# Bayesian Networks with Structural Restrictions: Performance and Parallelizations

by

Zhe Jin

(Under the Direction of John A.Miller)

## Abstract

Bayesian Network algorithms are widely applied in the fields of bioinformatics, document classification, data mining, and marketing informatics. In this paper, three Bayesian Network algorithms are evaluated: Naïve Bayes, Tree Augmented Naïve Bayes and Ordering-based Bayesian Networks. The algorithms are implemented using Scala, the bnlearn library in R and the WEKA software package. Several data sets with varying levels of attributes are used to test the accuracy of the algorithms and implementation testing is performed across all software platforms. We also parallelize these algorithms for efficiency gains when handling huge data sets. Significant speed-ups were achieved based on parallel processing.

Index words: Data mining; Parallel programming; Data analysis; Big data; Bayesian Networks

# Bayesian Networks with Structural Restrictions: Performance and Parallelizations

by

Zhe Jin

B.E., Nankai University, 2014

A Thesis Submitted to the Graduate Faculty
of The University of Georgia in Partial Fulfillment
of the
Requirements for the Degree

Master of Science

Athens, Georgia

2016

# Bayesian Networks with Structural Restrictions: Performance and Parallelizations

by

Zhe Jin

Approved:

Major Professor: John A.Miller

Committee: Lakshmish Ramaswamy
Khaled Rasheed

Electronic Version Approved:

Suzanne Barbour
Dean of the Graduate School
The University of Georgia
August 2016

# Acknowledgments

I would like to express my sincere appreciation and gratitude to my advisor Dr. John Miller, who has been a tremendous mentor for me. His guidance on both my academic research and future career has been invaluable to me. Dr. Miller played a key role in my entire two years of graduate study at UGA and made it such an enriching experience. I would also like to thank my committee members Dr. Ramaswamy and Dr. Rasheed. I benefited tremendously from their expertise, wisdom and advice. My gratitude also goes to Mr. Hao Peng and Mr. Zhaochong Liu, both of whom provided great advice, collaboration, and assistance on my research and programming.

Last but definitely not the least, Id like to thank my family, who has been my solid foundation throughout the years. Words cannot express my gratitude to my parents, my grandfather, my aunt and uncle, and my cousin. Without their continuous support and sacrifices, I would not have been able to achieve my academic goals. This is also dedicated in loving memory to my grandmother.

# Table of Contents

## List of Figures

# List of Tables

# Chapter 1

# Introduction

With the advancement of science and technology, there has been explosive growth of data over the last several years. An ever increasing volume of data is being captured, stored and accumulated on every aspect of society and human life [1]. With such an overwhelming availability of data, how to manage and analyze data, and more importantly, how to extract useful information and knowledge from it, has become an urgent issue.

In order to address this challenge, big data analytics techniques are being widely developed and adopted in many areas with the goal of extracting useful and insightful information from large amounts of data. Two important areas of big data analytics are prediction and classification. Several big data analytics frameworks, such as Spark and ScalaTion, provide support for prediction and classification on very large data-sets, including parallel and distributed processing, out-of-core computation and access to large data stores (e.g., Hadoop Distrubuted File System (HDFS), Spark's Resilient Distributed Dataset (RDD) and ScalaTion's Columnar Analytics Database) [2].

Among the various big data analytics techniques used for classification, Bayesian networks combine knowledge of graph theory and statistics, and provide a method for representing causal relationships among variables [3]. These techniques are used to find valuable knowledge hidden in data. In a Bayesian network, nodes represent random variables and edges represent conditional dependencies between variables. Using conditional probability, we can express dependencies between variables with easy-to-understand illustrations and clear

semantics. It also provides tremendous insight into further data analysis as well as accurate and effective forecasts. Due to their ability to be widely applied and potential for parallel execution, this paper focuses on Bayesian Networks.

One of the Bayesian network structure learning algorithms is the constraint based method. In 1988, Pearl discussed the Boundary DAG algorithm which uses Directed Acyclic Graphs (DAG) to represent a Bayesian Network [4]. Based on the DAG algorithm, Srinivas in 1990 proposed the SGS algorithm which can learn the structure of Bayesian network when the order of nodes in not given [5]. It uses the conditional test to determine the direction of edges between nodes in order to establish the structure. In 1993, Spirtes improved the SGS algorithm by ordering the conditional independence test from small to large [6]. It is more efficient and uses fewer calculations when building the sparse graph. Another algorithm, Local Causal Discovery (LCD) algorithm, was mentioned in 1997 by Cooper [7]. The advantage of this algorithm is that the search time is a polynomial time. Since it only does the conditional independence test on one variable at a time, the result of the algorithm is more reliable.

Another kind of Bayesian network learning algorithms are score-based methods, which has gained popularity in recent years. This method contains two types of algorithms, which are a scoring algorithm and a searching algorithm. A scoring function is used to evaluate how well the current candidate Bayesian network model describes the training data set (most popular functions include MDL [8], BDe [9], AIC [10], and BIC [11]). A search algorithm is used to find the Bayesian network with the highest score. Chickering has already proven that learning the Bayesian network structure is a NP-Hard problem [12], hence common search algorithms are inexact searching algorithms, such as heuristic algorithms, genetic algorithms, simulated annealing and MCMC.

In 1968, Chow and Liu proposed a method which can create a tree structured network, called Chow-Liu tree [13]. It uses conditional probabilities to build the tree and uses the Kullback–Leibler distance [14] to evaluate the network. It runs in polynomial time and is used to find the second order product approximation.

Cooper developed the K2 algorithm in 1992 [15], building a Bayesian network structure based on a given order of variables. Using a greedy search algorithm, it keeps adding arcs into the network structure and uses the MAP (maximum a posteriori) method to give scores to the network. This algorithm continues to add arcs until it cannot improve the MAP score any more.

The core of data mining tasks is trying to find useful information from huge data sets, so the challenge is the amount of time it takes for completion. Parallel computing techniques allow for these techniques to be used while minimizing the real time requirements for results. In this paper, we paralleled a Bayesian network structure learning algorithm, the tree augmented naive Bayes algorithm, the naive Bayes algorithm and the feature selection algorithms. We compare the time for running both sequential versions and parallel versions. We will also compare the accuracies to evaluate these algorithms.

Chapter 2 provides background on Bayesian network and structure learning. We will also briefly talk about the process of these algorithms. Chapter 3 introduces three popular Bayesian Network algorithms and two of them are used in the WEKA library. Chapter 4 focuses on the implementation process of the parallel algorithms. Chapter 5 shows the test result comparisons in terms of run time and accuracy. The conclusion and future work are mentioned in section 6.

# Chapter 2

# Background on Bayesian Networks

## 2.1 Bayesian Network

A Bayesian Network is a mathematical model based on probability reasoning [16]. Probability reasoning uses given information from some variables to calculate the probability of other variables value. Bayesian Networks use graphical network structure to show the joint probability distribution and conditional dependencies of variables. The basic theory used in the Bayesian Network is Bayes' theorem which describes the probability of an event with given situations.

$$P(A \mid B) = \frac{P(B \mid A)\, P(A)}{P(B)} \tag{2.1}$$

$P(A \mid B)$ is called conditional probability, it shows the probability of event A given event B.

$$P(A \mid B) = \frac{P(A \cap B)}{P(B)} \tag{2.2}$$

$P(A \cap B)$ is the joint probability which is the probability of event A and B occurring at the same time. $P(A)$ and $P(B)$ are the probability of A and B occurring without considering each other.

Bayesian Network structure is a directed acyclic graph (DAG), in which the nodes represent the random variables $\{X_1, X_2, X_3, \ldots, X_n\}$ and the arrow which connects the two nodes represents the conditional dependencies. So if two nodes are connected with each other, that means they are related, not conditionally independent. For example, if there is an arrow pointed from node $X_1$ to node $X_2$, node $X_1$ is called the parent and node $X_2$ is called the child. The arrow between node $X_1$ and $X_2$ represents the value of $P(X_2 \mid X_1)$. When the set of random variables is $G_v$, the set of conditional probabilities is $G_p$, the Bayesian Network G is always represented as $G = (G_v, G_p)$. Suppose $x \in G_v$, the joint probability can be represented as

$$p\{x_1, \ldots, x_n\} = p(x_n \mid x_1, \ldots, x_{n-1}) \ldots p(x_2 \mid x_1)\, p(x_1) \tag{2.3}$$

## 2.2 Naïve Bayes

Naïve Bayes Classifier is a simple classifier based on Bayes' theorem. The "Naïve" means that the algorithm assumes all the attributes are conditionally independent. Even though in most situations, the attributes are not conditionally independent, the algorithm can still get a good result. In Naïve Bayes Classifier, there are given attributes$\{A_1, A_2, \ldots, A_n\}$ and a class C which needs to be classified. Naïve Bayes model assumes that within a given class, the value of a feature is independent with any other features. In the training set, there are vectors of feature values and there is a class label for each vector. By training the model with the training set, the model can calculate the joint probability for each label.

$$p(A_i \mid C) = \frac{p(A_i)\, p(C \mid A_i)}{p(C)} \tag{2.4}$$

The classifier will predict the label which has the highest joint probability. The Naïve Bayes algorithm does not need to do structure learning and it is easy to build the model. The good performance on the accuracies also indicates that, relative to the relations between attributes and classes, the correlations between different features can be ignored. This result

is why Naïve Bayes has a good prediction results for many cases.

## 2.3 Tree Augmented Naïve Bayes

However, there are indeed situations where Naïve Bayes does not produce good results. As stated earlier, one of the most important assumptions in the Naïve Bayes algorithm is the conditional independence among attributes. When there is a strong correlation among attributes, Naïve Bayes fails to predict probabilities with a high level of accuracy. To improve the Naïve Bayes algorithm, instead of assuming attributes are completely independent from each other, we allow attributes to have some simple connections, which is called the Tree Augmented Naïve (TAN) Bayes algorithm [17].

In the Tree Augmented Naïve Bayes algorithm, besides the class, each attribute can have at most one other attribute as a parent. To find the parent for each attribute, the algorithm first builds a maximal weighted spanning tree based on the conditional mutual information between the features. The weight of the edge connecting attributes is annotated by the conditional mutual information. Then it chooses a root variable and sets the direction of all edges to be outward from it to transform the prior tree into a directed tree. As a last step, the algorithm adds a vertex labeled by class and adds an arc from class to each attribute to construct a TAN model.

## 2.4 Ordering-based Search Algorithm

In ScalaTion, we also implemented a Bayesian Network structure learning algorithm which is based on the paper "Ordering-based search: A simple and effective algorithm for learning Bayesian networks" published by Teyssier and Koller [18]. The method is based on the fact that for a given order of the variables in the network, finding the best scoring

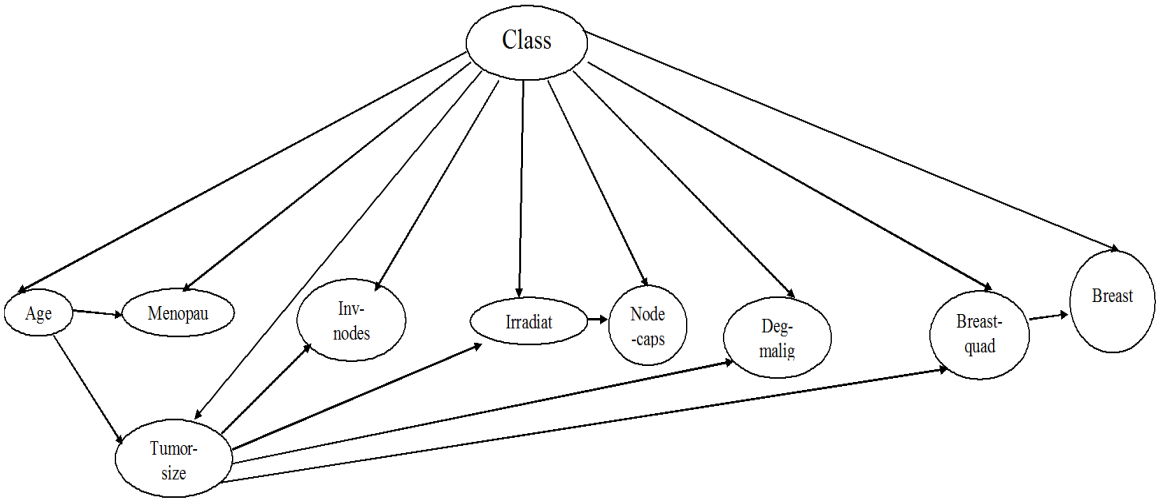


Figure 2.1: Bayesian Network Structure learned by TAN

network with the given order is not NP-hard [15] [19]. First of all, a scoring method of BIC/MDL or BDe is selected and then a number k is set as the number of parents per node. For each random starting order, every network is tested by swapping elements and comparing the resulting scores. For any order $k_1$ to $k_n$, we can first swap $k_1$ with $k_2$, get the score and swap back. Then we try to swap $k_2$ with $k_3$, save the score and swap back again. If the score of $\{k_2, k_1, k_3, \ldots, k_n\}$ is higher than $\{k_1, k_3, k_2, \ldots, k_n\}$, we keep the order and then keep swapping the left attributes. After testing all possible switches, the final score is the score for this starting order. This method allows for easy comparison of all starting orders to identify the best network.

## 2.5 Scoring Functions

Two popular and closely related scoring criteria for learning Bayesian Networks are Akaike Information Criterion (AIC)[10] and Bayesian Information Criterion (BIC)[11]. Both scoring

criteria need the maximum likelihood, which can be computed in the following ways as described by Bouckaert[21]. Three loops are needed to do this. Let the first loop be indexed by each feature $i$. Let the second loop be indexed by the $j–th$ parent configuration of feature $i$. A parent configuration is the Cartesian product of the sets of values that the parent features of a given feature $i$ can take. For example, if a feature has two parents, both of which are binary, then there are four possible parent configurations of this given feature. Let the third loop be indexed the $k–th$ value that feature $i$ can take. The maximum likelihood can then be calculated by iterating through the three loops as described above, and sum over $N_{ijk} * log(N_{ijk}/N_{ij})$. The aforementioned partial formula comes from page 2155 of Campos's "A Scoring Function for Learning Bayesian Networks based on Mutual Information and Conditional Independence Tests"[22] published in *Journal of Machine Learning Research* 7.Oct (2006). $N_{ijk}$ is the number of times the $i–th$ feature takes the $j–th$ parent configuration and the $k–th$ value in the training dataset. Similarly, $N_{ij}$ counts the number of times the $i–th$ feature takes the $j–th$ parent configuration, regardless of the actual value that feature $i$ takes. After maximum likelihood has been obtained, the second measure needed is the network complexity[22], which can be computed by iterating through all features, and summing over the product of the value count of each feature $i$ minus one and the number of parent configurations of the same feature. Then AIC can be simply computed by subtracting the network complexity from the maximum likelihood. BIC, on the other hand, can be computed by subtracting the product of the network complexity and $\frac{1}{2}log(N)$ from the maximum likelihood, where $N$ is the size of the training data. For both of these two scoring functions, a smaller or more negative value indicates a better network structure.

# Chapter 3

# Related Work

## 3.1 K2 Algorithm

One popular algorithm for learning a Bayesian Network is the K2 algorithm developed by Cooper and Herskovits [15]. The K2 algorithm is a heuristic search algorithm that requires an ordering of the nodes and the maximum number of parents any given node may have. The ordering of the nodes is used to enforce acyclicity in the network, as a node may only choose parents from nodes appearing before it in the ordering. The K2 algorithm would loop through all the nodes, each corresponding to a feature in the training data. For each node, the algorithm would loop through all nodes that are valid parent candidates to the given node. For each of the parent candidate, the algorithm first attempts to temporarily establish a parent-child relation in the network structure. The candidate that produces the greatest improvements in terms of a score measure of some scoring function would then be permanently added as a parent to the given node. The process would keep attempting to add one parent at a time until no more parent can be added, either due to the lack of candidates, or the maximum number of parent parameter has been reached.

## 3.2 Max-min hill-climbing Bayesian network structure learning algorithm

In the paper "The max-min hill-climbing Bayesian network structure learning algorithm", Tsamardinos and Brown describe an algorithm called Max-Min Hill-Climbing (MMHC) which can be used with thousands of variables to get a good quality result from learning

the Bayesian Network structure [23]. The MMHC algorithm first uses the Max-Min Parents and Children algorithm (MMPC) to learn the structure of the Bayesian network. After this, it uses a greedy Bayesian-scoring hill-climbing search to orient the structure.

MMPC is the method to calculate false positives related to the structure [24]. For Z estimated by a statistical test on the training data, when X and T are conditionally independent, the MMPC algorithm returns true for testing the independence of X and T given Z. For each variable T, the parent and children set of T is called PCT and the candidate set of PCT is called CPC. As a starting point, for each iteration in the forward phase the set CPC is updated by the function Max-Min Heuristic. The method of Max-Min Heuristic selects the variable which can maximize the minimum association with T relative to CPC. The loop stops when given the subset of CPC, all the other variables are independent with the target T. For the next step, MMPC tries to eliminate the false positives which may have been added in the first step. It executes the independence test for some subset $S \subseteq PC$. If the test is positive, X will be deleted from CPC.

The MMHC algorithm starts from an empty graph. It uses the BDeu [15] scoring method to evaluate the operators such as add-edge, delete-edge and reverse-edge and choose the operator which has the largest increase in score. For the edge addition operator, it can only add an edge which is found by the MMPC algorithm. It uses a TABU list to save the last 100 Bayesian network structures found. To avoid the local maxima, the best local change is not on the TABU list. If after 15 changes there is no score increase, the algorithm will stop and return the best Bayesian network structure found.

The advantage of the MMHC algorithm is that it is based on three different ideas: search and score, local learning, and constraint-based techniques [23]. It uses the greedy search

algorithm to find the optimal solution in an efficient way and uses the simulated annealing idea to avoid getting the local maxima.

# Chapter 4

## Development of Parallel Algorithms

Scala provides a parallel collection method to perform parallel computing. When using the par method, the parallel collection will use all available cores on the system to conduct the computation on the collection. Similar to the common parallel method, Scala divides the job into several parts and each core works on one part of the collection. The main method in parallel collection is the Fork/Join Framework [25, 26], which is a divide-and-conquer algorithm. Fork divides a big job into several smaller sub-jobs and runs them in parallel. Join collects the results and combines them to get the final result for the original job. For example, if the original task is to add the integers from 1 to 10,000, Fork may divide the job into ten tasks and each task involves adding a thousand numbers. Join will then collect the results and combine them together (in this case, adding them all together) to get the answer to the summation of 1 through 10,000.

Work-Stealing algorithm is another important method in Fork/Join Framework [27]. When Fork divides the job into several small tasks, these tasks are assigned to different threads. When one thread finishes the task assigned to it, it can steal tasks from other threads. To avoid potential conflicts in threads, all tasks are stored in a queue within each thread. Each thread works on the tasks based on priority from the top to the bottom of the queue, and it can only steal tasks from the bottom of other threads queue.

While running Naïve Bayes algorithm in parallel, we noticed that the frequency count of each value in each feature and the classes always takes a long time to complete. Initially, we

tried applying the par collection method directly on the counting loop, but the result proved to be incorrect. We found that the result array is a global variable, hence when different threads attempted to add values to the array, a conflict was created. This indicates different threads may change the same variable at the same time. To correct this, we divided the big data sets into several small subsets. Then we created several workers to count the frequencies only for each subset. In each worker, we created arrays to store the counting results.

```
val wrange = (0 until endworkers).par
wrange.tasksupport = new ForkJoinTaskSupport (new ForkJoinPool (endworkers))
for (w <- wrange) {
  for (l <- w * (size) until min((w+1)*size, m) if l < testStart || l >= testEnd) {
    val i = y(l)
    popCw(w)(i) += 1
    for (j <- 0 until n) popXw(w)(i, j, x(l, j)) += 1
  } // for
} // for
```

The number of endworkers equals to the number of available threads. So we use the par function on the wrange to parallely calculate the frequencies for each subset. The tasksupport is used to control the number of available threads when we do the scalability testing. After counting is completed for each worker, we processed all the arrays to collect results and combine them together. This method greatly improved the speed to get frequency counts compared to the previous serial version.

```
for (w <- 0 until endworkers) {
  popC += popCw(w)
  popX += popXw(w)
} // for
```

In Selective Naïve Bayes [28], selection of the feature set is the most important step in the algorithm. In the serial version, we start with an empty feature set. A feature is

then added to the feature set, a cross-validation test on the data set is conducted to get an accuracy and then the feature is moved out of the feature set. Then a second feature is added into the feature set and the same procedure is repeated. Each time an accuracy is obtained and compared to the previous one. If it is higher, the second accuracy will be saved as the highest accuracy and the index of the feature will be saved. If it is not, testing continues for the next feature. After the loop is completed on the entire feature set, the feature with the highest accuracy will be identified and saved. Testing on the features is complete when accuracies can no longer be improved.

To reduce run time, we developed an algorithm to run feature selection in parallel. Instead of testing on features one by one, we conducted the cross-validation tests on each feature at the same time by using the par function. Accuracies of each feature were saved in an array and after testing on all the features, we processed the array to find the highest accuracy and the index. Then we put the feature into the global feature set and tested on the rest of the features. We continued to find the best accuracy of different feature combinations and kept putting the next best feature into the global feature set. In the serial algorithm, the cross-validation function used the train and test functions to test the accuracy. In the train function, frequencies of the features are calculated and saved as global variables. If we conducted feature selection in parallel, the train function in different threads would have changed the frequencies at the same time. To keep this from happening, we saved the frequencies as local variables inside the feature selection function and passed them into the train function. In this way, the training function in one thread can only change the frequencies in its own thread, enabling us to execute the feature selection method accurately.

In the Ordering Based Bayesian Network algorithm, the structure learning procedure is very time consuming. It is largely dependent upon the number of times the loop needs to run to find the new random feature order. If we set the parameter to 1,000, it will run the

loop a thousand times to find the best feature order. In each loop, the procedure randomly generates a feature order, evaluates it using the AIC scoring algorithm and compares the score with the previous score. In the sequential version of the loop algorithm, it calls the training function to calculate frequencies and save them as global variables. The scoring function keeps calling these variables to evaluate the feature order, comparing the score with the max score and updates the max score when needed. After each loop completion, it will reset all global variables to zero and then begin the next loop. If these loops could run in parallel, it would save a lot of running time and improve efficiency. In the parallel version however, modifications are made to save the frequencies as local variables within each loop. The scoring function is also changed to read local variables as input and to output a score. Feature orders and their scores are saved as key value pairs. After we finish running all the loops, the max score and the feature order corresponding to it can be identified.

# Chapter 5

## Performance Evaluations

Our experiments utilized data sets downloaded from the UCI Machine Learning Repository [29] and KEEL dataset repository [30]. To simplify the data cleansing process, we chose data sets whose attributes are categorical. We also selected data sets that have large numbers of instances, such that the time efficiency differences between serial and parallel algorithms tend to be more significant and notable. Table 5.1 shows the data sets selected for experiment. In the adult data set, two attributes are continuous variables which are not supported in our algorithms right now. So we skip these two columns when we do the experiments.

For the accuracy evaluation, we compared our algorithms using ScalaTion, WEKA and R. We used Naïve Bayes, Selective Naïve Bayes, Tree Augmented Naïve Bayes (TAN), Augmented Naïve Bayes and Ordering-based Bayesian Network to conduct a 5 fold cross-validation test on all these data sets. For the cross-validation test, random cross-validation was adopted where the training and test data sets were randomly picked each time. We then ran cross-validation ten times on each data set and calculated the average accuracy.

WEKA is a knowledge analysis environment which is implemented in Java. It has many machine learning algorithms and is widely used in the machine learning area [31]. In WEKA, we picked Naïve Bayes, Tree Augmented Naïve Bayes (TAN), K2 Bayesian Network Learning algorithm and the Hill Climbing algorithm. We also utilized the cross-validation package in WEKA to perform the 5 fold cross-validation test using the chosen algorithms on these data

sets.

In R, we selected the bnlearn package, which is a Bayesian Network Structure learning package [32]. We tested the accuracies using Naïve Bayes and Tree Augmented Naïve Bayes (TAN). In the bnlearn package, 5-fold cross-validation function is also available for algorithm evaluation.

To test the time efficiency on parallel algorithms, we needed data sets significantly larger, with hundreds of thousands or even millions of instances. A data set expansion methodology was developed. Initially, we randomly pick a record from the original data set and randomly generate a new row of data with the exact same number of attributes. Then we add these two rows together to create a new record with values that are different from the original row and add it to the data set. This process is repeated until we reach the desired size of the data set. Following this methodology, we expanded both Nursery and Letter data set to 200,000 and 5,000,000 instances. The data sets with 5,000,000 instances were run using parallel processing on Naïve Bayes and TAN algorithms, while the data sets with 200,000 instances were used to test parallel Selective Naïve Bayes and Ordering-based Bayesian Network algorithm. To test the effect of additional threads, we ran the experiment by incrementing the number of threads by one on each successive run.

We conducted these tests on a 12-core Intel Xeon machine in the Georgia Advanced Computing Resource Center Zcluster which has 48GB memories, running a Red Hat Enterprise Linux Server release 5.11 operating system.

Table 5.1: Data Sets Information

| Data Sets | Attributes | Classes | Instances |
|---|---|---|---|
| Adult | 14 | 2 | 32459 |
| Breast Cancer | 8 | 2 | 286 |
| Connect-4 | 42 | 3 | 67557 |
| Flare | 10 | 2 | 710 |
| German | 24 | 2 | 1000 |
| Kr-vs-k | 6 | 18 | 28056 |
| Letter | 16 | 26 | 20000 |
| Nursery | 8 | 5 | 12960 |

Table 5.2: Naïve Bayes Accuracies

| Data Sets | NB | NB in R | NB in WEKA | Selective NB |
|---|---|---|---|---|
| Breast Cancer | 72.95% | 72.17% | 72.73% | 75.02% |
| Adult | 80.86% | 81.86% | 81.059% | 83.14% |
| Letter | 74.47% | 74.80% | 74.29% | 73.72% |
| German | 72.51% | 72.71% | 72.10% | 71.02% |
| Flare | 80.03% | 80.58% | 80.28% | 82.3% |
| Connect-4 | 72.12% | 72.13% | 72.12% | 66.20% |
| Nursery | 90.31% | 90.31% | 88.34% | 87.88% |
| Kr-vs-k | 36.06% | 36.17% | 35.58% | 34.44% |
| **Average** | **72.413%** | **72.59%** | **72.062%** | **71.73%** |

## 5.1 Comparison of Sequential Algorithms

Table 5.2 shows the accuracy comparison from our ScalaTion project, WEKA data mining software and the bnlearn library in R. For the Naïve Bayes algorithm, ScalaTion, WEKA and R yielded similar accuracy rates. Scalation and WEKA both used random cross-validation while R used 5-fold cross-validation, and the differences among the accuracy rates are within a small range. In WEKA we set the option 'use supervised discretization' to true, others are default settings. In R, we set the 'fit' as 'bayes', others are default.

Table 5.3: Tree Augmented Naïve Bayes Accuracies

| Data Sets | TAN | TAN in R | TAN in WEKA | Selective TAN |
|---|---|---|---|---|
| Breast Cancer | 63.47% | 66.57% | 69.23% | 71.23% |
| Adult | 80.06% | 81.90% | 86.01% | 82.12% |
| Letter | 84.83% | 87.92% | 85.67% | 83.69% |
| German | 61.62% | 64.17% | 71.90% | 72.80% |
| Flare | 73.54% | 80.87% | 81.27% | 82.96% |
| Connect-4 | 71.90% | 76.44% | 76.42% | 68.51% |
| Nursery | 87.68% | 93.40% | 93.97% | 82.44% |
| Kr-vs-k | 50.52% | 54.85% | 51.43% | 49.56% |
| **Average** | **71.70%** | **75.77%** | **76.99%** | **74.16%** |

As we can see, the TAN algorithm gets a much higher accuracy on some of the data sets, which is indication of strong interdependence among attributes for those data sets. By definition, Naïve Bayes ignores any correlation between attributes, hence when attributes are correlated, Naïve Bayes shows a relatively lower accuracy. By the same token, since the TAN algorithm assumes correlations between attributes, it will result in lower accuracy than the Naïve Bayes algorithm, even when the interdependences are weak.

As shown in Table 5.3, the accuracies for the TAN algorithm in Scalation is lower than R and WEKA. Our TAN uses the conditional mutual information to evaluate the correlations between the attributes, but WEKA uses scoring functions like Loglikelihood [21], AIC, BIC and MDL to calculate the correlations and R uses the discrete mutual information. Both WEKA and R use the Kruskal algorithm to build the maximum spanning tree, while our TAN algorithm uses the Prim algorithm, resulting in different tree structures. Hence the performance of our TAN algorithm is not quite as accurate as that of R and WEKA.

Table 5.2 also demonstrates that the selective algorithms improve accuracies on some of the data sets. By design, the selective algorithms only choose the attributes that are more

Table 5.4: Bayesian Network Accuracies

| Data Sets | Order Based | WEKA K2 | WEKA Hillclimber |
|---|---|---|---|
| Breast Cancer | 72.53% | 72.03% | 74.48% |
| Adult | 81.08% | 83.88% | 83.88% |
| Letter | 77.14% | 74.20% | 74.20% |
| German | 67.38% | 71.80% | 71.80% |
| Flare | 79.85% | 79.44% | 79.44% |
| Connect-4 | 71.75% | 72.14% | 72.11% |
| Nursery | 90.25% | 90.10% | 90.10% |
| Kr-vs-k | 36.25% | 35.64% | 35.64% |
| **Average** | **72.03%** | **72.40%** | **72.70%** |

relevant to the class [28]. If an attribute is not related to the class, it will be treated as less important when building the model and the outcome will impact the accuracy rate. Hence it is quite obvious that eliminating less important attributes may or may not increase the accuracies.

Compared to average accuracy from Naïve Bayes and TAN in ScalaTion, the Bayesian Network algorithm has a similar average accuracy. The Bayesian Network algorithm in ScalaTion has a similar accuracy level with the K2 and Hill Climbing algorithm in WEKA, and it is equivalent to the accuracy levels of the Naïve Bayes algorithm.

To prove our hypothesis of attribute independence on some data sets and further validate the various algorithm performance, we calculated the correlation coefficients between attributes on two data sets. A high correlation coefficient is indication of strong positive or negative relationship between two attributes. The closer to 1 or -1 a coefficient, the stronger the positive or negative correlation between these two attributes.

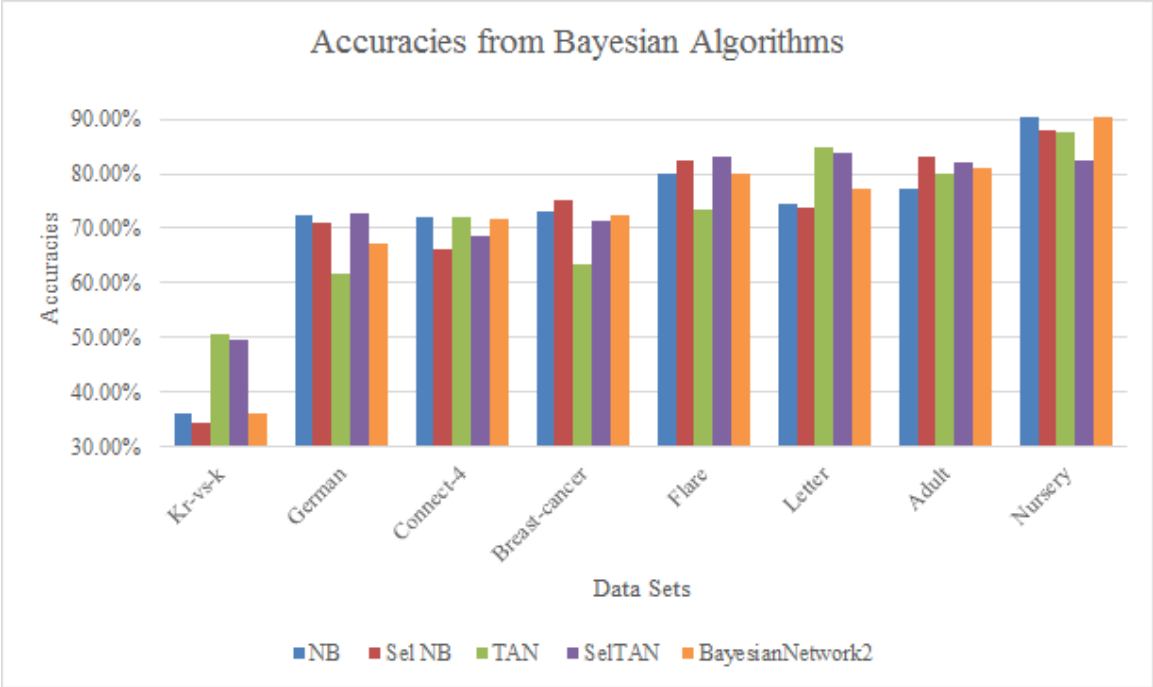


Figure 5.1: Accuracies from Bayesian Algorithms

Table 5.5: Breast Cancer Attributes Correlations

| Attributes | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 1 | | | | | | | |
| 2 | 0.241 | | | | | | |
| 3 | -0.045 | 0.019 | | | | | |
| 4 | -0.001 | -0.011 | -0.131 | | | | |
| 5 | 0.052 | 0.130 | 0.058 | -0.465 | | | |
| 6 | -0.043 | -0.161 | 0.132 | -0.213 | 0.098 | | |
| 7 | 0.067 | 0.077 | -0.022 | 0.040 | 0.024 | -0.073 | |
| 8 | -0.024 | -0.096 | -0.056 | 0.062 | -0.036 | 0.018 | 0.175 |

Table 5.5 demonstrates the coefficients calculated from the Breast Cancer data set. The highest coefficient is between attribute 4 and attribute 5, and it is at -0.465, which is a moderate correlation. Vast majority of the coefficients are around 0, which is indication of an absence of strong relationships between attributes. Based on this, we expect Naïve

| Attributes | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | | | | | | | | | | | | | | | |
| 2 | 0.758 | | | | | | | | | | | | | | |
| 3 | 0.852 | 0.672 | | | | | | | | | | | | | |
| 4 | 0.673 | 0.823 | 0.660 | | | | | | | | | | | | |
| 5 | 0.619 | 0.555 | 0.766 | 0.644 | | | | | | | | | | | |
| 6 | -0.033 | 0.046 | 0.062 | 0.043 | 0.139 | | | | | | | | | | |
| 7 | 0.046 | -0.041 | 0.025 | -0.020 | -0.029 | -0.357 | | | | | | | | | |
| 8 | 0.014 | -0.025 | -0.099 | 0.082 | -0.012 | -0.053 | -0.119 | | | | | | | | |
| 9 | 0.052 | 0.096 | 0.057 | 0.059 | -0.066 | -0.123 | -0.050 | -0.188 | | | | | | | |
| 10 | 0.148 | 0.160 | 0.115 | 0.012 | -0.070 | 0.086 | 0.178 | -0.318 | 0.132 | | | | | | |
| 11 | 0.035 | -0.055 | 0.012 | -0.012 | -0.073 | -0.342 | 0.600 | 0.043 | -0.060 | 0.058 | | | | | |
| 12 | -0.046 | -0.008 | -0.045 | 0.026 | -0.039 | -0.032 | -0.272 | 0.082 | 0.119 | -0.107 | 0.063 | | | | |
| 13 | 0.489 | 0.274 | 0.557 | 0.265 | 0.628 | 0.144 | -0.037 | 0.142 | -0.384 | -0.176 | 0.054 | -0.009 | | | |
| 14 | 0.098 | -0.001 | 0.046 | 0.025 | 0.018 | -0.253 | 0.555 | -0.085 | -0.053 | 0.029 | 0.527 | -0.185 | 0.003 | | |
| 15 | 0.274 | 0.231 | 0.260 | 0.298 | 0.493 | 0.127 | -0.078 | 0.007 | 0.278 | -0.087 | -0.226 | 0.050 | 0.108 | -0.064 | |
| 16 | -0.105 | -0.043 | -0.118 | -0.019 | -0.063 | 0.249 | -0.208 | 0.183 | -0.061 | -0.114 | -0.237 | 0.246 | -0.050 | -0.188 | 0.144 |

Figure 5.2: Letter Attributes Correlations

Bayes algorithm to generate better results than the TAN algorithm on accuracies. Results in table 5.2 and 5.3 provide strong evidence. Across all implementation methods, Naïve Bayes unequivocally outperforms TAN for the Breast Cancer data set. Our Scala projection yields 72.95% accuracy for Naïve Bayes but only 63.47% for TAN. This also indicates that Naïve Bayes algorithm is performing better on this data set because all the attributes are conditionally independent. But TAN algorithm forces to find a parent for every feature so it has a lower accuracy.

Figure 5.2 shows the correlation coefficients between attributes in the Letter data set. Contrary to what we observe for the Breast Cancer data set, there are several attribute pairs with fairly strong to strong correlations. Attributes 1 and 2 and attributes 2 and 4 have correlation coefficients of 0.852 and 0.823 respectively, both are considered strong. All the remaining pairs within the first 5 attributes score coefficients greater than 0.55, which is indication of positive relationship. Again, with the presence of attribute correlation, we expect TAN and Bayesian Network algorithms to produce better more accurate results. As

shown in Table 5.3 and 5.4, TAN generates the best accuracies among the three, Bayesian Network being second best. The correlation study on data sets provides insights as well as validations to the accuracies produced by Naïve Bayes, TAN and Bayesian Network algorithms. It confirms the hypothesis that when attribute correlation is present, Naïve Bayes algorithm fails to predict probabilities with a high level of accuracy.

## 5.2 Comparison of Parallel Algorithms

As shown in Figure 5.3 and 5.4, the run time of the parallel version of Naïve Bayes algorithm has a clear inverse relationship with the number of threads. As the number of threads increases, the run time decreases. Since training the Naïve Bayes model is simpler than other algorithms, the parallel function performs well and the speed-up number is a linear function of the number of threads. The Letter data set has twice as many attributes as the nursery data set, hence the run time approximately doubles that of the nursery data set.

For Selective Naïve Bayes, due to the feature selection process, the more features a data set has, the longer it will take to build the model. As shown in Figure 5.5 and 5.6, it takes almost 4 times as long to run the Letter data set than the Nursery data set on a single thread. But when feature selection has already selected a few features and the number of candidate features is less than the number of threads, in our implementation some threads will become unnecessary. In general, the more features the data set has, the better performance we will get when running on multiple threads. This explains why processing the Nursery data set only speeds up by about 6 when running on 8 threads and processing the Letter data set speeds up by 8.

Figure 5.7 and 5.8 shows that the parallel TAN algorithm speeds up by about 4.5 when running on 12 threads. It is because in the structure learning function, besides calculating

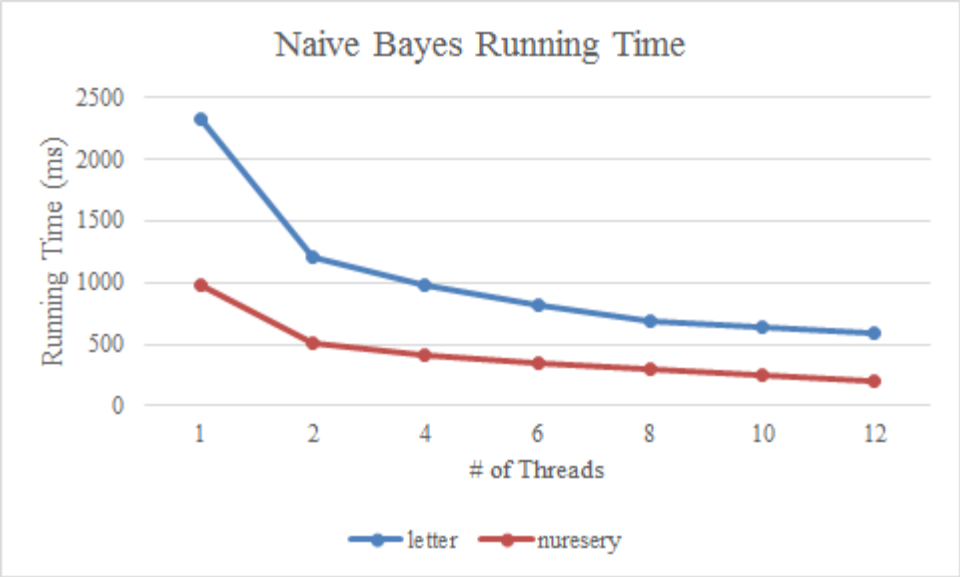


Figure 5.3: Naïve Bayes Running Time

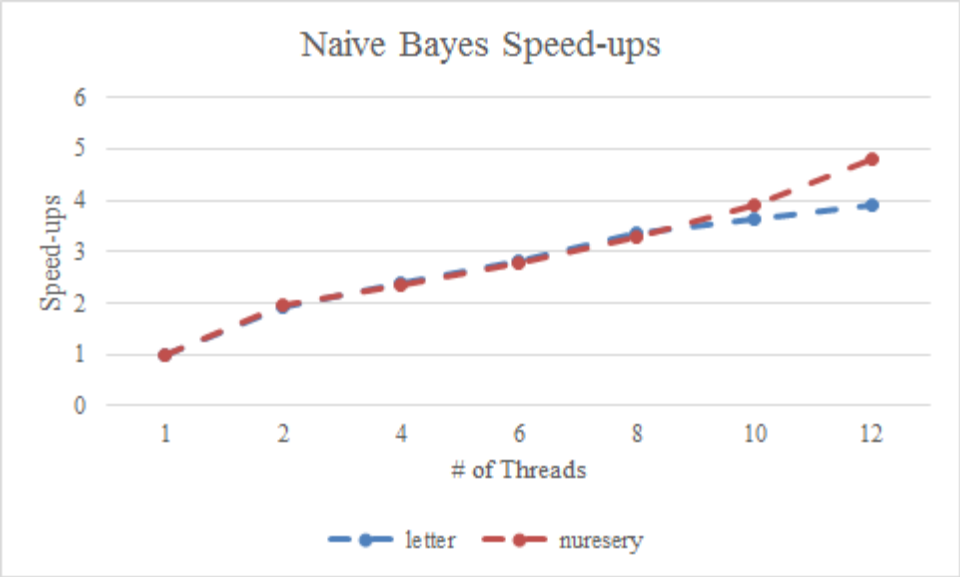


Figure 5.4: Naïve Bayes Speed-ups

the frequencies, it also builds a tree based on the calculated conditional mutual information. We only applied the parallel techniques on calculating the frequencies so the speed-ups are

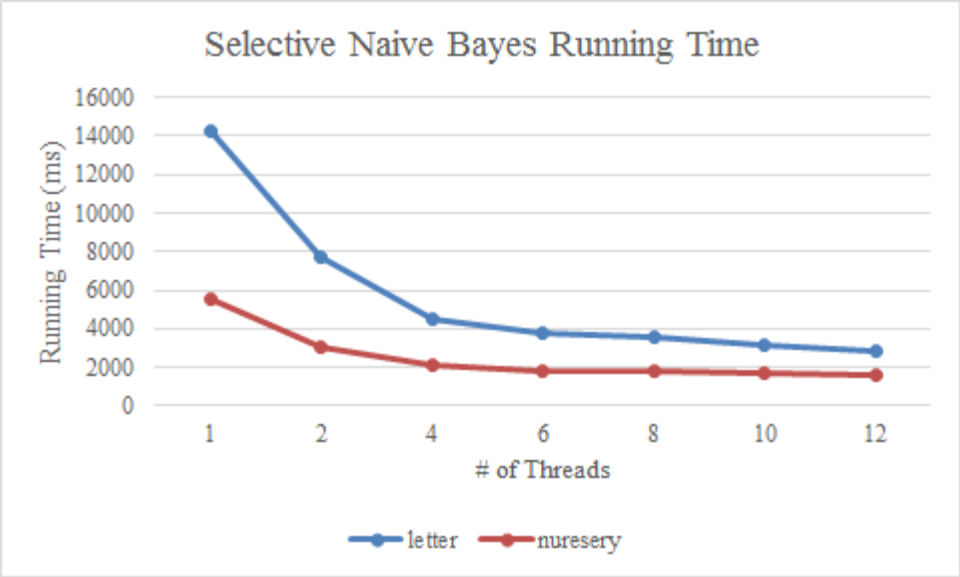


Figure 5.5: Selective Naïve Bayes Running Time

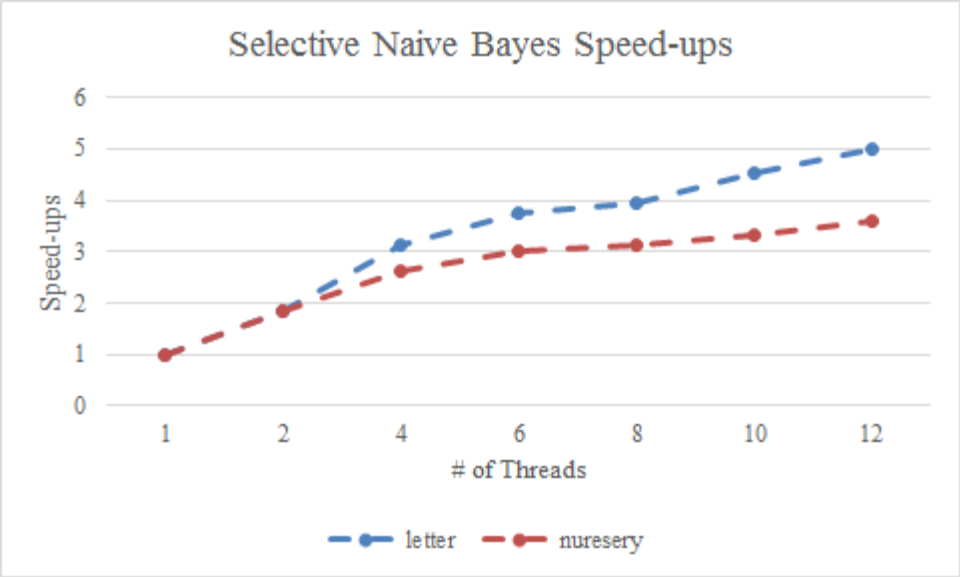


Figure 5.6: Selective Naïve Bayes Speed-ups

not as good as other algorithms.

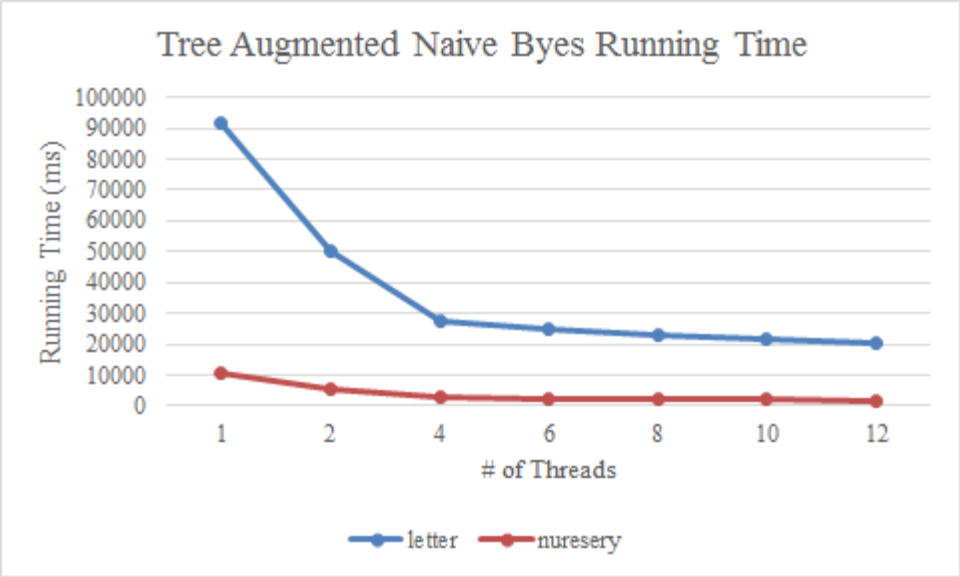


Figure 5.7: Tree Augmented Naïve Bayes Running Time

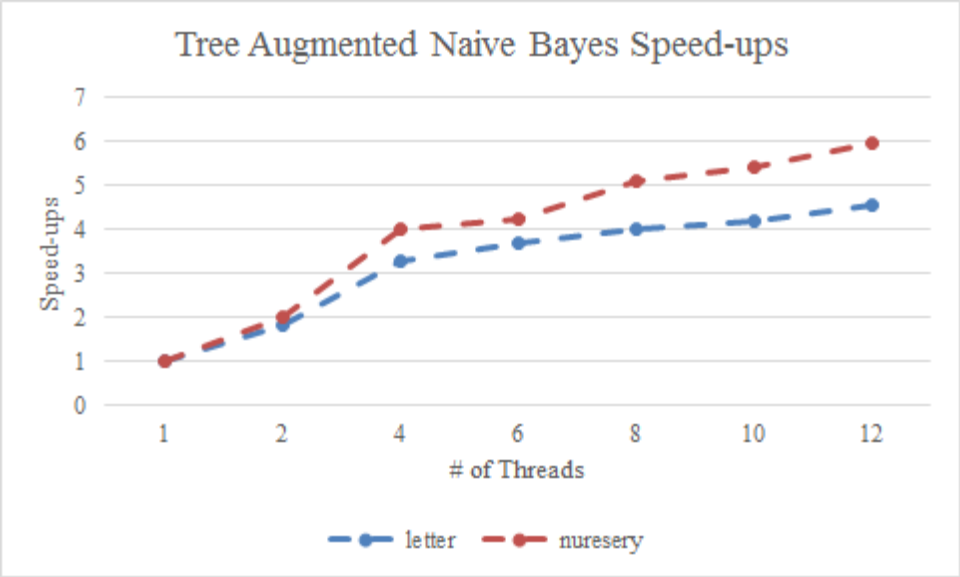


Figure 5.8: Tree Augmented Naïve Bayes Speed-ups

In the Bayesian Network algorithm, the goal is to find the best network structure, hence it will swap features and then compare scores. As a result, data sets with more attributes

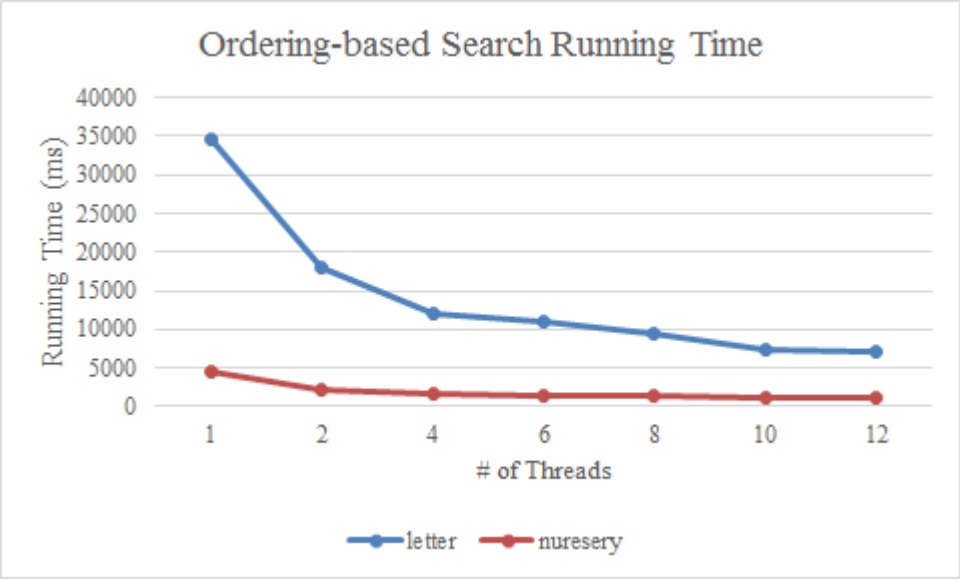


Figure 5.9: Ordering-based Bayesian Network with $k = 2$ Running Time

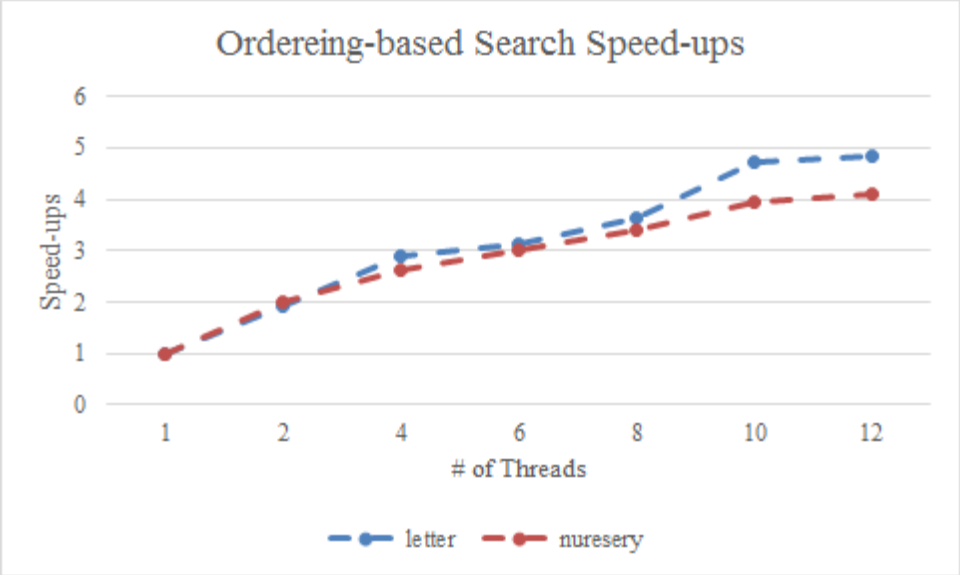


Figure 5.10: Ordering-based Bayesian Network with $k = 2$ Speed-ups

Table 5.6: Bayesian Algorithms Running Times (ms)

| Algorithms | Nursery | Letter |
|---|---|---|
| NB | 979.7 | 2323.8 |
| NB in WEKA | 2124.7 | 2045.2 |
| NB in R | 3749.0 | 7050.0 |
| NB Parallel | 204.0 | 593.9 |
| TAN | 10732.5 | 91651.3 |
| TAN in WEKA | 7939.8 | 12814.8 |
| TAN in R | 4012.0 | 10049.0 |
| TAN Parallel | 1797.9 | 20176.2 |

have taken significantly longer run times. Since the Letter data set has more attributes, it takes 6 to 8 times longer to run the Letter data set than the Nursery data set. However, the speed-ups for the data set with more attributes is almost the same as that of the data set with fewer attributes.

Table 5.6 shows the comparison of running times for Naïve Bayes and TAN algorithms. In sequential mode, the Naïve Bayes running time from our ScalaTion project clearly shows advantage over WEKA and R, cutting the run time by 21% and 29% for Nursery and Letter respectively. For TAN, R shows the best efficiency over our ScalaTion project and WEKA. We executed the parallel algorithms with twelve available threads. The run time for Naïve Bayes is drastically reduced by 89.8% and 90.0% respectively. For TAN, our parallel algorithm cut the run time by about 83.3% for Nursery and 78.0% for Letter, relative to our sequential version. The use conditional mutual information requires us to work in a higher dimension in the frequency method, so the running time is slower than the other algorithms.

# Chapter 6

## Conclusions and Future Work

In this study, we evaluated several Bayesian network algorithms including Naïve Bayes, Tree Augmented Naïve Bayes, and Ordering-Based Bayesian Network. We developed these algorithms using ScalaTion, WEKA and R and compared the accuracies across these packages. Furthermore, we applied parallel techniques to execute these algorithms with the goal of shortening run time and improving efficiency.

The experiment results demonstrate that our Bayesian algorithms developed in ScalaTion produce outcomes that are equivalent to WEKA and R. Our results also show that TAN and Ordering-Based Bayesian Network provides more accurate predictions than Naïve Bayes when strong attribute correlation is present.

The run time comparisons shows that our Naïve Bayes algorithms run faster than R and WEKA when running in sequence. Our parallel version further reduces run time by up to a factor of 5. Although our TAN algorithm is not as fast as WEKA and R in the sequential version when running on the letter data set, it outperforms R and WEKA in nursery data set when running in parallel. When we run our parallelized algorithms on the machine in the Zcluster, the load is always about 8 to 10. So that means the threads on the machine are busy. It will slow down the algorithms and this is why we can only get a speed-up at most 5 when running on multiple threads.

Our future research will include fine-tuning the TAN algorithm to reach the accuracy levels achieved by WEKA. Alternative scoring functions such as AIC or BIC could be used to compare the structures with WEKA. Our TAN run time is also longer than expected and additional work is needed to simplify the TAN structure learning functions. We also need to find a better machine which has a lower load to test our parallelized algorithms so we can get a better speed-up. In their current state, these algorithms only work on categorical data. Subsequent work will focus on enhancements to enable inclusion of continuous data.

# Bibliography


[1] K. Luu, C. Zhu, and M. Savvides, "Distributed class dependent feature analysisa big data approach," in *Big Data (Big Data), 2014 IEEE International Conference on*. IEEE, 2014, pp. 201–206.

[2] X. Lin, P. Wang, and B. Wu, "Log analysis in cloud computing environment with hadoop and spark," in *Broadband Network & Multimedia Technology (IC-BNMT), 2013 5th IEEE International Conference on*. IEEE, 2013, pp. 273–276.

[3] Y. Tang, Y. Wang, K. M. Cooper, and L. Li, "Towards big data bayesian network learning-an ensemble learning based approach," in *2014 IEEE International Congress on Big Data*. IEEE, 2014, pp. 355–357.

[4] J. Pearl, *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann, 2014.

[5] P. Spirtes, C. N. Glymour, and R. Scheines, "Causality from probability," 1989.

[6] P. Spirtes, C. N. Glymour, and Scheines, *Causation, prediction, and search*. MIT press, 2000.

[7] G. F. Cooper, "A simple constraint-based algorithm for efficiently mining observational databases for causal relationships," *Data Mining and Knowledge Discovery*, vol. 1, no. 2, pp. 203–224, 1997.

[8] J. Rissanen, "Modeling by shortest data description," *Automatica*, vol. 14, no. 5, pp. 465–471, 1978.

[9] D. Heckerman, D. Geiger, and D. M. Chickering, "Learning bayesian networks: The combination of knowledge and statistical data," *Machine learning*, vol. 20, no. 3, pp. 197–243, 1995.

[10] H. Akaike, "A new look at the statistical model identification," *IEEE transactions on automatic control*, vol. 19, no. 6, pp. 716–723, 1974.

[11] G. Schwarz *et al.*, "Estimating the dimension of a model," *The annals of statistics*, vol. 6, no. 2, pp. 461–464, 1978.

[12] D. M. Chickering, D. Heckerman, and C. Meek, "Large-sample learning of bayesian networks is np-hard," *The Journal of Machine Learning Research*, vol. 5, pp. 1287–1330, 2004.

[13] C. Chow and C. Liu, "Approximating discrete probability distributions with dependence trees," *IEEE transactions on Information Theory*, vol. 14, no. 3, pp. 462–467, 1968.

[14] S. Kullback and R. A. Leibler, "On information and sufficiency," *The annals of mathematical statistics*, vol. 22, no. 1, pp. 79–86, 1951.

[15] G. F. Cooper and E. Herskovits, "A bayesian method for the induction of probabilistic networks from data," *Machine learning*, vol. 9, no. 4, pp. 309–347, 1992.

[16] F. Ruggeri, R. S. Kenett, and F. W. Faltin, "Encyclopedia of statistics in quality and reliability," 2007.

[17] N. Friedman, D. Geiger, and M. Goldszmidt, "Bayesian network classifiers," *Machine learning*, vol. 29, no. 2-3, pp. 131–163, 1997.

[18] M. Teyssier and D. Koller, "Ordering-based search: A simple and effective algorithm for learning bayesian networks," *arXiv preprint arXiv:1207.1429*, 2012.

[19] W. Buntine, "Theory refinement on bayesian networks," in *Proceedings of the Seventh conference on Uncertainty in Artificial Intelligence*. Morgan Kaufmann Publishers Inc., 1991, pp. 52–60.

[20] T. Silander and P. Myllymaki, "A simple approach for finding the globally optimal bayesian network structure," *arXiv preprint arXiv:1206.6875*, 2012.

[21] R. R. Bouckaert, "Bayesian belief networks: from construction to inference," 2001.

[22] L. M. d. Campos, "A scoring function for learning bayesian networks based on mutual information and conditional independence tests," *Journal of Machine Learning Research*, vol. 7, no. Oct, pp. 2149–2187, 2006.

[23] I. Tsamardinos, L. E. Brown, and C. F. Aliferis, "The max-min hill-climbing bayesian network structure learning algorithm," *Machine learning*, vol. 65, no. 1, pp. 31–78, 2006.

[24] I. Tsamardinos, C. F. Aliferis, A. Statnikov, and L. E. Brown, "Scaling-up bayesian network learning to thousands of variables using local learning techniques," *Vanderbilt University DSL TR-03-02*, 2003.

[25] D. Lea, "A java fork/join framework," in *Proceedings of the ACM 2000 conference on Java Grande*. ACM, 2000, pp. 36–43.

[26] A. Prokopec, P. Bagwell, T. Rompf, and M. Odersky, "A generic parallel collection framework," in *European Conference on Parallel Processing*. Springer, 2011, pp. 136–147.

[27] J. Dinan, D. B. Larkins, P. Sadayappan, S. Krishnamoorthy, and J. Nieplocha, "Scalable work stealing," in *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*. ACM, 2009, p. 53.

[28] P. Langley and S. Sage, "Induction of selective bayesian classifiers," in *Proceedings of the Tenth international conference on Uncertainty in artificial intelligence*. Morgan Kaufmann Publishers Inc., 1994, pp. 399–406.

[29] M. Lichman, "UCI machine learning repository," 2013. [Online]. Available: http://archive.ics.uci.edu/ml

[30] J. Alcalá-Fdez, L. Sanchez, S. Garcia, M. J. del Jesus, S. Ventura, J. M. Garrell, J. Otero, C. Romero, J. Bacardit, V. M. Rivas *et al.*, "Keel: a software tool to assess evolutionary algorithms for data mining problems," *Soft Computing*, vol. 13, no. 3, pp. 307–318, 2009.

[31] S. R. Garner *et al.*, "Weka: The waikato environment for knowledge analysis," in *Proceedings of the New Zealand computer science research students conference*. Citeseer, 1995, pp. 57–64.

[32] M. Scutari, "Learning bayesian networks with the bnlearn r package," *arXiv preprint arXiv:0908.3817*, 2009.