A New Method for Vertical Parallelisation of TAN Learning Based on Balanced Incomplete Block Designs[⋆]

Anders L. Madsen[1 2], Frank Jensen[1], Antonio Salmerón[3], Martin Karlsen[1], Helge Langseth[4], and Thomas D. Nielsen[2]

[1] HUGIN EXPERT A/S, Aalborg, Denmark
[2] Department of Computer Science, Aalborg University, Denmark
[3] Department of Mathematics, University of Almería , Spain
[4] Department of Computer and Information Science, Norwegian University of Science and Technology, Norway

Abstract. The framework of Bayesian networks is a widely popular formalism for performing belief update under uncertainty. Structure restricted Bayesian network models such as the Naive Bayes Model and Tree-Augmented Naive Bayes (TAN) Model have shown impressive performance for solving classification tasks. However, if the number of variables or the amount of data is large, then learning a TAN model from data can be a time consuming task. In this paper, we introduce a new method for parallel learning of a TAN model from large data sets. The method is based on computing the mutual information scores between pairs of variables given the class variable in parallel. The computations are organised in parallel using balanced incomplete block designs. The results of a preliminary empirical evaluation of the proposed method on large data sets show that a significant performance improvement is possible through parallelisation using the method presented in this paper.

Keywords: Bayesian networks, TAN, parallel learning

1 Introduction

A *Bayesian network* (BN) [5, 14–16, 19] is a powerful and popular model for probabilistic inference. Its graphical nature makes it well-suited for representing complex problems where the interactions between entities represented as variables are described using *conditional probability distributions*.

Structure restricted Bayesian network models such as the Naive Bayes (NB) Model [7] and Tree-Augmented Naive Bayes (TAN) Model [11] have shown impressive performance for solving classification tasks [7, 20, 24]. Data sets to be analysed using NB and TAN models are ever increasing in number and size.

[⋆] Published by Springer: Lecture Notes in Computer Science Volume 8754, 2014, pp 302-317. The final publication is available at link.springer.com. DOI: 10.1007/978-3-319-11433-0_20

The size increases both with respect to the number of variables in the data sets and the number of cases in each data set. Large data sets may challenge the efficiency of pure sequential algorithms for constructing NB and TAN models from data. On the other hand, the computational power of computers is increasing and access to computers supporting parallel processing is improving. This includes improved access to computers with multiple CPUs and multicore CPUs as well as high performance computers such as supercomputers. Therefore, there is an increasing need for algorithms supporting parallel processing. In this paper, we present a new method for parallel computing when learning a TAN model from large data sets, where variables are scored pairwise in parallel. Hence, this method focuses mainly on learning from data sets, where the number of feature variables is large. The challenge is to distribute the pairwise scoring of subsets of variables onto a set of *compute nodes*.

Balanced Incomplete Block (BIB) diagrams [23] are related to the statistical issue of design of experiments. In [23] the author writes *"Combinatorial design theory concerns questions about whether it is possible to arrange elements of a finite set into subsets so that certain balance properties are satisfied.*" When learning a TAN model from data with many feature variables, where the scoring is distributed onto a number of compute nodes, we need to arrange the features into subsets such that all pairs of variables are scored (at least) once. Design theory can provide one solution to this challenge. This paper describes how.

In [8] the authors describe a MapReduce-based method for learning Bayesian networks from massive data using a search & score algorithm while [4] describes a MapReduce-based method for machine learning on multicore computers. Also, [21] presents the R package **bnlearn** which provides implementations of some structure learning algorithms including support for parallel computing. [2] introduces a method for accelerating Bayesian network parameter learning using Hadoop and MapReduce. In this paper, we consider parallelisation of TAN learning on distributed memory concurrent computers using the standardized and portable message-passing system referred to as the *Message Passing Interface* (MPI) [10]. We employ the SPMD (Single Program, Multiple Data) technique to achieve parallelism in the learning of the TAN model from data through a MPI implementation. The implementation has a *master* process and a number of *worker* processes where the master process will also be a worker process. Tasks are divided into subtasks and run simultaneously on multiple processors with different input. The results of the subtasks are communicated to a master process, which collects the results and produces the final outcome.

This paper is organised as follows. Section 2 presents preliminaries and notation including an introduction to BIB designs. Section 3 describes the details of the proposed method for parallel TAN learning while Section 4 presents the results of a preliminary empirical evaluation. Finally, Section 5, Section 6 and Section 7 give a discussion, conclusions and outline future work, respectively.

2 Preliminaries and Notation

2.1 Bayesian Networks

Let $\mathcal{X} = \{X_1, \ldots, X_n\}$ be a set of discrete random variables such that $\text{dom}(X)$ is the state space of X and $||X|| = |\text{dom}(X)|$. A discrete BN $\mathcal{N} = (\mathcal{X}, G, \mathcal{P})$ over $\mathcal{X}$ consists of an acyclic directed graph (DAG) $G = (V, E)$ with vertices V and edges E and a set of CPDs $\mathcal{P} = \{P(X \,|\, \text{pa}(X)) : X \in \mathcal{X}\}$, where $\text{pa}(X)$ denotes the parents of X in G [5, 15, 19]. The discrete BN $\mathcal{N}$ specifies a joint probability distribution over $\mathcal{X}$ as

$$P(\mathcal{X}) = \prod_{i=1}^{n} P(X_i \,|\, \text{pa}(X_i)).$$

We only consider discrete Bayesian networks in this paper. We use upper case letters, e.g., X_i and Y, to denote variables and lower case letters, e.g., x_j and y, to denote states of variables. Sets of variables are denoted using calligraphic letters, e.g., $\mathcal{X}$ and $\mathcal{F}$.

A TAN model $\mathcal{T} = (\mathcal{X}, G, \mathcal{P})$ is a restricted type of BN with $\mathcal{X} = \{C\} \cup \mathcal{F}$ where C is a class variable and $\mathcal{F}$ is a set of feature variables and $G(\mathcal{F})$ is a tree where $G(\mathcal{X}')$ is the subgraph of G induced by $\mathcal{X}'$ and C is parent of each $F \in \mathcal{F}$.

Example 1. Figure 1 shows the graph G of a TAN model with $|\mathcal{F}| = n$ features where $\text{pa}(F_i) = \{F_{i-1}, C\}$ for $i = 2, \ldots, n$, $\text{pa}(F_1) = \{C\}$ and $\text{pa}(C) = \emptyset$.

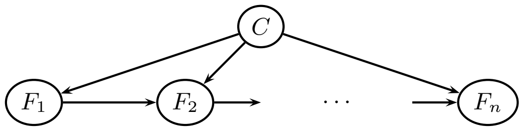


Fig. 1. A TAN model with n features.

The class variable C is a parent of each $F \in \mathcal{F}$ and each $F \in \mathcal{F}$ has at most one other parent. If C is removed from G, a tree is obtained over the remaining variables, i.e., $\mathcal{F}$.

2.2 Learning a TAN from Complete Data

Let $\mathcal{D} = (c_1, \ldots, c_N)$ denote a data set of N complete cases over variables $\mathcal{X} = \{C\} \cup \mathcal{F}$ where C is the classification variable and $\mathcal{F}$ is a set of n features. The task of constructing a TAN model over $\mathcal{X}$ from $\mathcal{D}$ basically amounts to finding a maximal weighted spanning tree over $\mathcal{F}$, directing edges such that each vertex has at most one parent and adding C as a parent of each $F \in \mathcal{F}$. The algorithm of [11] based on [3] is basically:

1. Compute mutual information $I(F_i, F_j \,|\, C)$ for each pair, $i \neq j$.

2. Build a complete graph G over $\mathcal{F}$ with edges annotated by $I(F_i, F_j \,|\, C)$.
3. Build a maximal spanning tree T from G.
4. Select a vertex and recursively direct edges outward from it.
5. Add C as parent of each $F \in \mathcal{F}$.

In order to build the complete graph G over $\mathcal{F}$, we need to compute $I(F_i, F_j \,|\, C)$ for $\binom{n}{2} = n(n-1)/2$ pairs where $n = |\mathcal{F}|$, i.e., there are n feature variables. When using multiple processes we need to determine how to distribute the scoring between processors to avoid computing the score of any pair more than once. The ultimate level of parallelisation would be to create one process for each pair to score. However, this may not be the most efficient approach in practice.

Once the structure G has been determined, the parameters of $\mathcal{P}$ are estimated. We assume data $\mathcal{D}$ is complete and do not consider the process of estimating $\mathcal{P}$ from $\mathcal{D}$. Thus, we focus only on determining the structure of G.

2.3 Balanced Incomplete Block Designs

The use of block designs dates back to the statistical theory of design of experiments [9], highly motivated in its origin by agricultural experiments. In such scenarios, the goal was to compare the yield of different plant varieties, considering that the yield could be significantly affected by the environment, i.e., the conditions under which the plants are grown. The idea was to remove the effect of the environment by setting up *blocks* of uniform environmental conditions, and distribute the plants among the blocks, as testing every plant in each block might potentially have an unaffordable cost. The term *balanced design* refers to the fact of keeping the probability that two varieties are compared (i.e., that they fall inside the same block) constant for every pair. BIB designs are used to distribute the pairwise scoring to obtain the highest level of parallelism making sure that all pairs are scored and no pair is scored more than once.

The work of testing for independence between pairs of variables should be distributed evenly among the processes (or processors) available. At the same time, we want each process to access as little data as possible (in order to minimize IO and memory usage). If we have n variables, p processes, and each process reads data for k variables, then the following inequality must always be satisfied in order to cover all pairs of variables

$$p\binom{k}{2} \geq \binom{n}{2}.$$

Solving for k produces $k \geq n\big/\sqrt{p}$, and equality holds only when $p = 1$.

In order to come as close as possible to this theoretical minimum, we must distribute the data among the processes in such a way that each pair of variables is assigned to exactly one process. We use BIB designs to achieve this. In Design Theory, a design is defined as:

Definition 1 (Design [23]). *A design is a pair $(X, \mathcal{A})$ s. t. the following properties are satisfied:*

1. X is a set of elements called points, and
2. $\mathcal{A}$ is a collection of nonempty subsets of X called blocks.

We only consider cases where each block is a set (and not a multiset) and each point will correspond to a subset of variables. A BIB design is defined as:

Definition 2 (BIB design [23]). *Let v, k and λ be positive integers s. t. $v > k \geq 2$. A (v, k, λ)-BIB design is a design $(X, \mathcal{A})$ s. t. the following properties are satisfied:*

1. $|X| = v$,
2. each block contains exactly k points, and
3. every pair of distinct points is contained in exactly λ blocks.

We use BIB designs to control the process of scoring pairs of feature variables. A point corresponds to a subset of feature variables and a process is created for each block. The number of blocks in a design is denoted b and r denotes the *replication number*, i.e., how often each point appears in a block. Property 3 in the definition is the *balance* property that we need. We only want to score each pair once and therefore require $\lambda = 1$. A BIB design is called incomplete as $k < v$. A BIB design where $v = b$ or $r = k$ is symmetric, i.e., the number of points equals the number of blocks or the replication number equals the block size. In a (v, k, λ)-BIB design, every point occurs in r blocks where $r = \lambda(v - 1) /(k - 1)$ and the number of blocks is $b = vr/k$ [23].

Example 2. Consider the $(7, 3, 1)$-BIB design. In this design, $b = 7 \cdot 3/3 = 7$ and $r = 1 \cdot (7 - 1)/(3 - 1) = 3$. Hence, each point appears in three blocks and there are seven blocks. The blocks are (one out of a number of possibilities):

$$\{123\}, \{145\}, \{167\}, \{246\}, \{257\}, \{347\}, \{356\}, \tag{1}$$

where $\{abc\}$ is shorthand notation for $\{a, b, c\}$. This BIB design is *symmetric* as the number of blocks equals the number of points. This will not be the case in general.

Examples of other designs that are known to exist include $(16, 20, 5, 4, 1)$, $(91, 91, 10, 10, 1)$ and $(871, 871, 30, 30, 1)$, using the notation (v, b, r, k, λ) for each BIB design [6].

There is no single method to construct all BIB designs. However, a difference set can be used to generate some symmetric BIB designs.

Definition 3 (Difference Set[23]). *Assume $(G, +)$ is a finite group of order v in which the identity element is 0. Let k and λ be positive integers such that $2 \leq k < v$. A (v, k, λ)-difference set in $(G, +)$ is a subset $D \subseteq G$ that satisfies the following properties:*

1. $|D| = k$,
2. the multiset $[x - y : x, y \in D, x \neq y]$ contains every element in $G \setminus \{0\}$ exactly λ times.

In our case, we are restricted to using $(\mathbb{Z}_v, +)$, the integers modulo v.

If $D \subseteq \mathbb{Z}_v$ is a difference set in group $(G, +)$, then $D + g = \{x + g | x \in D\}$ is a translate of D for any $g \in G$. The multiset of all v translates of D is denoted $Dev(D)$ and called the development of D [23], page 42.

Theorem 1 ([23], Theorem 3.8 p. 43). *Let D be a (v, k, λ)-difference set in an Abelian group $(G, +)$. Then $(G, Dev(D))$ is a symmetric (v, k, λ)-BIB design.*

Example 3. The set $D = \{0, 1, 3\}$ is a $(7, 3, 1)$-difference set in $(\mathbb{Z}_7, +)$. The blocks constructed by iteratively adding one to each element of D (modulo 7) are:

$$\{013\}, \{124\}, \{235\}, \{346\}, \{450\}, \{561\}, \{602\}. \tag{2}$$

Notice that the ith element of each block is unique across all blocks. This property is used to assign blocks to processes.

Table 1 [12] shows difference sets for a set of symmetric BIB designs. The corresponding BIB design blocks are constructed as illustrated in Example 3. Notice that the first element of each difference set is unique. This means that the first element of each block can be used to associate process ranks and blocks.

Table 1. Difference sets for a set of symmetric BIB designs.

BIB design	Difference set	k/v
(3,2,1)	(0,1)	0.67
(7,3,1)	(0,1,3)	0.43
(13,4,1)	(0,1,3,9)	0.31
(21,5,1)	(0,1,4,14,16)	0.24
(31,6,1)	(0,1,3,8,12,18)	0.19
(57,8,1)	(0,1,3,13,32,36,43,52)	0.14
(73,9,1)	(0,1,3,7,15,31,36,54,63)	0.12
(91,10,1)	(0,1,3,9,27,49,56,61,77,81)	0.11
(133,12,1)	(0,9,10,12,26,30,67,74,82,109,114,120)	0.09
(183,14,1)	(0,12,19,20,22,43,60,71,76,85,89,115,121,168)	0.08

Notice how the block size k increases and that some values are *missing* in the sequence. That is, there is no symmetric BIB design for $k = 7$ with $\lambda = 1$. We know that a symmetric BIB design exists when $k - 1$ is a *prime power* and a conjecture states that a symmetric BIB design exists only when this is the case. For instance, $8 - 1 = 7^1$ and $9 - 1 = 2^3$ whereas $7 - 1 = 6 = 2 * 3$ [13, 12].

3 Parallelisation of TAN Learning

There are two obvious approaches to parallelise the TAN learning algorithm described in Section 2.2. One approach is to assign the same number of cases to each process. Each process would then count the configurations of all pairs of variables (together with the class variable) in the data assigned to the process. The counts from all processes are combined and used to perform the pairwise scoring. We refer to this as *horizontal* parallelisation. This approach to horizontal parallelization is *embarrassingly* parallel, i.e., it requires little effort to separate the problem into a number of parallel tasks.

A second approach (and the one investigated in this paper) is to distribute the scoring to the processes. The idea is to assign a set of variables to each process such that each pair of variables is assigned to exactly one process as we need to score each pair of variables at least once. This is only possible for certain combinations of numbers of variables and processes. We refer to this as vertical parallelisation.

Horizontal parallelisation mainly addresses learning from data sets with many cases whereas vertical parallelization mainly addresses learning from data sets with many feature variables. Horizontal and vertical parallelization can be combined to cope with data sets where both N and $|\mathcal{F}|$ are large. In this paper, we focus only on vertical parallelisation.

3.1 Parallel Scoring Using BIB Designs

In learning the structure of a TAN model, each pair of variables $X_i, X_j \in \mathcal{F}$ should be scored for mutual information given C (at least once). The task of calculating these scores in parallel can be solved using BIB designs. That is, BIB designs are used to control the process of scoring all pairs of features in Step 1 of the algorithm in Section 2.2, i.e., computing the mutual information between $F_i, F_j \in \mathcal{F}$ given the class variable C. This means that BIB designs are used to divide $\mathcal{F}$ into subsets to be assigned to each process. Each process will score pairs of features assigned to it.

Fisher's inequality states that $b \geq v$ [23] (who cites [9]). That is, no design with $b < v$ is possible. This means that the number of blocks b is larger than or equal to the number of points v. On the other hand, $|\mathcal{F}|$ is usually much larger than the number of processors available. This means that each point should represent a subset of variables, i.e., each point p represents a set of variables $\mathcal{F}_p \subseteq \mathcal{F}$. We do not include the class variable C in the set of points. As we need to score pairs exactly once, we are only interested in designs with $\lambda = 1$.

A separate process with a unique rank is created for each block where the rank is a number from zero to the number of processes minus one. Each process computes the pairwise scores represented by the block as described below. This means that ideally the number of blocks should match the number of processes and each point in all blocks should represent the same number of features. This may not be possible in practice as BIB designs for any combination of v and k do not necessarily exist. Instead either some processors will be idle, more blocks than processes can be created or idle processors can be used for other tasks such as horizontal parallelisation.

The process of computing the scores in Step 1 of the algorithm in Section 2.2 can be organized and distributed using a BIB design. Each process computes the score for each pair of features from different points. This is referred to as inter-point scoring. This means that all variables in different points are scored. In addition, each process computes the score for each pair of features in a unique point. This is referred to as intra-point scoring and ensures that all pairs are scored exactly once. This is demonstrated by the next example continuing Example 2.

Example 4. In the $(7, 3, 1)$-BIB design, each point $p = 1, \ldots, 7$ represents a subset $\mathcal{F}_p \subseteq \mathcal{F}$. If we assume $|\mathcal{F}| = 140$, then $|\mathcal{F}_p| = 20$, i.e., each point represents 20 features. As $k = 3$ each process is assigned 60 features, but each process does not score all pairs as described below. The seven blocks $(b = 7)$ of the $(7, 3, 1)$-BIB design are shown in (1).

Example 5. Consider again the $(7, 3, 1)$-BIB design and assume $|\mathcal{F}| = 14$. This means that each point represents two features. Each process is assigned six features. The seven blocks $(b = 7)$ of the $(7, 3, 1)$-BIB design are shown in (1).

Figure 2 is a graphical illustration of how BIB designs are used to calculate the scores in parallel using seven processes and assuming $|\mathcal{F}| = 14$. Each process is assigned a block containing three points. Each point represents two variables.

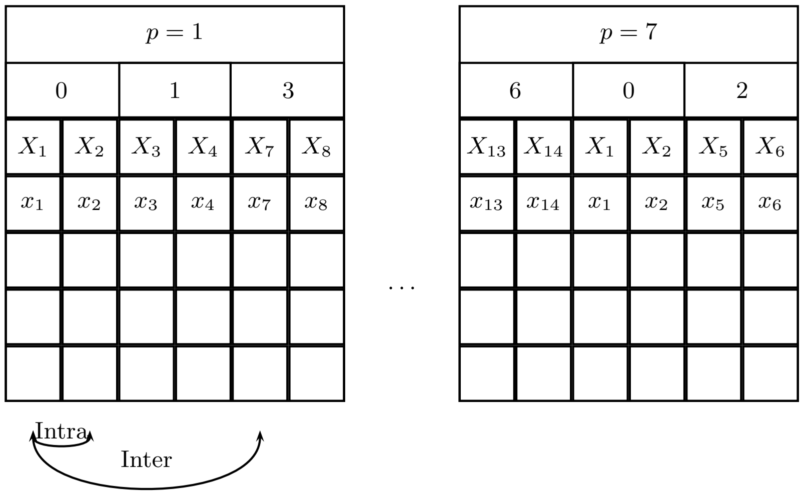

$p = 1$
0
1
3
X_1 X_2 X_3 X_4 X_7 X_8
x_1 x_2 x_3 x_4 x_7 x_8
$p = 7$
6
0
2
X_{13} X_{14} X_1 X_2 X_5 X_6
x_{13} x_{14} x_1 x_2 x_5 x_6
...
Intra
Inter

Fig. 2. Illustration of how BIB designs are used to parallelise the pairwise scoring.

The figure illustrates how process $p = 1$ performs both inter-point and intra-point calculations. The block assigned to process $p = 1$ is $\{013\}$. Each point represents a unique pair of variables, e.g., point 0 represents the set $\{X_1, X_2\}$. Calculating the score for X_1 and X_2 is an intra-point operation whereas calculating the score for X_1 and X_3 is an inter-point score as X_1 and X_3 are in different points. The challenge is to make sure that all processes perform the same number of computations.

Notice that each process reads $k/v = 3/7 = 43\%$ of the feature data in addition to the class data.

Each process p computes the score for all pairs of feature variables X_i, X_j where X_i and X_j belongs to subsets represented by different points in the block represented by p. These are the inter-point operations. In addition, each process p computes the score for all pairs of feature variables X_i, X_j where X_i and X_j belong to the subset represented by the first point in the block represented by p. These are the intra-point operations. In this way, all pairs are scored exactly once.

3.2 Generating Symmetric BIB Designs with $\lambda = 1$

Symmetric BIB designs with $\lambda = 1$ can be generated using difference sets as described in Section 2.3 where Table 1 shows the difference sets needed to generate symmetric BIB designs with $\lambda = 1$ for $k \leq 14$. Each process generates its corresponding block using its unique rank by adding the rank to each element of the difference set modulus the number of processes created.

3.3 Theoretical Performance Improvement

Each process p represents one block of points. If $v < |\mathcal{F}|$, which is usually the case, then each point represents a subset $\mathcal{F}_p \subseteq \mathcal{F}$. Here we assume that each point represents the same number of features, i.e., we assume each point to represent m feature variables. This means that each process p performs $\binom{k}{2}m^2$ inter-point calculations, where k is the block size and m is the number of feature variables in each point. Each process p performs $\binom{m}{2}$ intra-point calculations.

Using a (v, k, λ)-BIB design each process will have to read k/v of the data set in order to calculate the scores assigned to the process. If $v = 7$ and $k = 3$ (as in Example 2), then each process reads $3/7 = 43\%$ of the data. If $v = 91$ and $k = 10$ (as in Example 2), then each process reads $10/91 = 11\%$ of the data. The last column of Table 1 shows the amount of feature data read by each process for $p \in \{3, 7, 13, 21, 57, 73, 91, 133, 183\}$. All feature data are read for $p = 1$, which is equivalent to a pure sequential algorithm.

Example 6. Assume a $(13, 4, 1)$-BIB design with $m = 10$ where m is the number of feature variables in each point, i.e., $|\mathcal{F}| = 130$. Each process performs $\binom{4}{2}10^2 + \binom{10}{2} = 600 + 45 = 645$ calculations and reads 40 data files (in addition to the data file containing the class variable) out of 130 data files, i.e. $41/131 = 31\%$ of $\mathcal{D}$. In total $13 \cdot 41 = 533$ files are read by the 13 processes.

A pure sequential implementation will read all data, i.e., 130 feature data files and one class data file, and score $\binom{130}{2} = 8385$ pairs of feature variables. That is, each process in the parallel program computes 8% of the scores computed by the single process in a pure sequential application.

Although the proposed method achieves a linear (in the number of processes) speed-up in the number of calculations (mutual information scores) performed by a single process, it only achieves a speed-up of the square root of the number of processes in the amount of data needed by a single process. This is optimal for vertical parallelisation as explained in Section 2.3.

4 Empirical Evaluation

This section reports on a preliminary empirical evaluation of the proposed parallel TAN learning algorithm.

4.1 Data Sets

Three different sources of data sets were considered in the empirical evaluation. Random samples were generated from two real-world Bayesian networks of different sizes, i.e., the Munin1 [1] and Munin2 [1] networks. For each of these networks a variable was arbitrarily chosen as class variable. The third source of data was a sample generated from a real-world financial data set.

Table 2. Data sets used in the experiments.

data set	$\lvert\mathcal{X}\rvert$	N
Munin1	189	750,000
Munin2	1,003	750,000
Bank	1,823	1,140,000

Table 2 describes properties of the data sets used in the experiments. Munin1 and Munin2 are data sets of 750,000 cases generated from the Munin1 and Munin2 networks, respectively, while Bank is a data set with 1,140,000 cases over financial data. Bank is an artificial data set generated from a real-world data set maintaining some of the statistical properties of the original data. Variable names and values have been anonymised. Continuous variables were discretized into five bins. All data sets used in the empirical evaluation are complete, i.e., there are no missing values in the data.

4.2 Hardware

The empirical evaluation was performed on three different computer systems. One Linux server and two supercomputers Fyrkat and Vilje both running Linux:

1. A linux server running Ubuntu (kernel 2.6.38-11-server) with a four-core Intel Xeon(TM) E3-1270 Processor and 32 GB RAM.

2. Fyrkat[5] is a computer cluster where each worker node used has 2 Intel Xeon (TM) X5260 Processors and 16GB RAM. It has a total of 80 such nodes. This cluster system uses SLURM (simple Linux Utility for Resource Management) for resource management.
3. Vilje[6] is a computer cluster where each worker node has dual eight-core Xeon E5-2670 Processors and 32GB. It has a total of 1404 such nodes. This cluster system uses PBS (Portable batch System) for resource management.

The algorithms were implemented using HUGIN software version 8.0 [17, 18] and MPI. The HUGIN software does not have any special features necessary for implementing the ideas presented in this paper. It is important to notice that the experiments were performed when the system was being used by other users and running other applications. This is likely to impact performance and produce a higher variance in execution times than if the experiments were performed on a dedicated system.

4.3 Scoring Function

In the implementation, the score $I(X_i, X_j \,|\, C)$ was computed from the *likelihood* test statistics $G = 2 * \sum_i O_i \cdot \log(\frac{O_i}{E_i})$, where O_i is the observed frequency and E_i is the expected frequency under the null hypothesis. Since $G = 2 \cdot N \cdot I(X_i, X_j)$, the mutual information score can be computed as $I(X_i, X_j) = G/(2 \cdot N)$.

If the counts computed by each process are stored and communicated to the master process, then (assuming complete data) the parameters of the conditional probability distribution can be estimated.

4.4 Evaluations

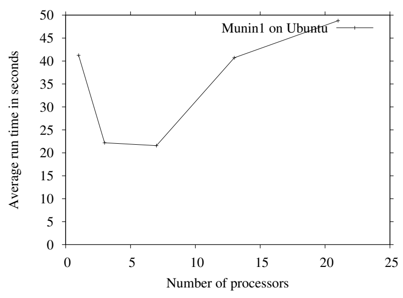
The parallel algorithm was implemented employing the SPMD model. The system has a master process and a number of worker processes. Each process has a unique identifier referred to as its rank. The process of rank zero is referred to as the master process. Each process, including the master process, reads data for the feature variables assigned to it and the class variable. Each variable is stored in a single file. This means that each process only reads data for its assigned features and the class variable. The block assigned to a process is uniquely identified using the rank of the process and the difference set (each process knows the number of processes created). The unique block of process p is calculated using the rank of p and the difference set (see Theorem 1). All processes including the process of rank zero compute the mutual information for its assigned pairs and communicate the results back to the process of rank zero. The process of rank zero collects the results and creates the maximum spanning tree (Step 3 - 5). These last steps are very fast to perform compared to data reading and scoring.

[5] http://fyrkat.grid.aau.dk
[6] https://www.hpc.ntnu.no/display/hpc/Vilje

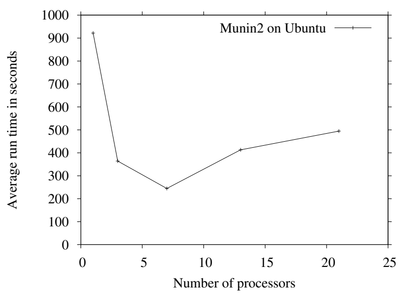
The average computation time was calculated over ten runs with the same data. The computation time was measured as the elapsed (wall-clock) time between two specific points in the program. Time was measured in the master process from before it started reading data for its assigned features until the scoring was completed and all results communicated to the master process. We also report the time used for reading data and performing the scoring for the process of rank zero to get an indication of the division of work between reading data and computing scores as well as to verify the speed-up obtained.

4.5 Results

This section reports on the results of the empirical evaluation of the proposed method for vertical parallelisation of learning the structure of a TAN. Experiments were performed using the three data sets described in Section 4.1 and three systems described in Section 4.2.



(a) Munin1



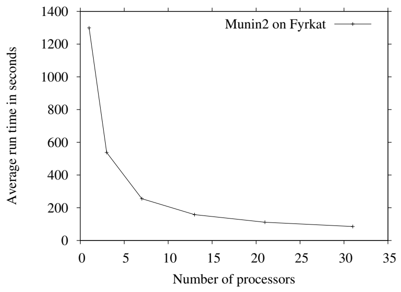
(b) Munin2

Fig. 3. Average run times for Munin1 and Munin2 on Ubuntu.

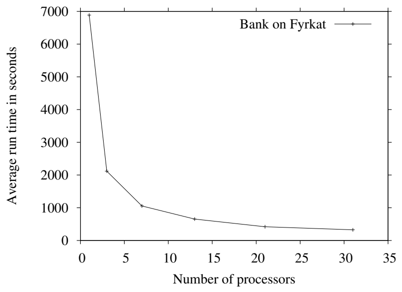
Figure 3 (left) shows the average run time in seconds for Munin1 on Ubuntu while Fig. 3 (right) shows the average run time in seconds for Munin2 on Ubuntu. The figure shows that performance improved up to seven processes. For 13 and 21 processes performance deteriorated. This is expected as Ubuntu has only four physical cores (and eight logical cores).

On Fyrkat data files were assumed mounted on the compute nodes before executing the application. Figure 4 (left) shows the average running time for Munin2 on Fyrkat while Fig. 4 (right) shows the average running time for Bank on Fyrkat. It is clear that the average running time improved as the number of blocks, i.e., processors used, increased. The performance should be expected to deteriorate if the number of blocks is higher than the number of processors used.

Figure 5 shows the average running time on Vilje for Munin2 (left) and Bank (right). It is clear that the average running time improved as the number of blocks, i.e., processors used, increased.

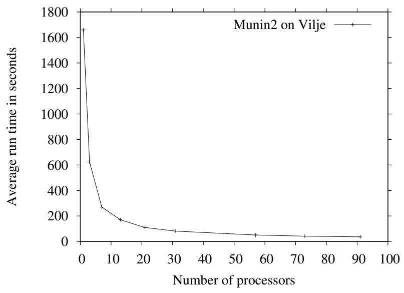


(a) Munin2

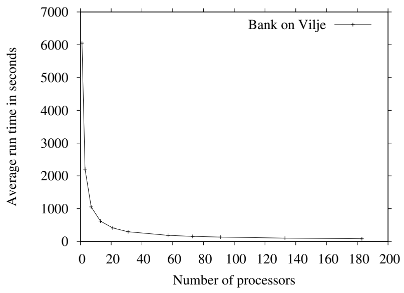


(b) Bank

Fig. 4. Average run times for Munin2 and Bank on Fyrkat.



(a) Munin2



(b) Bank

Fig. 5. Average run times for Munin2 and Bank on Vilje.

Figure 6 (left) shows the speed-up factors for reading data (*Time*) and the *theoretical* number of files read by each process (*Files*) relative to the case of one processor as well as the square root of the number of processors (*Optimal*). The *theoretical* number of files read by each process is computed as $k/v \cdot |\mathcal{X}|$. Figure 6 (right) shows the speed-up factor for scoring as a function of the number of processors used.

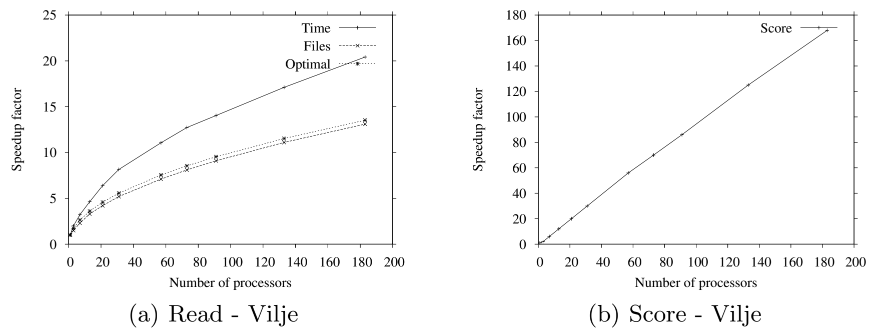


Fig. 6. Speed up as a function of the number of processors (relative to one processor).

It is clear from Fig. 6 that the theoretical number of files read by each process as expected follows the square root of the number of processes. Similarly, the speed-up factor for time used on testing is approximately a linear function of the number of processors used. It is more surprising that the measured time performance shows a higher speed-up that the square root of the number of processes. The number of times a file was read by different processes increased as the number of processors used increased. This means that the impact of *caching* files increased with the number of processors used. We believe that this explains the unexpected observed behaviour.

5 Discussion

This paper has introduced a new method for vertical parallelisation of TAN learning based on using symmetric BIB designs with $\lambda = 1$. The use of BIB designs makes it possible to achieve a work *balance* between processes such that all processes score (almost) an equal number of pairs of feature variables. Symmetric BIB designs with $\lambda = 1$ do not exist for all values of v and k. Table 3 shows symmetric BIB designs with $\lambda = 1$ for $k \leq 14$. Difference sets for symmetric BIB designs with $\lambda = 1$ for much higher k are known, see e.g.,[12]. If the number of processors (or cores) does not match with a block size for which a symmetric BIB design with $\lambda = 1$ is known to exist, then idle processors can be used for horizontal parallelisation or parallelisation of the counting process.

Difference sets for b up to 1893 are implemented and this can be increased as needed taking the prime power conjecture into account.

The results of the experimental evaluation show a clear time performance improvement as the number of blocks, i.e., processors used, is increased. Notice that even on a single CPU machine with multiple cores, a performance improvement is achieved. For this system performance deteriorates when the number of blocks is higher than the number of logical cores in the CPU. This should be expected. There is some variance in the run time measured. This should also be expected as the evaluation is performed on systems serving other users, i.e., the experiments have not been performed on isolated systems.

Notice that the performance evaluation has been performed on three different systems. From a personal computer running as a Linux server to powerful supercomputers using different types of resource management systems. The results of the experiments show that a performance improvement can be realised on each of these types of systems taking advantage of parallel computation. The empirical evaluation has been performed using data sets of different complexity both with respect to the number of variables and the number of cases.

6 Conclusion

This paper introduces a new method to vertical parallelisation of learning the structure of a TAN model from data. The approach is based on the use of BIB designs to distribute computing pairwise mutual information between features given the class variable.

The results of an empirical evaluation of the proposed method on desktop as well as supercomputers show a significant time performance improvement over the pure sequential method.

7 Future Work

The principle of vertical parallelisation of pairwise scoring introduced in this paper can be applied to the process of structure learning of a Bayesian network using, e.g., the PC algorithm [22]. In the PC algorithm all variables are initially tested for pairwise independence, which is similar to the scoring of all pairs of $\mathcal{F}$. In addition, a set of conditional independence tests are performed. The principles of vertical parallelisation can be applied in both cases. The plan for future work includes investigating how BIB designs can be applied to perform the conditional independence tests in parallel.

Parallelising the counting process (horizontal parallelisation) can be considered as orthogonal to the pairwise scoring (vertical parallelisation). This means that the methods can be combined to achieve even further performance improvements when data sets are extremely large, i.e., do not fit into main memory of the computer. Furthermore, we plan to investigate options for taking advantage of multithreaded programming solving the tasks assigned to each process.

This may include both data reading for horizontal parallelisation of the counting scheme as well as inter and intra pairwise conditional independence testing. Furthermore, BIB designs can also be applied to improve the performance of the pairwise scoring by a single process.

Acknowledgments

This work was performed as part of the AMIDST project. AMIDST has received funding from the European Union's Seventh Framework Programme for research, technological development and demonstration under grant agreement no 619209. The data set Bank has kindly been provided for testing by Cajamar.

References

1. S. Andreassen, F. V. Jensen, S. K. Andersen, B. Falck, U. Kjærulff, M. Woldbye, A. R. Sørensen, A. Rosenfalck, and F. Jensen. MUNIN — an expert EMG assistant. In John E. Desmedt, editor, *Computer-Aided Electromyography and Expert Systems*, chapter 21. Elsevier Science Publishers, Amsterdam, 1989.
2. A. Basak, I. Brinster, X. Ma, and O.J. Mengshoel. Accelerating Bayesian network parameter learning using Hadoop and MapReduce. In *Proceedings of the 1st International Workshop on Big Data, Streams a nd Heterogeneous Source Mining: Algorithms, Systems, Programming Models and Applications*, pages 101–108, 2012.
3. C.K. Chow and C.N. Liu. Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory*, IT-14(3):462–467, 1968.
4. C.-T. Chu, S.K. Kim, Y.-A. Lin, Y. Yu, G. Bradski, A.Y. Ng, and K. Olukotun. Map-reduce for machine learning on multicore. In *NIPS*, pages 281–288, 2006.
5. R.G. Cowell, A.P. Dawid, S.L. Lauritzen, and D.J. Spiegelhalter. *Probabilistic Networks and Expert Systems.* Springer, 1999.
6. J.W. Di Paola, J.S. Wallis, and W.D. Wallis. A list of (v,b,r,k,λ) designs for $r \leq 30$. In *Proc. 4th S-E Cont. Combinatorics, graph theory and computing*, pages 249–258, 1973.
7. P. Domingos and M. Pazzani. On the optimality of the simple Bayesian classifier under zero-one loss. *Machine Learning*, pages 103–130, 1997.
8. Q. Fang, K. Yue, X. Fu, H. Wu, and W. Liu. A MapReduce-Based Method for Learning Bayesian Network from Massive Data. In Yoshiharu Ishikawa, Jianzhong Li, Wei Wang, Rui Zhang, and Wenjie Zhang, editors, *Web Technologies and Applications*, volume 7808 of *Lecture Notes in Computer Science*, pages 697–708. Springer Berlin Heidelberg, 2013.
9. R.A. Fisher. An examination of the different possible solutions of a problem in incomplete blocks. *Annals of Eugenics*, pages 52–75, 1940.
10. The MPI Forum. MPI: A Message Passing Interface, 1993.
11. N. Friedman, D. Geiger, and M. Goldszmidt. Bayesian network classifiers. *Machine Learning*, pages 1–37, 1997.
12. D.M. Gordon. La Jolla Difference Set Repository. URL=http://www.ccrwest.org/diffsets/diff_sets/ [Accessed 15 May 2014].
13. D.M. Gordon. The Prime Power Conjecture is True for $n < 2000000$. *Electronic J. Combinatorics*, 1(1, R6):1–7, 1994.

14. F.V. Jensen and T.D. Nielsen. *Bayesian Networks and Decision Graphs.* Springer, 2nd edition, 2007.
15. U.B. Kjærulff and A.L. Madsen. *Bayesian Networks and Influence Diagrams: A Guide to Construction and Analysis.* Springer, 2nd edition, 2013.
16. D. Koller and N. Friedman. *Probabilistic Graphical Models — Principles and Techniques.* MITPress, 2009.
17. A.L. Madsen, F. Jensen, U.B. Kjærulff, and M. Lang. HUGIN - The Tool for Bayesian Networks and Influence Diagrams. *International Journal on Artificial Intelligence Tools 14*, 3:507–543, 2005.
18. A.L. Madsen, M. Lang, U.B. Kjærulff, and F. Jensen. The Hugin Tool for Learning Bayesian Networks. In *Proc. of ECSQARU*, pages 549–605, 2003.
19. J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference.* Series in Representation and Reasoning. Morgan Kaufmann Publishers, San Mateo, CA, 1988.
20. I. Rish. An empirical study of the naive Bayes classifier. In *IJCAI workshop on Empirical Methods in AI*, pages 41–46, 2001.
21. M. Scutari. Learning Bayesian Networks with the bnlearn R Package. *Journal of Statistical Software*, 35(3):1–22, 2010.
22. P. Spirtes, C. Glymour, and R. Scheines. *Causation, Prediction, and Search.* Adaptive Computation and Machine Learning. MIT Press, second edition, 2000.
23. D. Stinson. *Combinatorial Designs — Constructions and Analysis.* Springer, 2003.
24. N. L. Zhang. Hierarchical latent class models for cluster analysis. *Journal of Machine Learning Research*, 5:697–723, 2004.