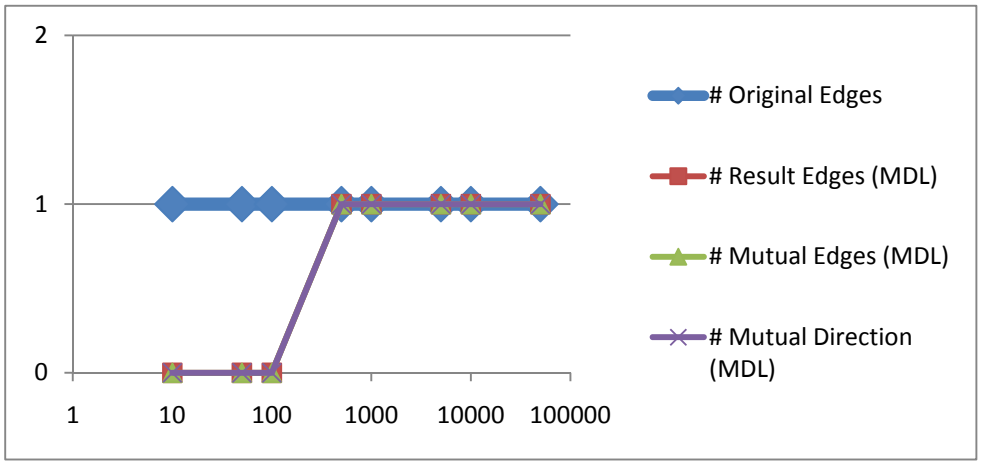
EXPERIMENT 2

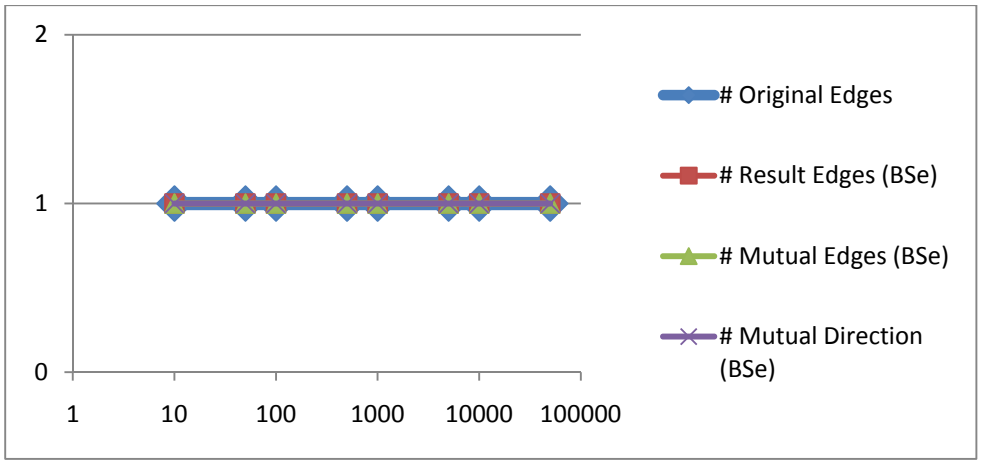
In the graphs below, the vertical axis represents the number of sentences in the database and the horizontal axis represents the number of edges.

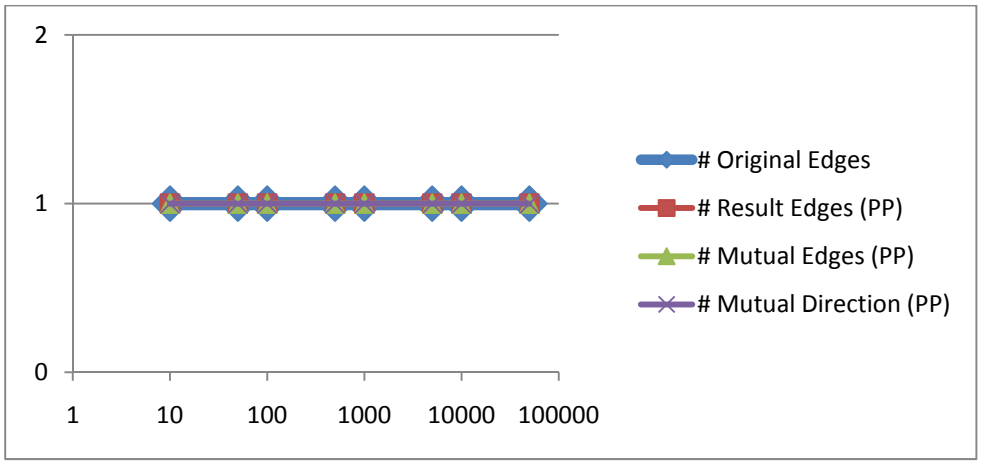
The acronyms in the graphs represent the following metrics:

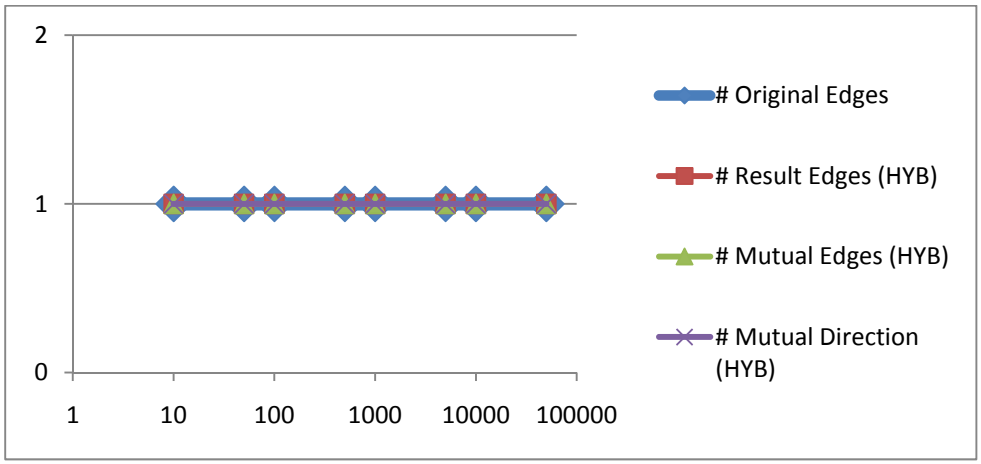
- MDL = the MDL Metric
- BSe = the BSe Metric
- PP = the Perplexity Metric
- HYB = the Modified Metric.

BN 1

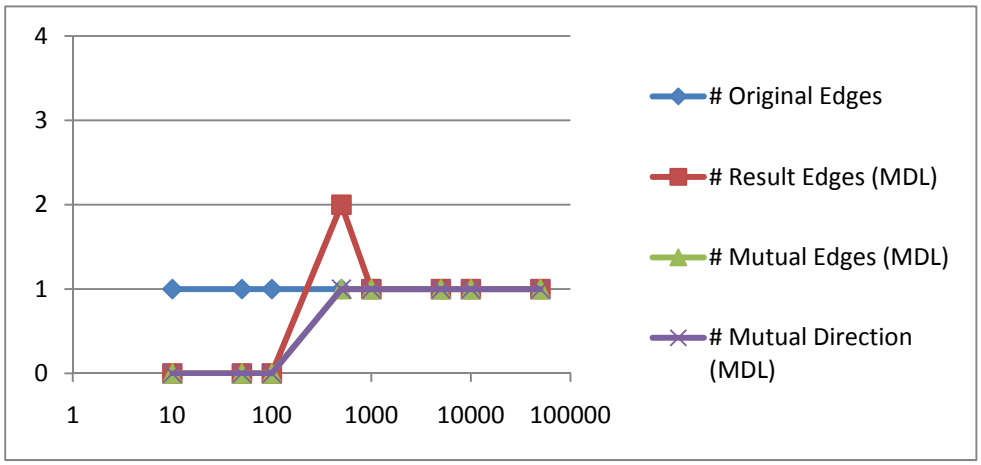


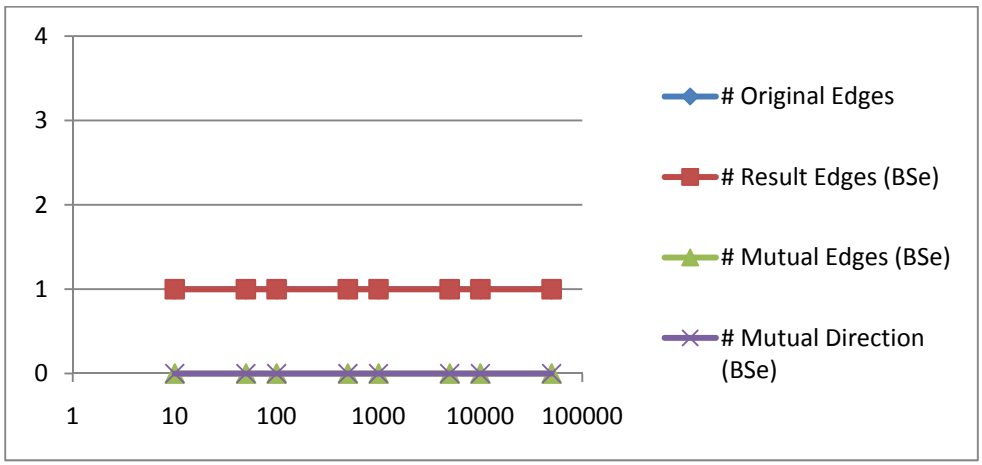

2
1
0
1
10
100
1000
10000
100000
Original Edges
Result Edges (BSe)
Mutual Edges (BSe)
Mutual Direction (BSe)

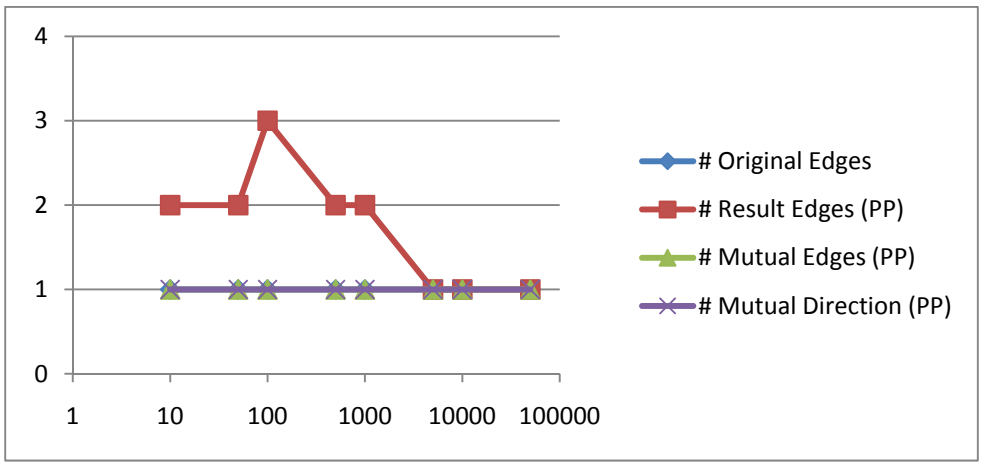

2
1
0
1
10
100
1000
10000
100000
Original Edges
Result Edges (PP)
Mutual Edges (PP)
Mutual Direction (PP)

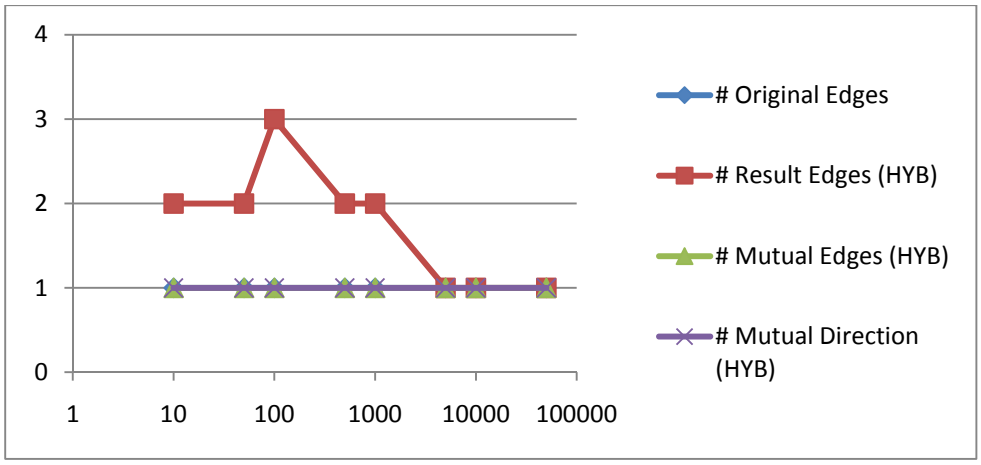

2
1
0
1
10
100
1000
10000
100000
Original Edges
Result Edges (HYB)
Mutual Edges (HYB)
Mutual Direction (HYB)

BN 2

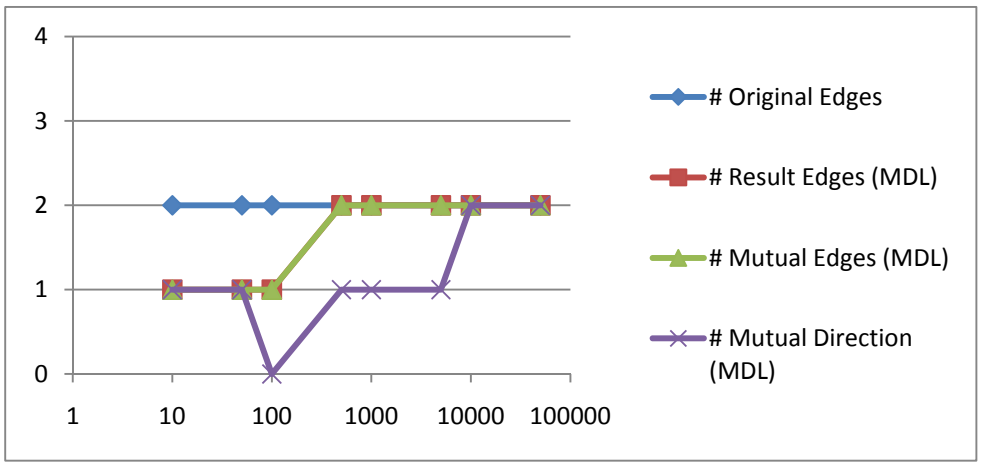

4
3
2
1
0
1
10
100
1000
10000
100000
Original Edges
Result Edges (MDL)
Mutual Edges (MDL)
Mutual Direction (MDL)

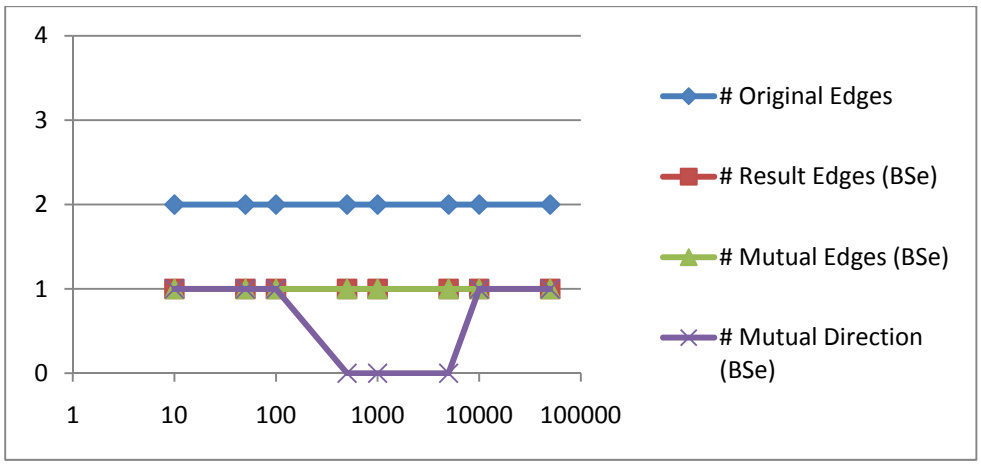

4
3
2
1
0
1
10
100
1000
10000
100000
Original Edges
Result Edges (BSe)
Mutual Edges (BSe)
Mutual Direction (BSe)

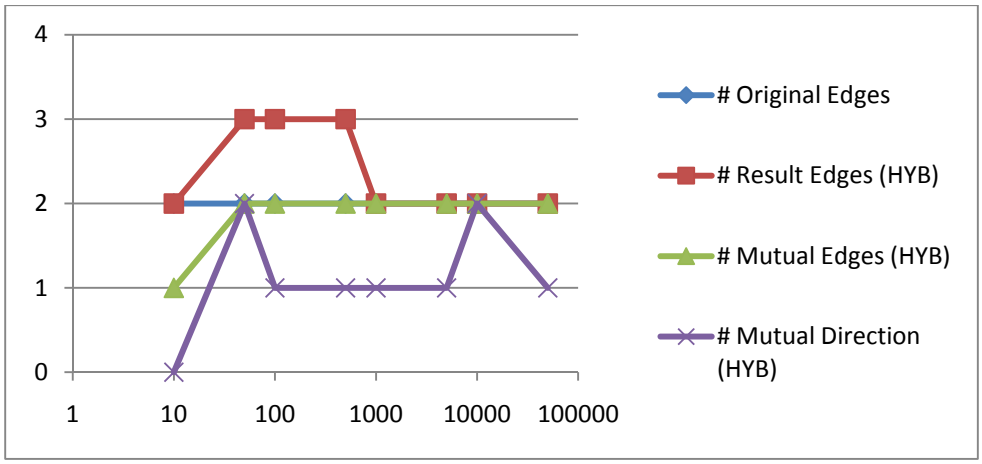

4
3
2
1
0
1
10
100
1000
10000
100000
Original Edges
Result Edges (PP)
Mutual Edges (PP)
Mutual Direction (PP)

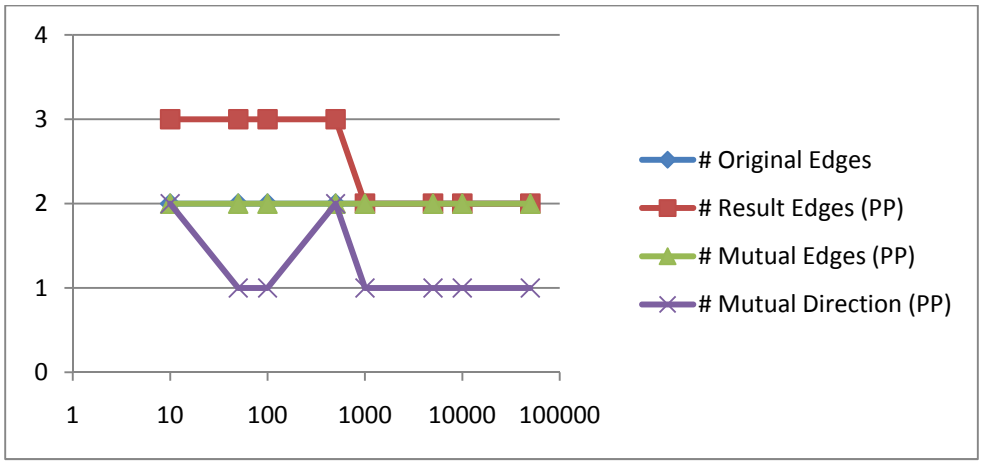

4
3
2
1
0
1
10
100
1000
10000
100000
Original Edges
Result Edges (HYB)
Mutual Edges (HYB)
Mutual Direction (HYB)

BN 3

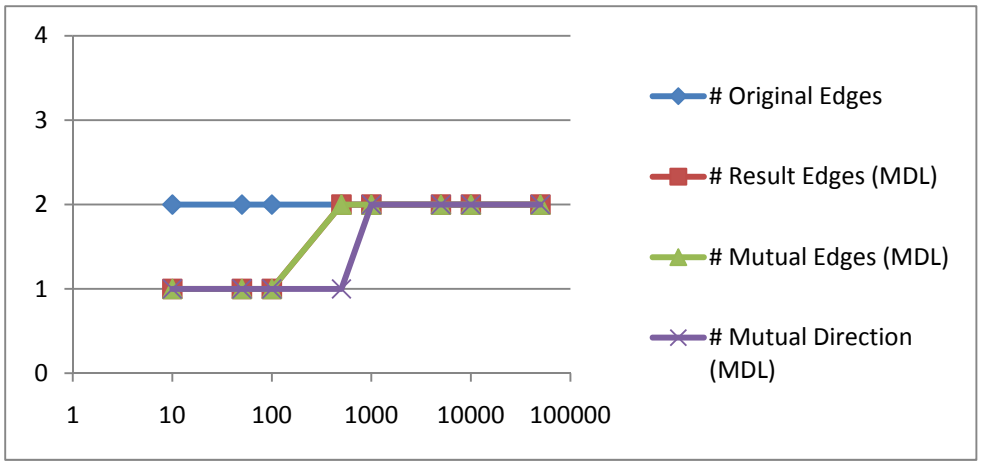

4
3
2
1
0
1
10
100
1000
10000
100000
Original Edges
Result Edges (MDL)
Mutual Edges (MDL)
Mutual Direction (MDL)

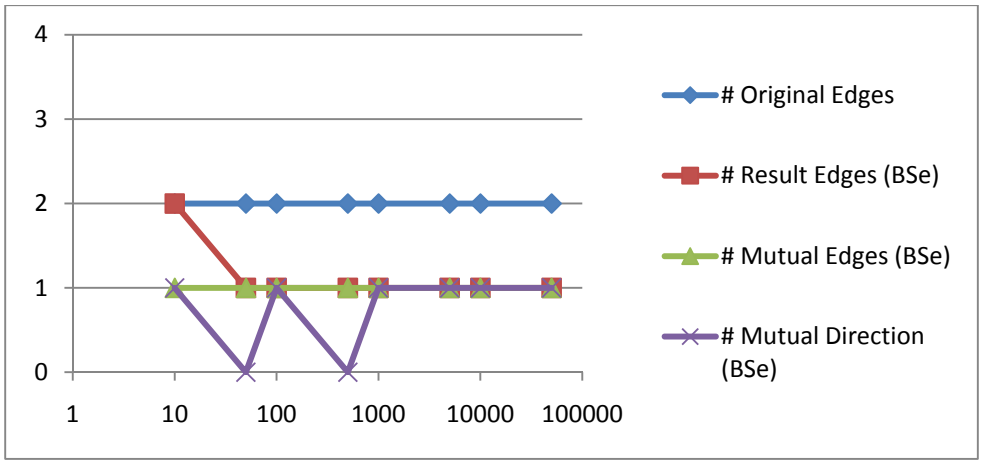

4
3
2
1
0
1
10
100
1000
10000
100000
Original Edges
Result Edges (BSe)
Mutual Edges (BSe)
Mutual Direction (BSe)

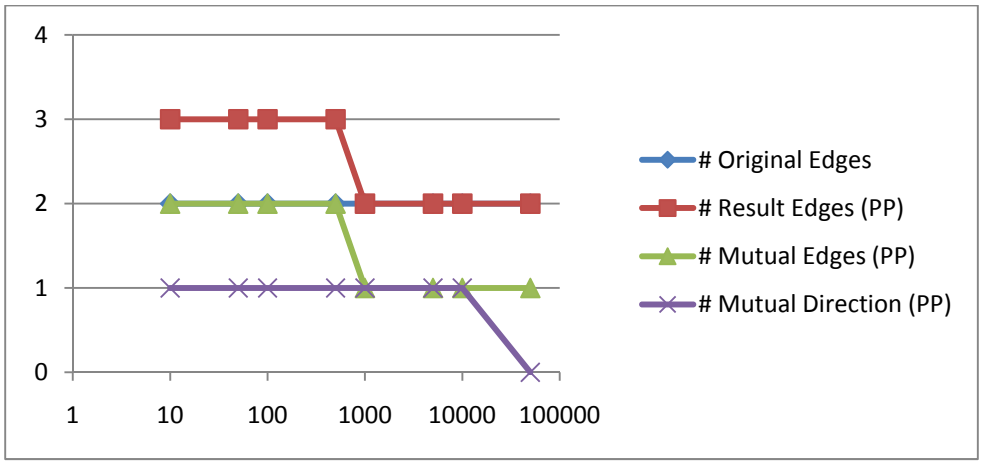

4
3
2
1
0
1
10
100
1000
10000
100000
Original Edges
Result Edges (HYB)
Mutual Edges (HYB)
Mutual Direction (HYB)

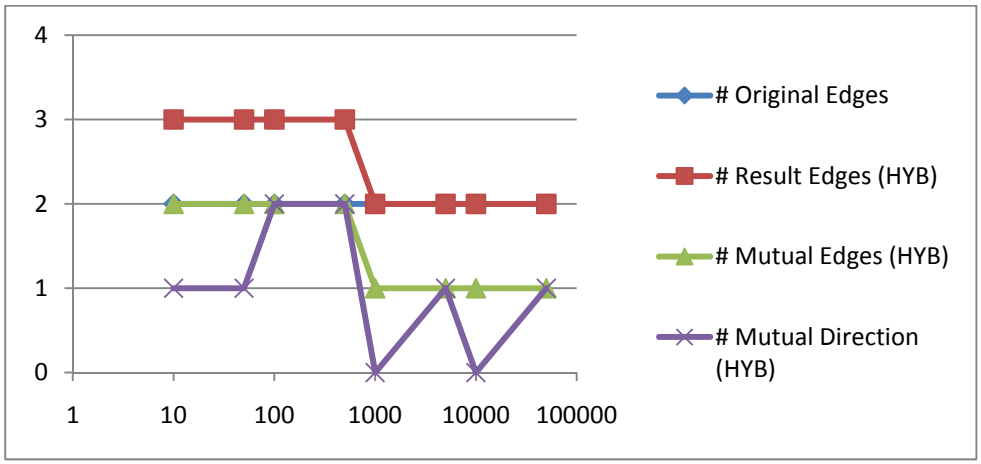

4
3
2
1
0
1
10
100
1000
10000
100000
Original Edges
Result Edges (PP)
Mutual Edges (PP)
Mutual Direction (PP)

BN 4

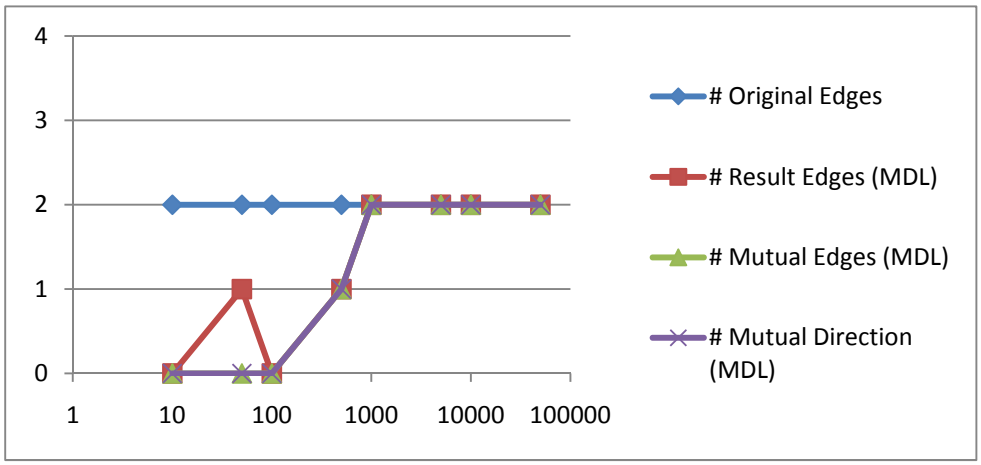

4
3
2
1
0
1
10
100
1000
10000
100000
Original Edges
Result Edges (MDL)
Mutual Edges (MDL)
Mutual Direction (MDL)

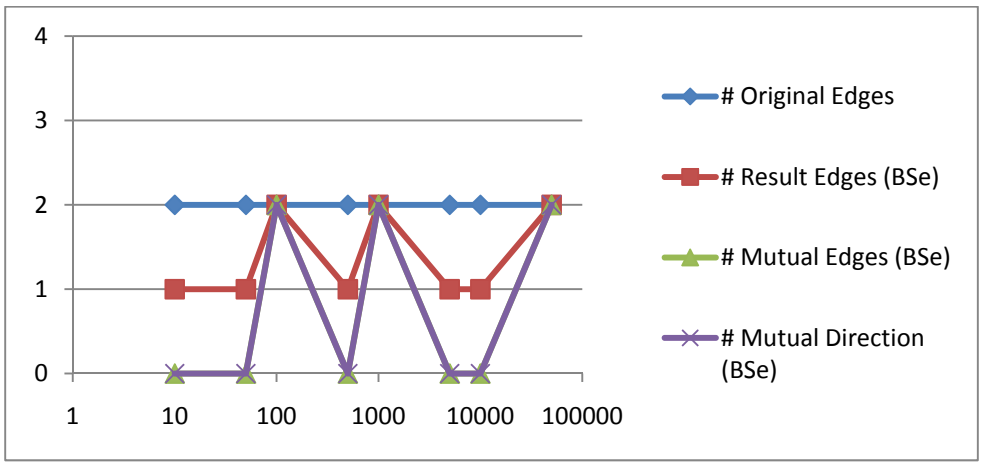

4
3
2
1
0
1
10
100
1000
10000
100000
Original Edges
Result Edges (BSe)
Mutual Edges (BSe)
Mutual Direction (BSe)

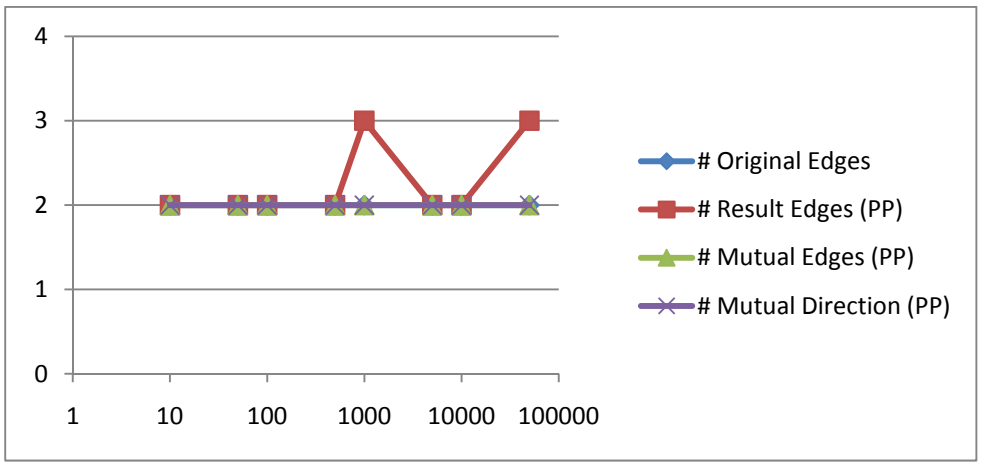

4
3
2
1
0
1
10
100
1000
10000
100000
Original Edges
Result Edges (PP)
Mutual Edges (PP)
Mutual Direction (PP)

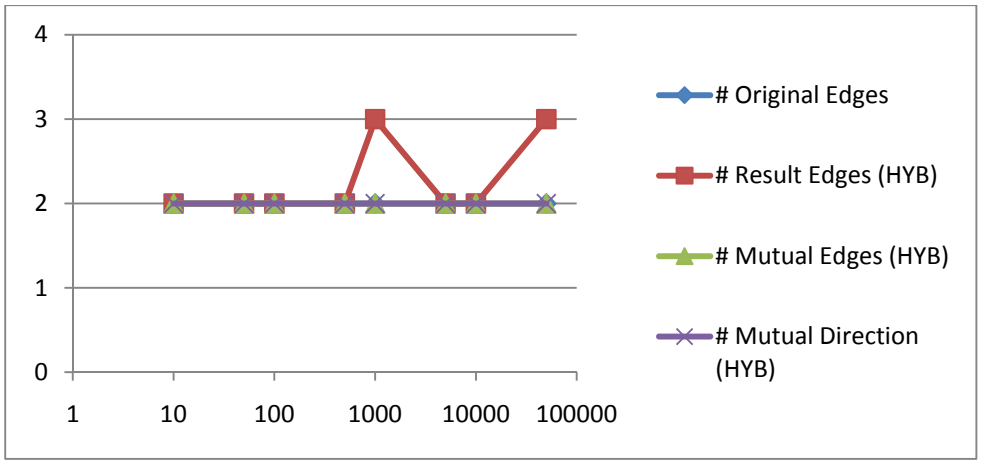

4
3
2
1
0
1
10
100
1000
10000
100000
Original Edges
Result Edges (HYB)
Mutual Edges (HYB)
Mutual Direction (HYB)

BN 5

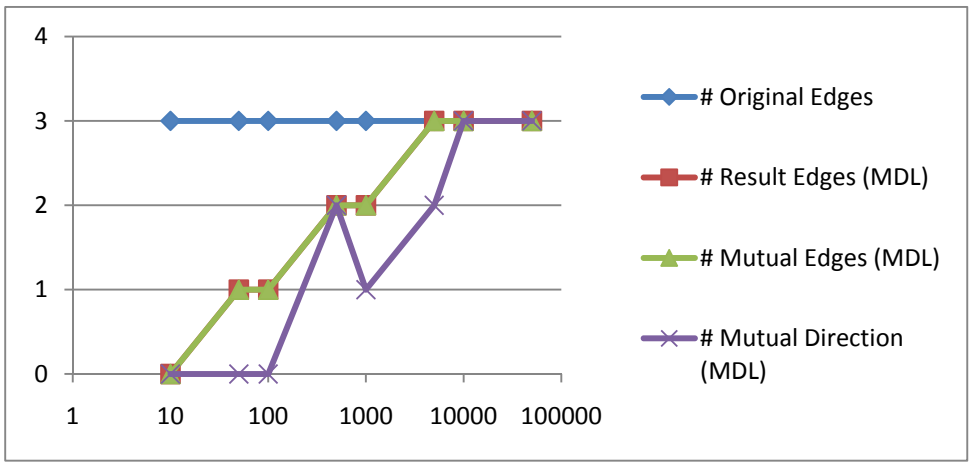

4
3
2
1
0
1
10
100
1000
10000
100000
Original Edges
Result Edges (MDL)
Mutual Edges (MDL)
Mutual Direction (MDL)

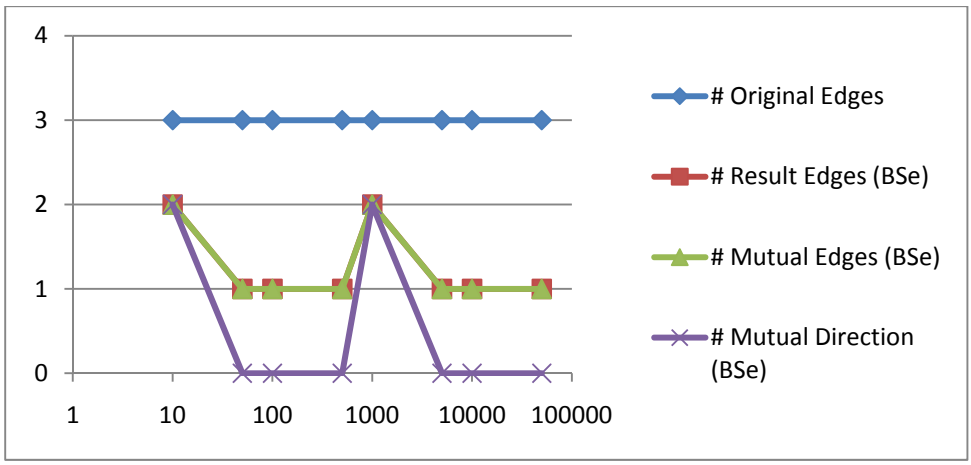

4
3
2
1
0
1
10
100
1000
10000
100000
Original Edges
Result Edges (BSe)
Mutual Edges (BSe)
Mutual Direction (BSe)

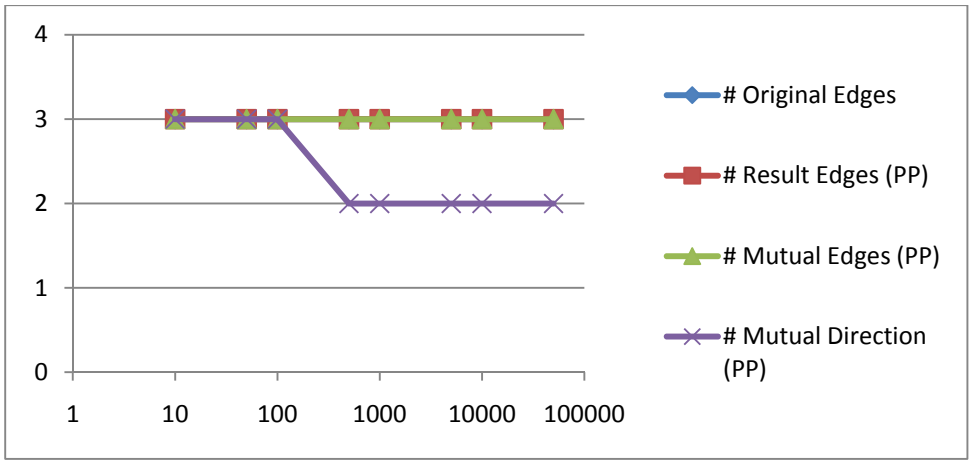

4
3
2
1
0
1
10
100
1000
10000
100000
Original Edges
Result Edges (PP)
Mutual Edges (PP)
Mutual Direction (PP)

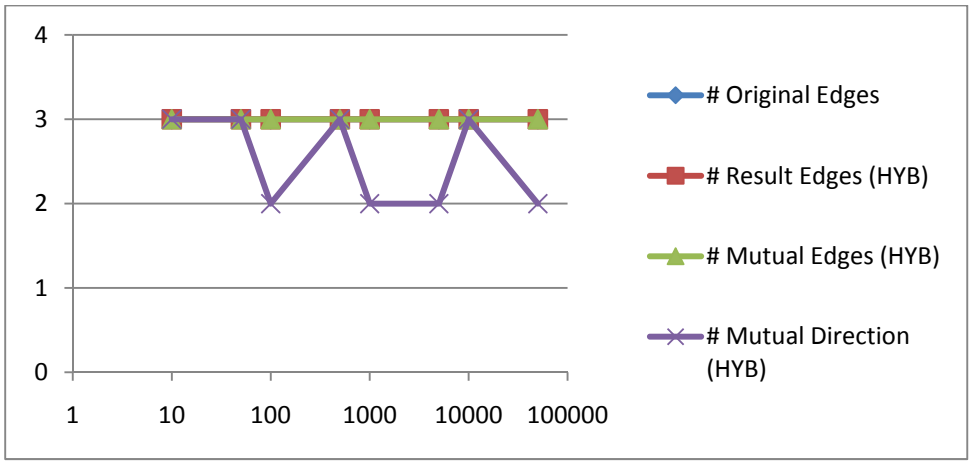

4
3
2
1
0
1
10
100
1000
10000
100000
Original Edges
Result Edges (HYB)
Mutual Edges (HYB)
Mutual Direction (HYB)

BN 6

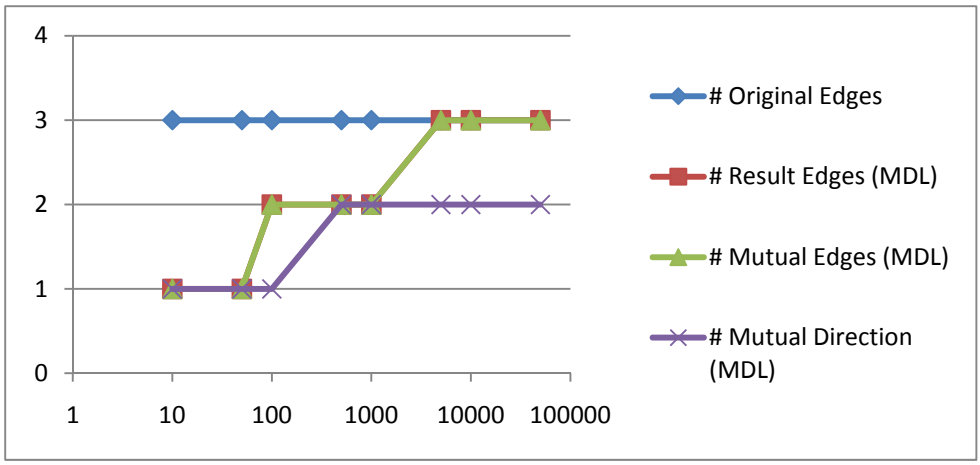

4
3
2
1
0
1
10
100
1000
10000
100000
Original Edges
Result Edges (MDL)
Mutual Edges (MDL)
Mutual Direction (MDL)

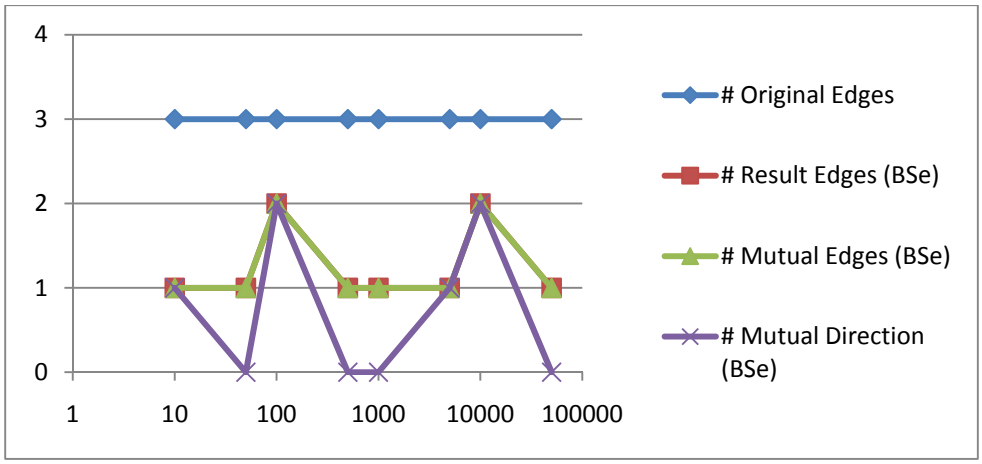

4
3
2
1
0
1
10
100
1000
10000
100000
Original Edges
Result Edges (BSe)
Mutual Edges (BSe)
Mutual Direction (BSe)

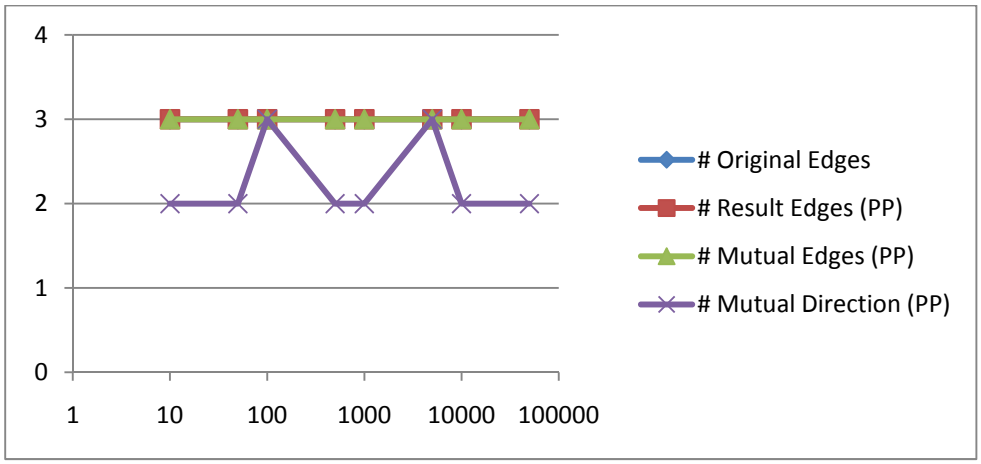

4
3
2
1
0
1
10
100
1000
10000
100000
Original Edges
Result Edges (PP)
Mutual Edges (PP)
Mutual Direction (PP)

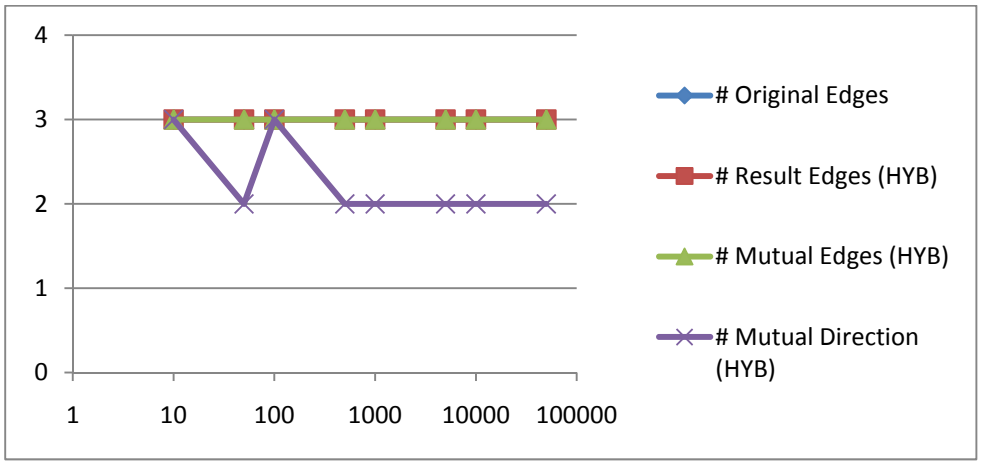

4
3
2
1
0
1
10
100
1000
10000
100000
Original Edges
Result Edges (HYB)
Mutual Edges (HYB)
Mutual Direction (HYB)

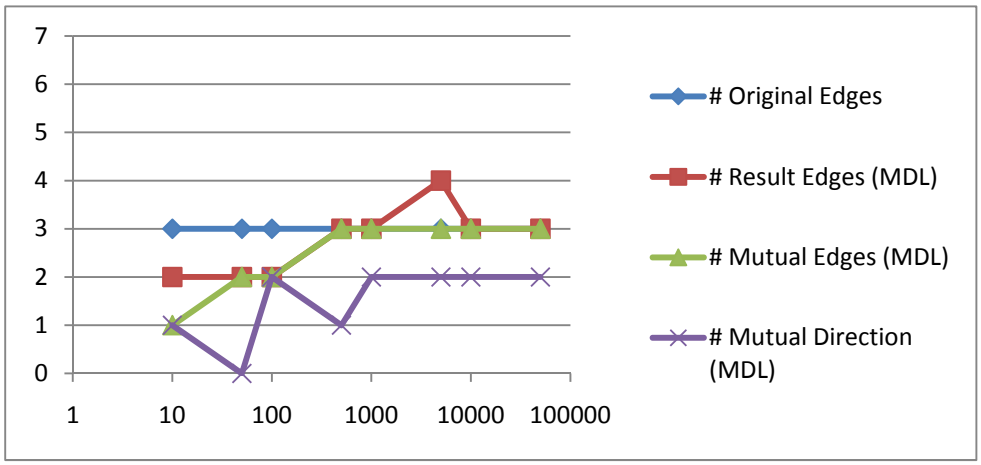
BN 7

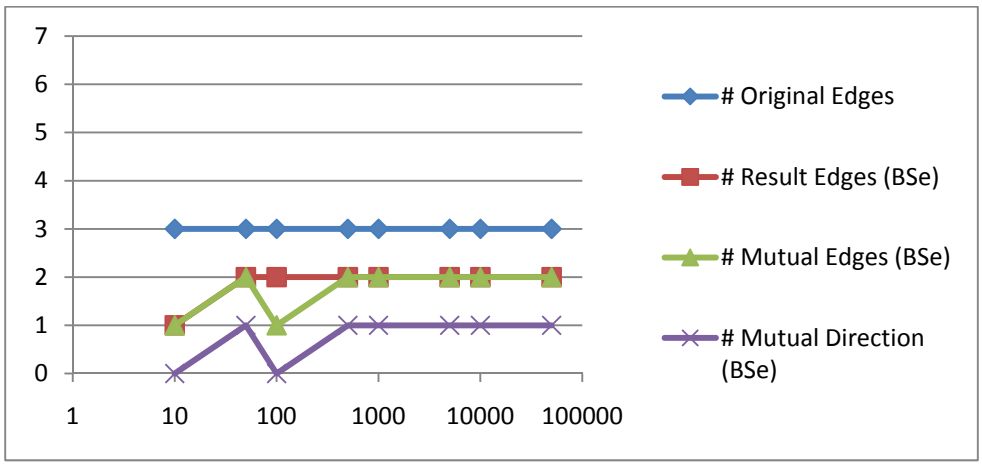

4
3
2
1
0
1
10
100
1000
10000
100000
Original Edges
Result Edges (MDL)
Mutual Edges (MDL)
Mutual Direction (MDL)

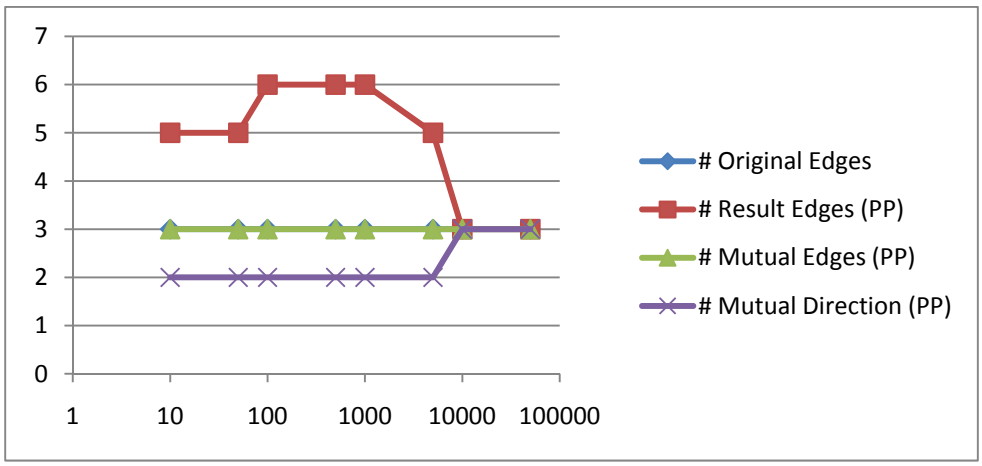

4
3
2
1
0
1
10
100
1000
10000
100000
Original Edges
Result Edges (BSe)
Mutual Edges (BSe)
Mutual Direction (BSe)

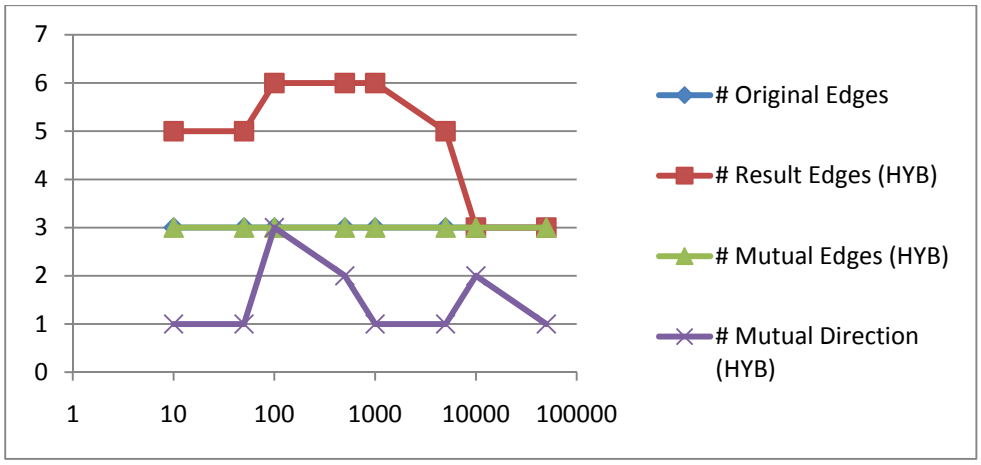

4
3
2
1
0
1
10
100
1000
10000
100000
Original Edges
Result Edges (PP)
Mutual Edges (PP)
Mutual Direction (PP)


4
3
2
1
0
1
10
100
1000
10000
100000
Original Edges
Result Edges (HYB)
Mutual Edges (HYB)
Mutual Direction (HYB)

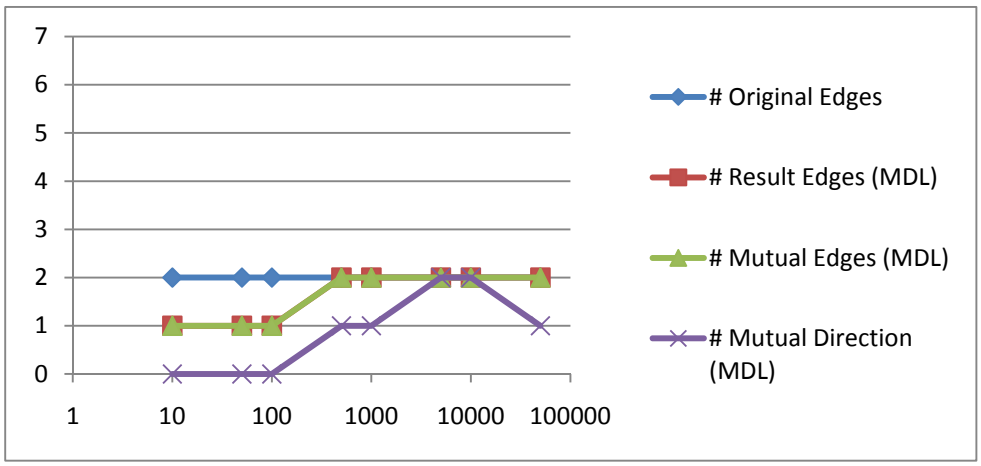

7
6
5
4
3
2
1
0
1
10
100
1000
10000
100000
Original Edges
Result Edges (MDL)
Mutual Edges (MDL)
Mutual Direction (MDL)

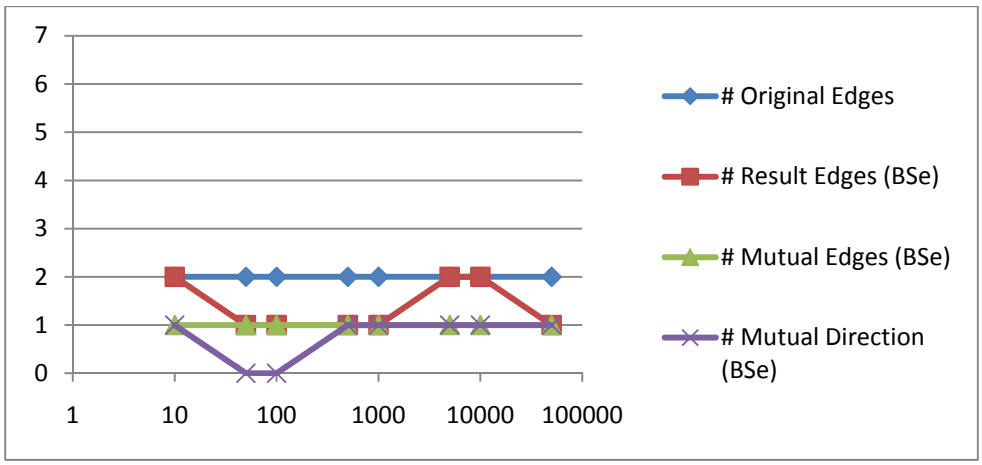

7
6
5
4
3
2
1
0
1
10
100
1000
10000
100000
Original Edges
Result Edges (BSe)
Mutual Edges (BSe)
Mutual Direction (BSe)

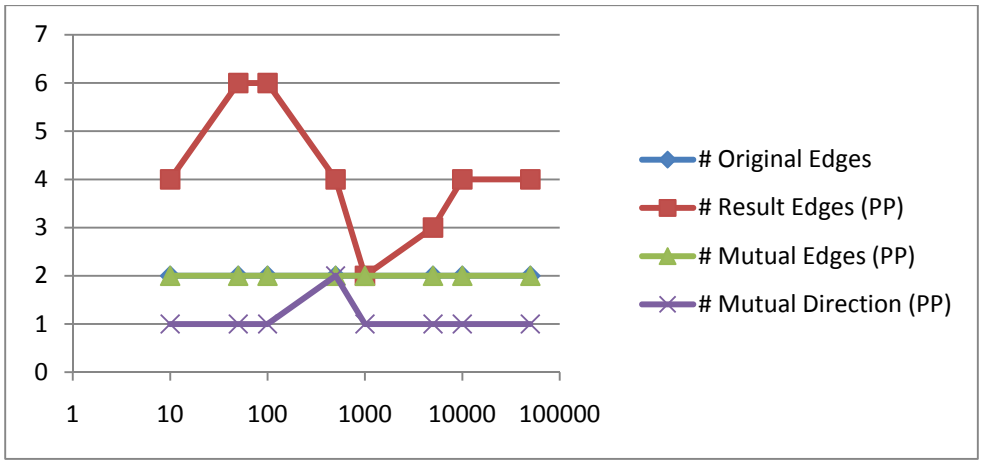

7
6
5
4
3
2
1
0
1
10
100
1000
10000
100000
Original Edges
Result Edges (PP)
Mutual Edges (PP)
Mutual Direction (PP)

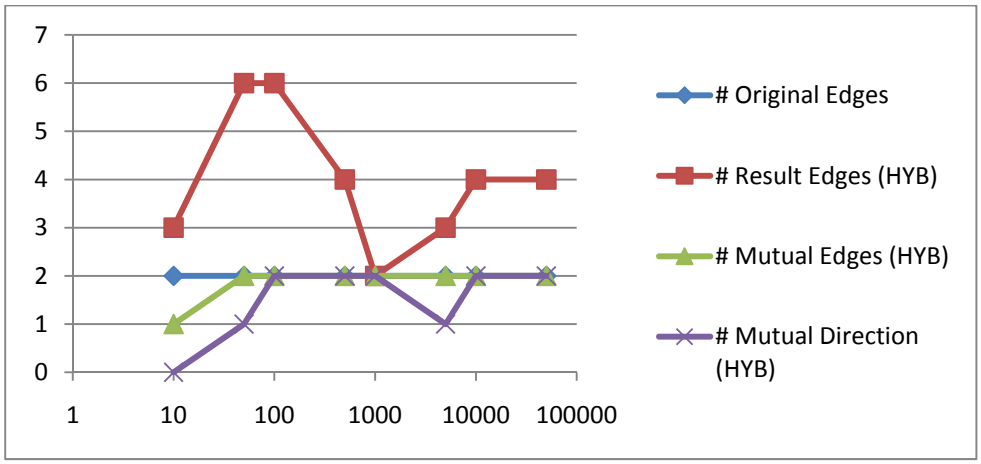

7
6
5
4
3
2
1
0
1
10
100
1000
10000
100000
Original Edges
Result Edges (HYB)
Mutual Edges (HYB)
Mutual Direction (HYB)

BN 9

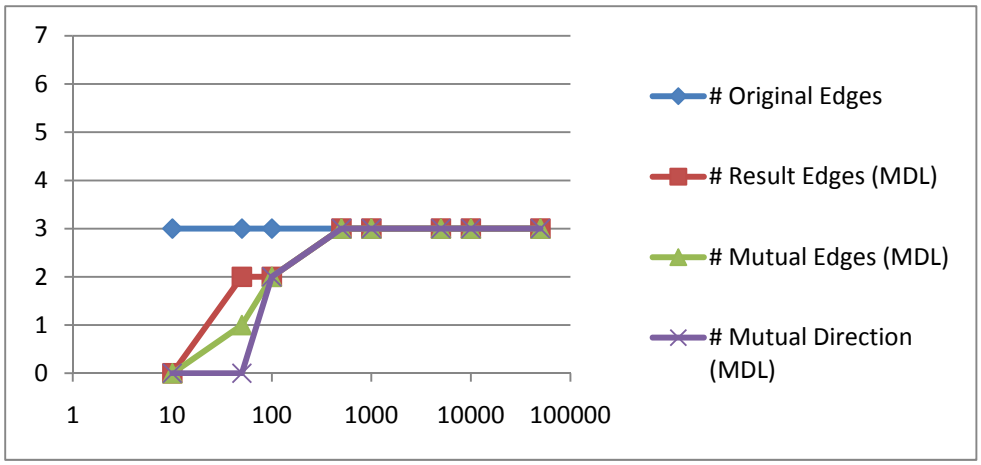

7
6
5
4
3
2
1
0
1
10
100
1000
10000
100000
Original Edges
Result Edges (MDL)
Mutual Edges (MDL)
Mutual Direction (MDL)

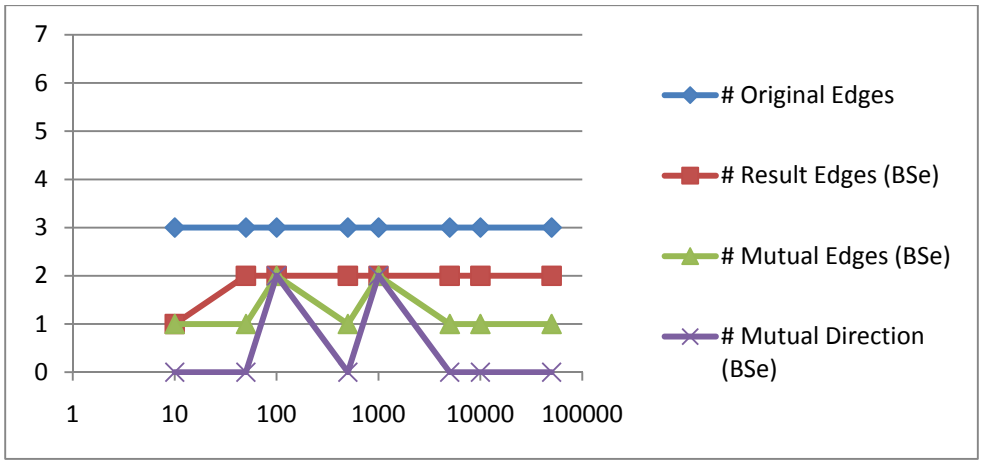

7
6
5
4
3
2
1
0
1
10
100
1000
10000
100000
Original Edges
Result Edges (BSe)
Mutual Edges (BSe)
Mutual Direction (BSe)

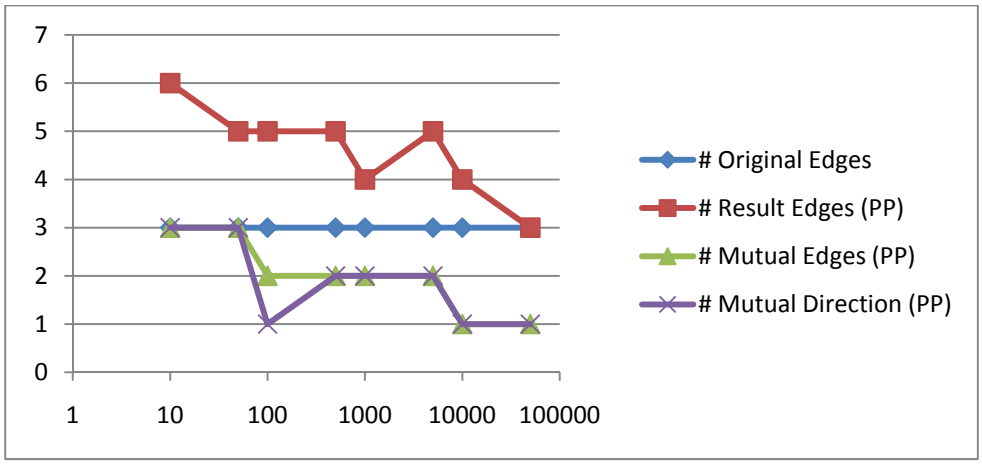

7
6
5
4
3
2
1
0
1
10
100
1000
10000
100000
Original Edges
Result Edges (PP)
Mutual Edges (PP)
Mutual Direction (PP)

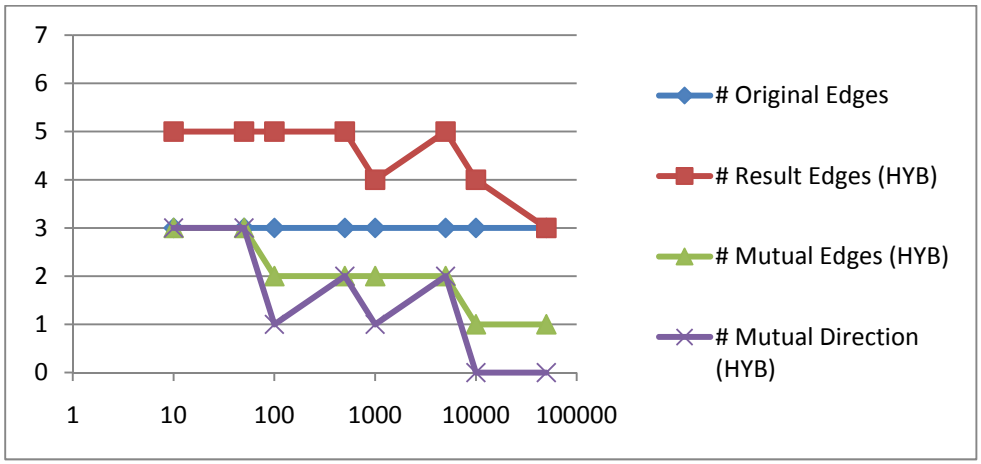

7
6
5
4
3
2
1
0
1
10
100
1000
10000
100000
Original Edges
Result Edges (HYB)
Mutual Edges (HYB)
Mutual Direction (HYB)

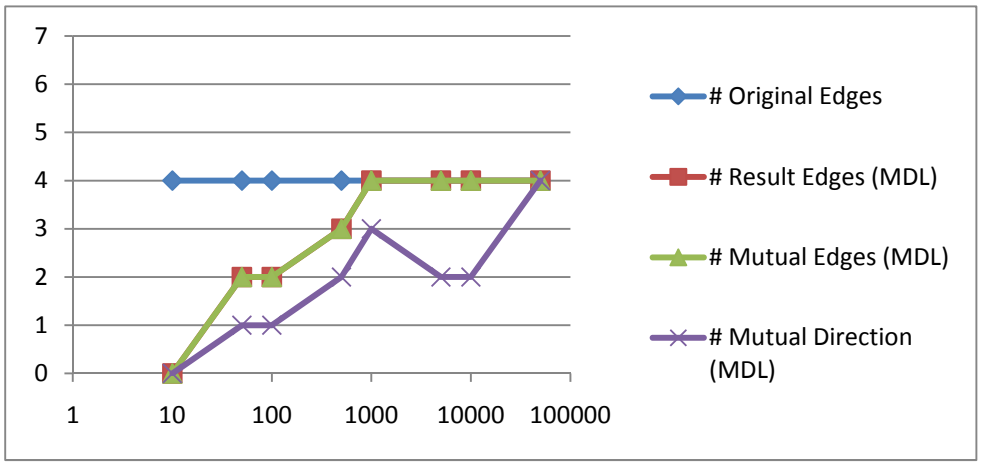
BN 10

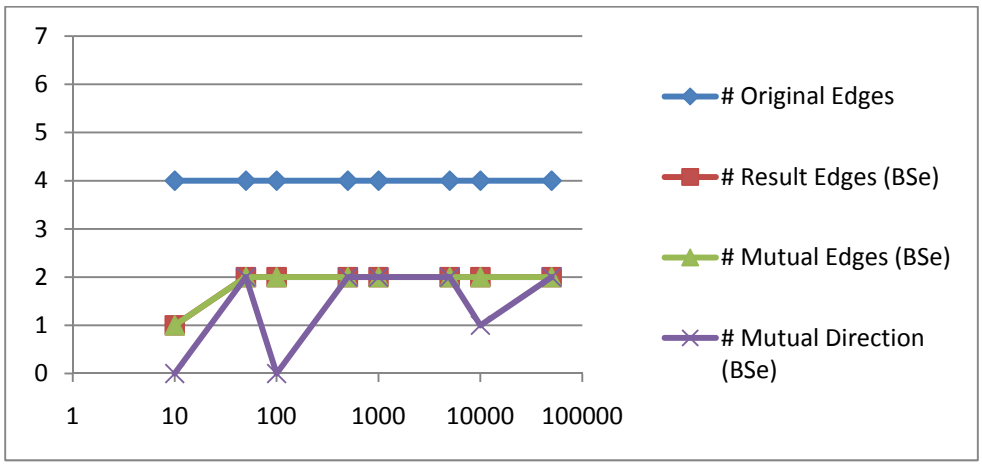

7
6
5
4
3
2
1
0
1
10
100
1000
10000
100000
Original Edges
Result Edges (MDL)
Mutual Edges (MDL)
Mutual Direction (MDL)

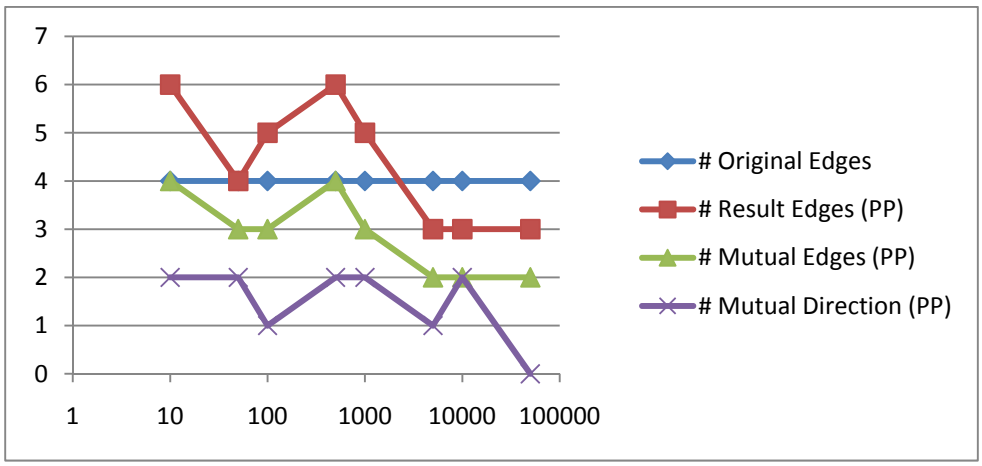

7
6
5
4
3
2
1
0
1
10
100
1000
10000
100000
Original Edges
Result Edges (BSe)
Mutual Edges (BSe)
Mutual Direction (BSe)

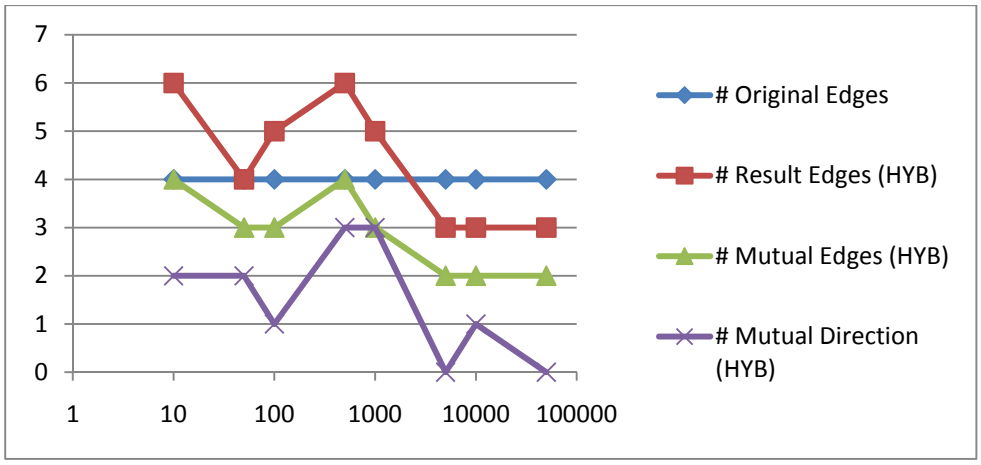

7
6
5
4
3
2
1
0
1
10
100
1000
10000
100000
Original Edges
Result Edges (PP)
Mutual Edges (PP)
Mutual Direction (PP)


7
6
5
4
3
2
1
0
1
10
100
1000
10000
100000
Original Edges
Result Edges (HYB)
Mutual Edges (HYB)
Mutual Direction (HYB)

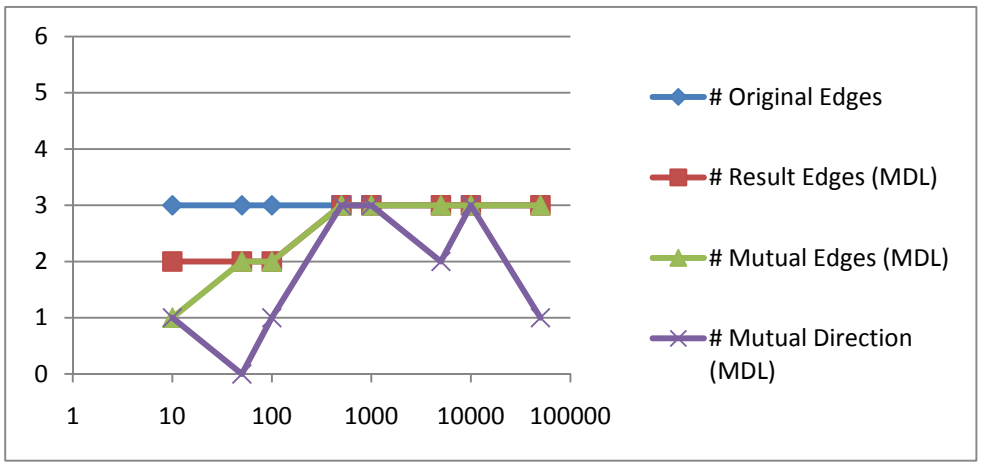

7
6
5
4
3
2
1
0
1
10
100
1000
10000
100000
Original Edges
Result Edges (MDL)
Mutual Edges (MDL)
Mutual Direction (MDL)

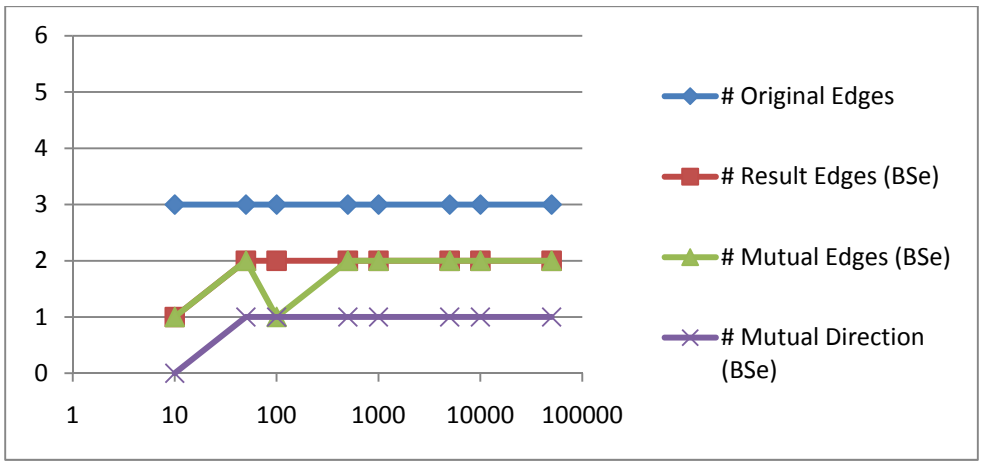

7
6
5
4
3
2
1
0
1
10
100
1000
10000
100000
Original Edges
Result Edges (BSe)
Mutual Edges (BSe)
Mutual Direction (BSe)

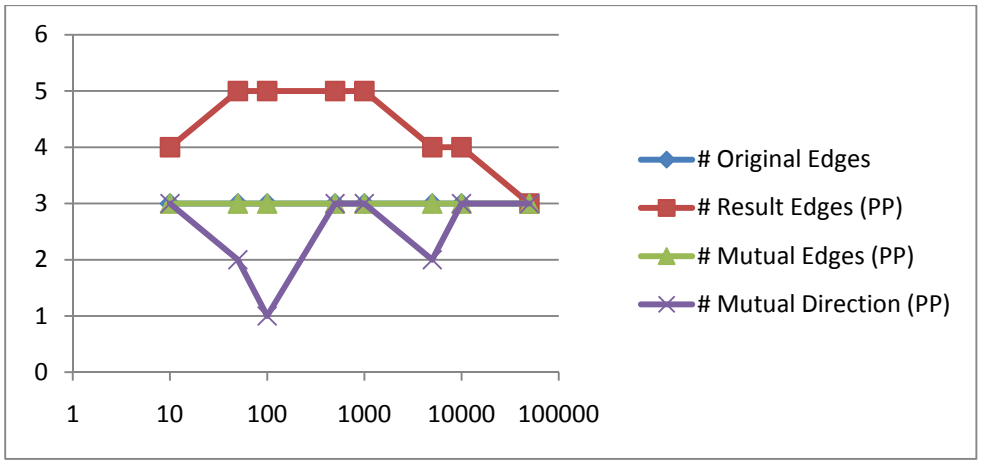

7
6
5
4
3
2
1
0
1
10
100
1000
10000
100000
Original Edges
Result Edges (PP)
Mutual Edges (PP)
Mutual Direction (PP)

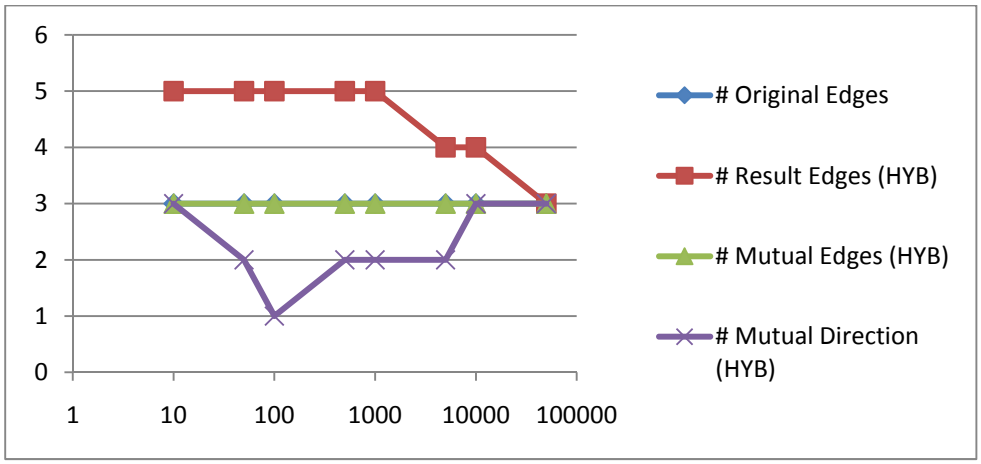

7
6
5
4
3
2
1
0
1
10
100
1000
10000
100000
Original Edges
Result Edges (HYB)
Mutual Edges (HYB)
Mutual Direction (HYB)

BN 12

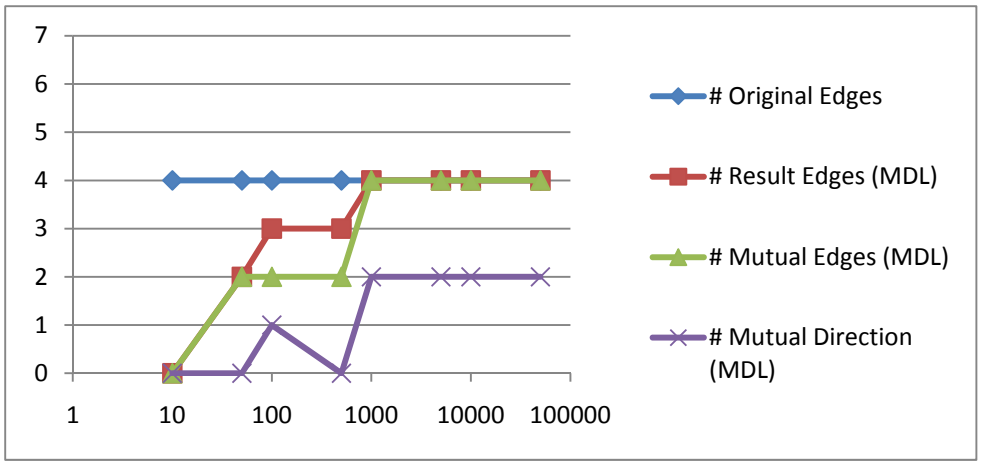

6
5
4
3
2
1
0
1
10
100
1000
10000
100000
Original Edges
Result Edges (MDL)
Mutual Edges (MDL)
Mutual Direction (MDL)

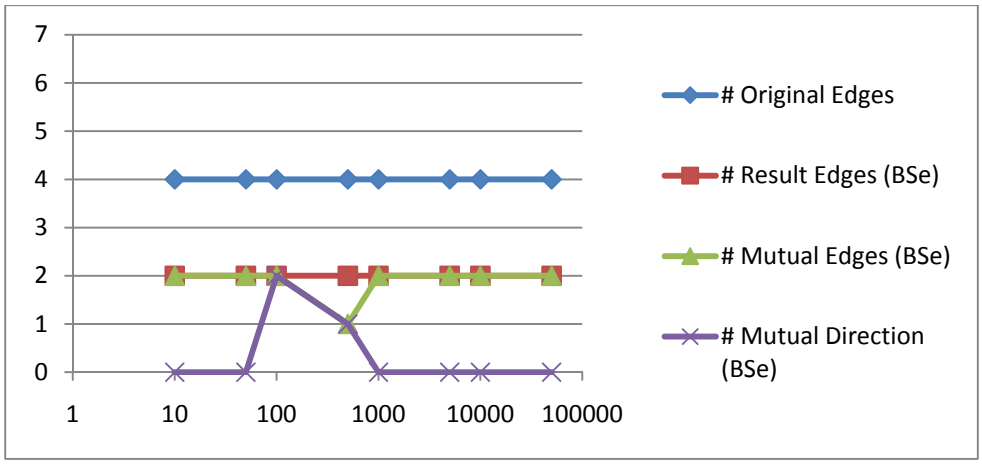

6
5
4
3
2
1
0
1
10
100
1000
10000
100000
Original Edges
Result Edges (BSe)
Mutual Edges (BSe)
Mutual Direction (BSe)

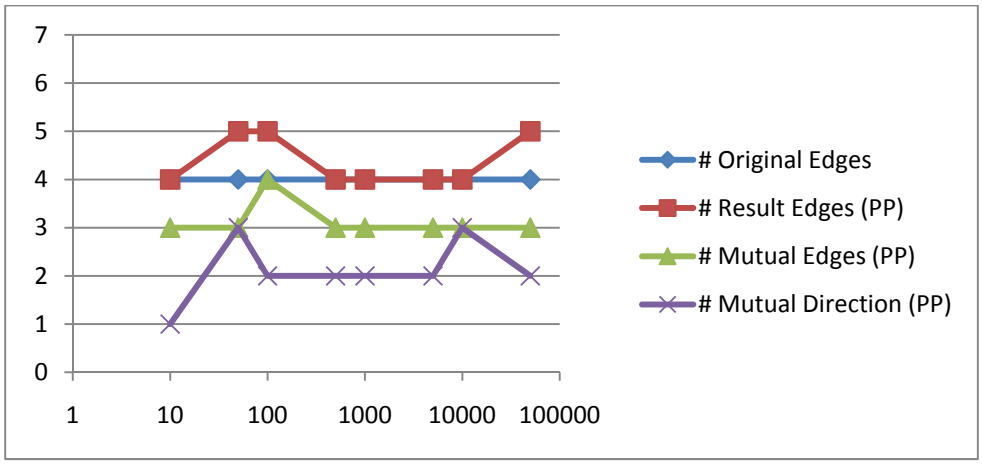

6
5
4
3
2
1
0
1
10
100
1000
10000
100000
Original Edges
Result Edges (PP)
Mutual Edges (PP)
Mutual Direction (PP)

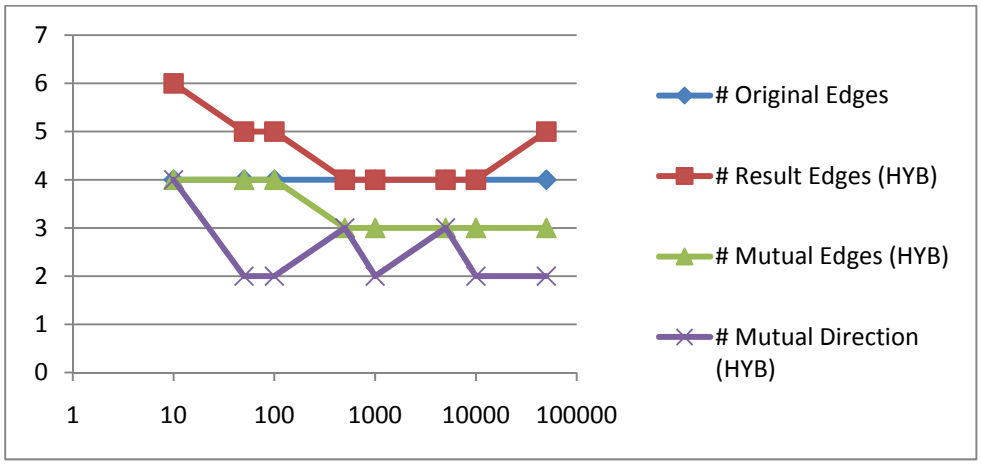

6
5
4
3
2
1
0
1
10
100
1000
10000
100000
Original Edges
Result Edges (HYB)
Mutual Edges (HYB)
Mutual Direction (HYB)

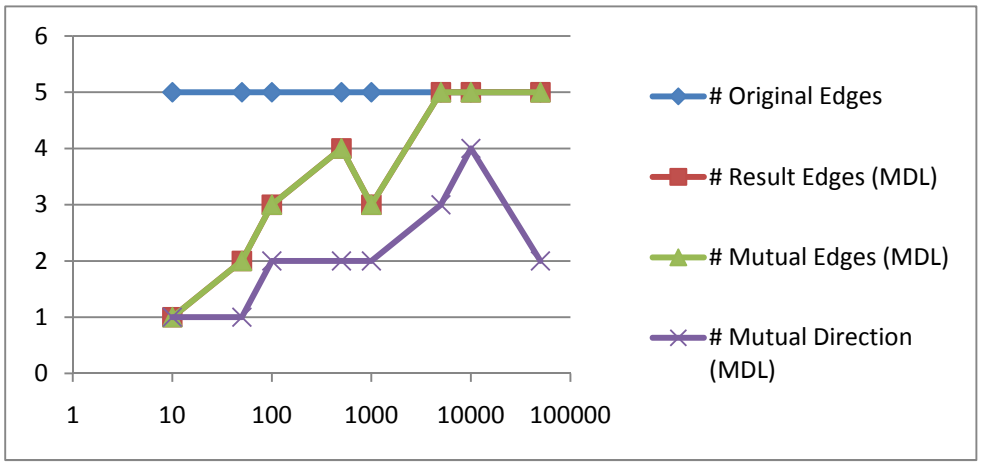
BN 13

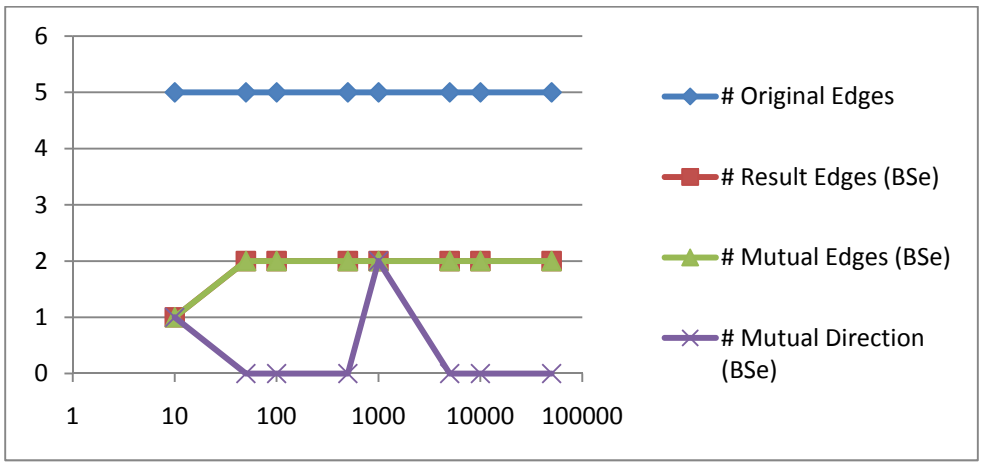

7
6
5
4
3
2
1
0
1
10
100
1000
10000
100000
Original Edges
Result Edges (MDL)
Mutual Edges (MDL)
Mutual Direction (MDL)

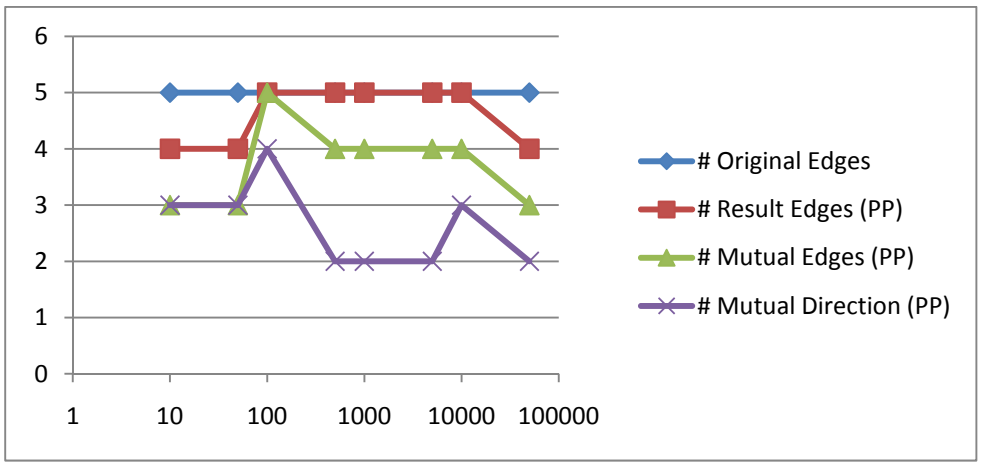

7
6
5
4
3
2
1
0
1
10
100
1000
10000
100000
Original Edges
Result Edges (BSe)
Mutual Edges (BSe)
Mutual Direction (BSe)

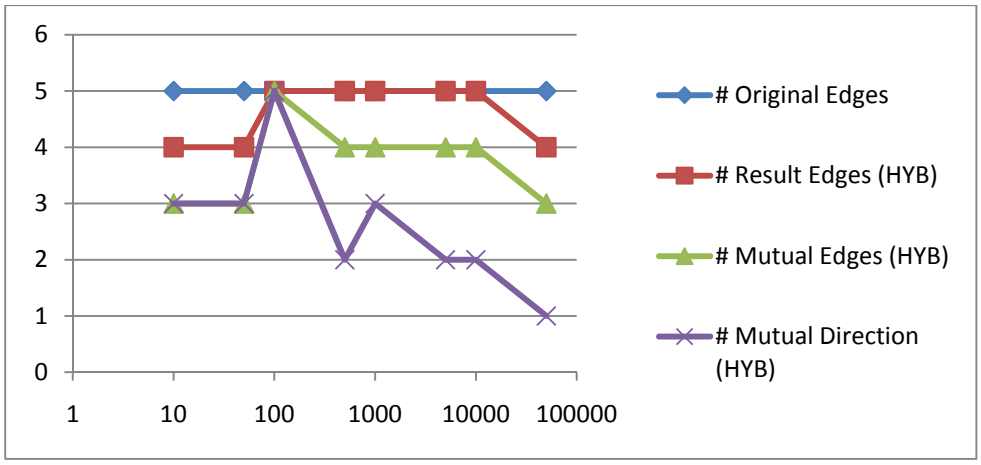

7
6
5
4
3
2
1
0
1
10
100
1000
10000
100000
Original Edges
Result Edges (PP)
Mutual Edges (PP)
Mutual Direction (PP)


7
6
5
4
3
2
1
0
1
10
100
1000
10000
100000
Original Edges
Result Edges (HYB)
Mutual Edges (HYB)
Mutual Direction (HYB)

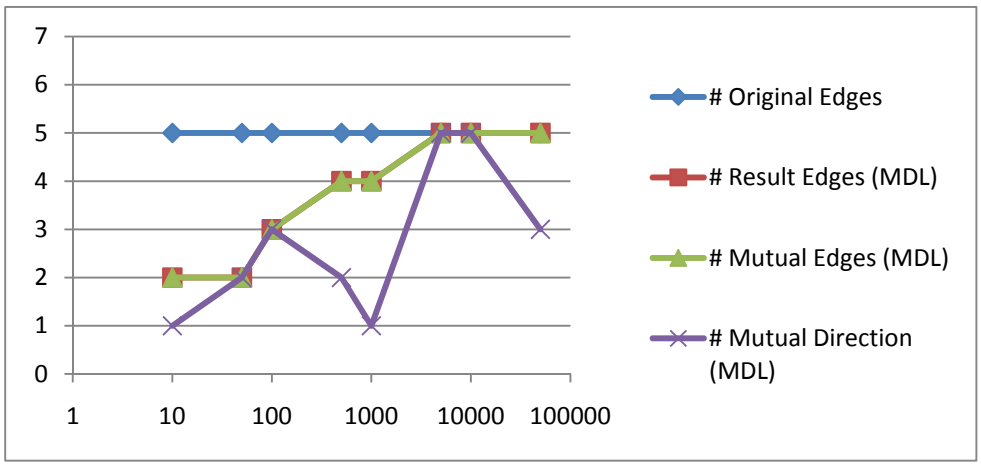

6
5
4
3
2
1
0
1
10
100
1000
10000
100000
Original Edges
Result Edges (MDL)
Mutual Edges (MDL)
Mutual Direction (MDL)

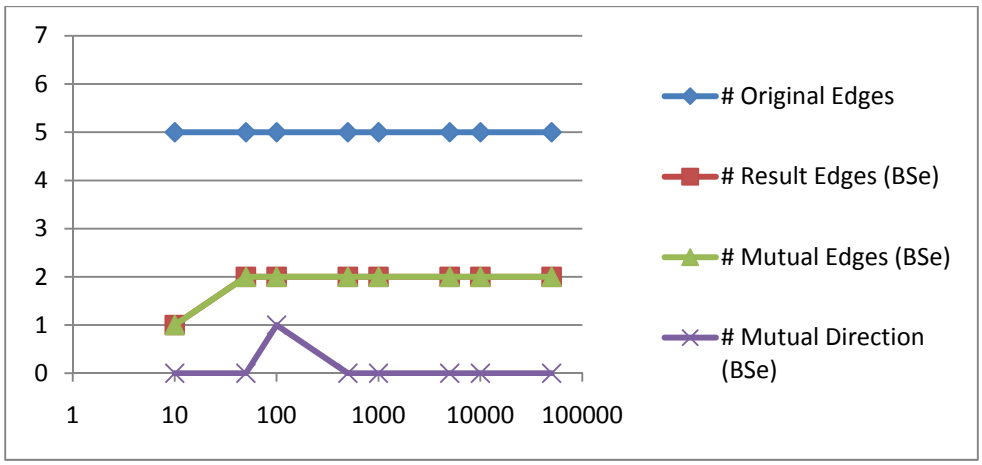

6
5
4
3
2
1
0
1
10
100
1000
10000
100000
Original Edges
Result Edges (BSe)
Mutual Edges (BSe)
Mutual Direction (BSe)

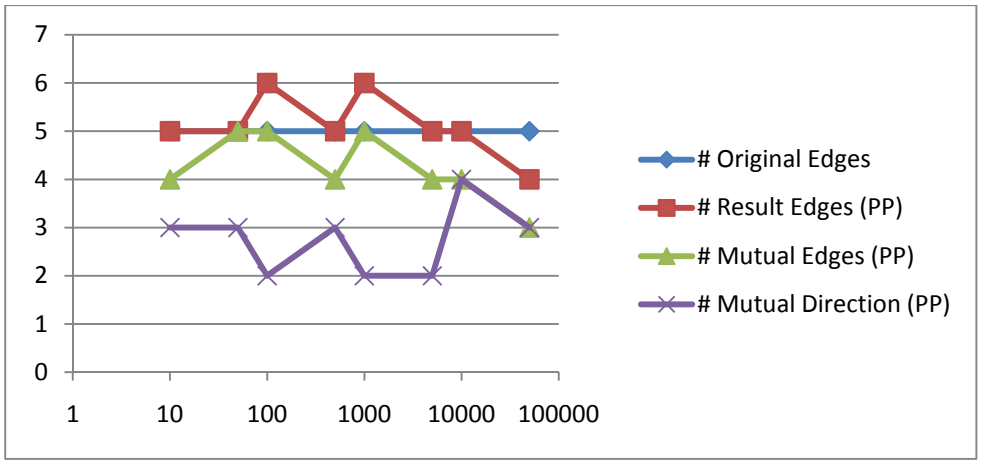

6
5
4
3
2
1
0
1
10
100
1000
10000
100000
Original Edges
Result Edges (PP)
Mutual Edges (PP)
Mutual Direction (PP)

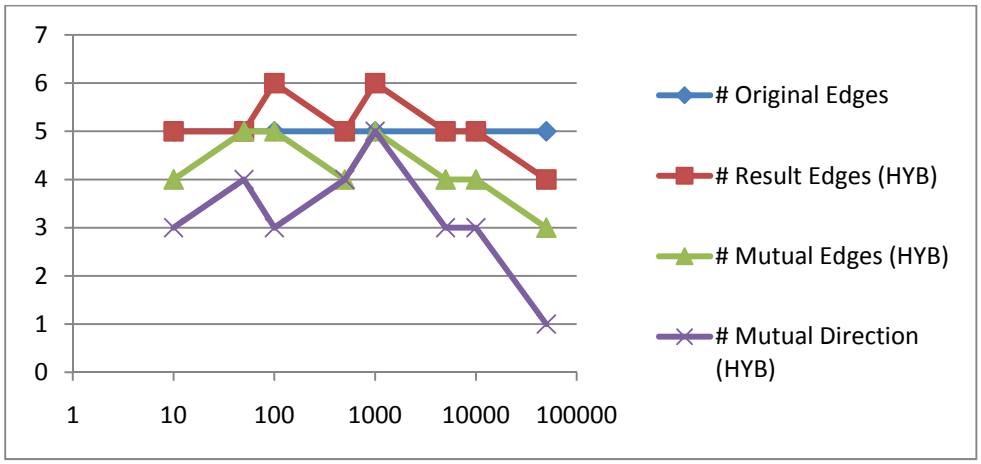

6
5
4
3
2
1
0
1
10
100
1000
10000
100000
Original Edges
Result Edges (HYB)
Mutual Edges (HYB)
Mutual Direction (HYB)

BN 15


7
6
5
4
3
2
1
0
1
10
100
1000
10000
100000
Original Edges
Result Edges (MDL)
Mutual Edges (MDL)
Mutual Direction (MDL)

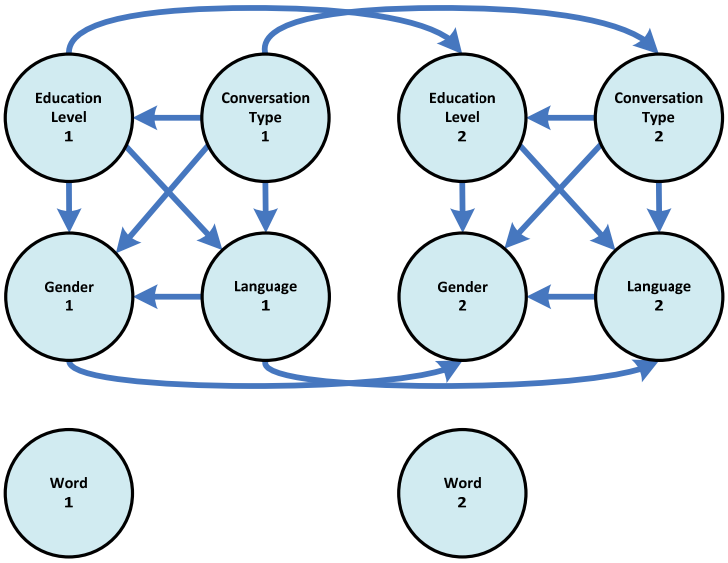

7
6
5
4
3
2
1
0
1
10
100
1000
10000
100000
Original Edges
Result Edges (BSe)
Mutual Edges (BSe)
Mutual Direction (BSe)





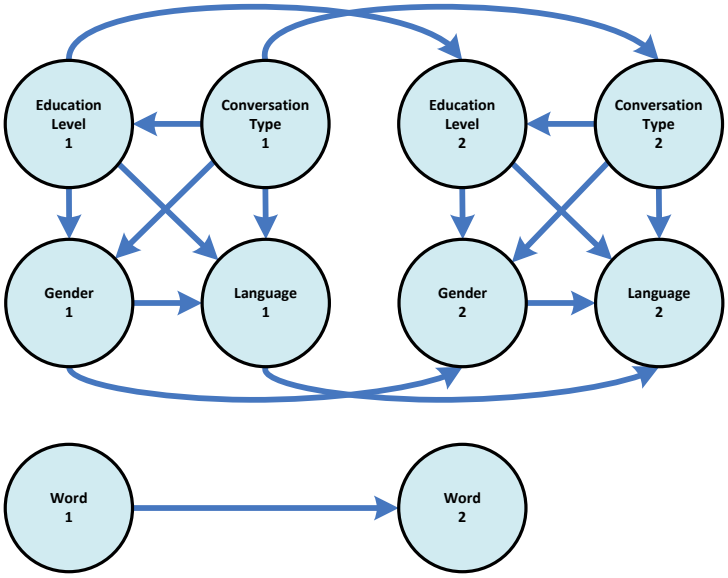
EXPERIMENT 3

FIRST TRIAL



SECOND TRIAL

In this trial, a new constraint is added. The model that is explored by the algorithm must incorporate N-gram model.



THIRD TRIAL

In this trial, a new constraint is added. The model that is explored by the algorithm must have at least one edge between non word variable and word variable.