Parallel Filter-Based Feature Selection Based on Balanced Incomplete Block Designs

Antonio Salmerón[1] and **Anders L. Madsen**[2] and **Frank Jensen**[3] and **Helge Langseth**[4] and **Thomas D. Nielsen** [5] and **Darío Ramos-López**[6] and **Ana M. Martínez**[7] and **Andrés R. Masegosa**[8]

Abstract. In this paper we propose a method for scaling up filter-based feature selection in classification problems. We use the conditional mutual information as filter measure and show how the required statistics can be computed in parallel avoiding unnecessary calculations. The distribution of the calculations between the available computing units is determined based on balanced incomplete block designs, a strategy first developed within the area of statistical design of experiments. We show the scalability of our method through a series of experiments on synthetic and real-world datasets.

1 INTRODUCTION

For datasets with a high number of variables, a key step in developing classification models is to determine the most informative variables for the model and exclude the less informative ones. Furthermore, some of the variables may be redundant in the context of others. Learning classifiers with an excessive number of variables may yield models overfitting irrelevant aspects of the data, and therefore showing a poor predictive power. Also, the complexity of learning usually grows with the number of variables, and hence the computational cost of learning with all the available variables may become infeasible.

Feature selection methods can be grouped into three main blocks, *wrapper*, *filter*, and *embedded* methods [17, 19, 1]. There are also mixed filter-wrapper approaches, some of which are able to operate in high dimensional domains [5, 4].

Wrapper methods [15] explore the space of possible feature subsets, optimizing some model-dependent metric like accuracy. It means that for each explored feature subset, a model is learnt and evaluated. Traditional wrapper approaches for feature selection are often time consuming, and therefore not viable within the context of learning from large amounts of data or in high dimensionality domains. Embedded methods [18] are also model-dependent. They use some specific property of the target model to guide the search procedure. On the other hand, filter methods [10] are independent of the underlying model, and can be seen as a pre-processing step performed before model learning. Univariate filters use some score function to produce a ranking of the features. Sometimes a threshold is defined and the features that score below it are discarded, other times a maximum number of features could be retained. There are also methods based on multivariate filters, which attempt to asses the goodness of a feature subset rather than a single feature [24].

In this paper we focus on univariate filter methods, as their low computational complexity, compared to wrapper schemes, make them appropriate for problems involving a high number of variables. We propose a method for scaling up filter-based feature selection in classification problems. We use the conditional mutual information as filter measure and show how the required statistics can be computed in parallel avoiding unnecessary calculations. The distribution of the calculations between the available computing units is determined based on balanced incomplete block designs, a strategy first developed within the area of statistical design of experiments.

2 FILTER-BASED FEATURE SELECTION

Some of the most outstanding filter methods are based on the use of information-theoretic score functions, all of them related to the concept of *entropy* [8]. Throughout this paper we shall assume a set of discrete variables $\boldsymbol{X} = \{X_1, \ldots, X_n\}$ and a class variable C. The entropy of a discrete variable $X \in \boldsymbol{X}$ is

$$H(X) = -\sum_{x \in \Omega_X} p(x) \log p(x).$$

The entropy measures the uncertainty in the distribution of X. Given two variables $X_i, X_j \in \boldsymbol{X}$, the *conditional entropy* of X_i given X_j is

$$H(X_i|X_j) = -\sum_{x_j \in \Omega_{X_j}} p(x_j) \sum_{x_i \in \Omega_{X_i}} p(x_i|x_j) \log p(x_i|x_j),$$

and it quantifies the uncertainty that remains in the distribution of X_i after observing X_j.

The amount of information shared by two variables $X_i, X_j \in \boldsymbol{X}$ can be measured by their *mutual information* [25, 8]:

$$I(X_i, X_j) = H(X_i) - H(X_i|X_j).$$

Note that the mutual information is symmetric, $I(X_i, X_j) = I(X_j, X_i)$. It can be interpreted as the amount of uncertainty in X_i that is removed after observing X_j. Similarly, given $X_i, X_j, X_k \in \boldsymbol{X}$ the *conditional mutual information* between X_i and X_j given X_k can be defined as

[1] University of Almería, Spain, email: antonio.salmeron@ual.es
[2] HUGIN EXPERT A/S and Aalborg University, Denmark, email: alm@hugin.com
[3] HUGIN EXPERT A/S, Aalborg, Denmark, email: fj@hugin.com
[4] Norwegian University of Science and Technology, Trondheim, Norway, email: helgel@idi.ntnu.no
[5] Aalborg University, Denmark, email: tdn@cs.aau.dk
[6] University of Almería, Spain, email: dramoslopez@ual.es
[7] Aalborg University, Denmark, email: ana@cs.aau.dk
[8] Norwegian University of Science and Technology, Trondheim, Norway, email: andresrm@idi.ntnu.no

$$I(X_i, X_j|X_k) = H(X_i|X_k) - H(X_i|X_j, X_k).$$

Alternatively, the conditional mutual information can be written as

$$I(X_i, X_j|X_k) = \sum_{x_i, x_j, x_k} p(x_i, x_j|x_k) \log \frac{p(x_i, x_j|x_k)}{p(x_i|x_k)p(x_j|x_k)}.$$

The mutual information is defined analogously in the multivariate setting for random vectors $\boldsymbol{X}_i$, $\boldsymbol{X}_j$ and $\boldsymbol{X}_k$ as

$$I(\boldsymbol{X}_i, \boldsymbol{X}_j|\boldsymbol{X}_k) = \sum_{\boldsymbol{x}_i, \boldsymbol{x}_j, \boldsymbol{x}_k} p(\boldsymbol{x}_i, \boldsymbol{x}_j|\boldsymbol{x}_k) \log \frac{p(\boldsymbol{x}_i, \boldsymbol{x}_j|\boldsymbol{x}_k)}{p(\boldsymbol{x}_i|\boldsymbol{x}_k)p(\boldsymbol{x}_j|\boldsymbol{x}_k)}.$$

A thorough analysis of information-theoretic filter methods for variable selection is reported in [6]. The analysis is based on the *conditional likelihood* of the class variable given $\boldsymbol{S}$ (features included in the model) and $\boldsymbol{\tau}$ (parameters of the distributions involved in the model) for a data set $\mathcal{D} = \{(\boldsymbol{x}^i, c^i), i = 1, \ldots, n\}$, defined as

$$\mathcal{L}(\boldsymbol{S}, \boldsymbol{\tau}|\mathcal{D}) = \prod_{i=1}^{n} q(c^i|\boldsymbol{x}^i, \boldsymbol{\tau}),$$

where q is the learnt model. It is shown in [6] that the conditional likelihood is maximized by minimizing $I(\boldsymbol{X} \setminus \boldsymbol{S}, C|\boldsymbol{S})$. In other words, the conditional likelihood is maximized when the mutual information between the class and the features not included in the model, given the variables actually included, is minimized.

A general framework for filter-based feature selection can be established by making two technical assumptions about the independence structure of the domain [6]: For $X_i, X_j \in \boldsymbol{S}$, $X_k \in \boldsymbol{X} \setminus \boldsymbol{S}$ it holds that X_i and X_j are conditionally independent both when conditioning on X_k and on $\{X_k, C\}$. The framework is based on selecting features greedily, using a filter measure defined as

$$J_{\text{cmi}}(X_i) = I(X_i, C|\boldsymbol{S}), \tag{1}$$

where X_i is the candidate variable to include in the model.

Utilizing the assumptions mentioned above, $J_{\text{cmi}}(X_i)$ can also be calculated as

$$J_{\text{cmi}}(X_i) = I(X_i, C) - \sum_{X_j \in \boldsymbol{S}} \big(I(X_i, X_j) - I(X_i, X_j|C)\big). \tag{2}$$

The advantage of the definition in Equation (2) compared to Equation (1) is that the former does not require the computation of the conditional mutual information over multi-dimensional random variables. Several existing filter-based methods can be shown to be related to Equation (2). A remarkable example is the *joint mutual information* filter measure [28], defined as

$$\begin{aligned} J_{\text{jmi}}(X_i) &= \sum_{X_j \in \boldsymbol{S}} I(\{X_i, X_j\}, C) \\ &= I(X_i, C) - \frac{1}{|S|} \sum_{X_j \in \boldsymbol{S}} \big(I(X_i, X_j) - I(X_i, X_j|C)\big), \end{aligned} \tag{3}$$

whose relation to Equation (2) is clear. Among the measures tested in [6], J_{jmi} is the one showing the best accuracy/stability tradeoff, and is therefore the one we utilize in the following. Note, however, that our approach works equally well for other measures, like J_{cmi} (when calculated using Equation (2)).

2.1 Scaling up filter based feature selection

Filter methods usually rank the features independently from each other. This is the approach taken in [22], where vertical parallelization is used. By *vertical parallelization* we mean distributing the variables among the computing units, corresponding to splitting the dataset by columns, while *horizontal parallelization* refers to splitting the dataset by rows and including all the variables in each split. In [22] different subsets of candidate variables are distributed through the available computing units, where the feature selection filters are actually applied. The selection is carried out through a voting process, where first the features that are discarded in each computing unit receive one vote, after which the features that have received the most votes are excluded from the final model.

The approach we adopt in this paper is different, as we propose to filter the variables globally and ensure scalability by distributing the calculations between the available computing units. More precisely, we follow the filter procedure described in Algorithm 1. It is based on a greedy strategy intended to select the feature that maximizes the joint mutual information measure in Equation (3). The filter measure is based on the mutual and conditional mutual information of each pair of features, computed in Steps 3 to 9. The variables are then included one by one (Steps 12 to 19) until a given maximum number of features have been selected.

```
1  Function Filter(X,C,M)
   Input: The set of features, X = {X_1, ..., X_N}. The class
          variable, C. The maximum number of features to be
          selected, M.
   Output: S, a set of selected features.
2  begin
3      for i ← 1 to N do
4          Compute I(X_i, C);
5          for j ← i + 1 to N do
6              Compute I(X_i, X_j|C);
7              Compute I(X_i, X_j);
8          end
9      end
10     X* ← arg max_{1≤i≤N} I(X_i, C);
11     S ← {X*};
12     for i ← 1 to M − 1 do
13         R ← X \ S;
14         for X ∈ R do
15             Compute J_jmi(X) as in Equation (3) using the
               statistics computed in Steps 4, 6, and 7;
16         end
17         X* ← arg max_{X∈R} J_jmi(X);
18         S ← S ∪ {X*};
19     end
20     return S;
21 end
```

Algorithm 1: Filter-based feature selection based on conditional likelihood maximization.

Our proposal for scaling up Algorithm 1 consists of parallelizing Steps 4 to 9, which involve the calculation of the (conditional) mutual information between each pair of variables. These steps are also the most computationally demanding when the probabilities involved in the computations are estimated from data. An immediate approach for scaling up the algorithm could be to simply generate

one computing thread for each pair of variables and then process the threads in parallel. However, with n variables this approach would require accessing the underlying database $\binom{n}{2}$ times, inducing a significant overhead in terms of disk/network access. Alternatively, one might group the variables in blocks so that each block only accesses the data a single time in order to calculate the sufficient statistics required for estimating the (conditional) mutual information between all pairs of variables within the block. A key issue here is finding an appropriate block size and at the same time ensure that the blocks, in combination, guarantee that all pairs of variables are considered exactly once. This is the idea we propose.

To get an intuitive understanding of this process we can as an analogy consider the organization of the Speedway World Championship (SWC). After the initial pre-qualifying rounds for the SWC, the remaining 16 highest ranked riders should be compared to each other to obtain a final ranking of the riders. One approach to achieve this would be to pair-up the riders so that each rider will participate in 15 races, yielding a total of 120 rounds with two riders competing in each round. This setup would put a strain on the riders and not use the full capacity of the speedway track, which is designed to accommodate four riders simultaneously. Instead, the SWC employs a heat-system ensuring that each of the 16 riders will meet each of the other riders at some time during the competition. Specifically, the heat-system consists of 20 heats with four riders in a heat. Each rider participates in only five heats, and within a single heat all riders compete jointly, thereby meeting each other. After completing the 20 heats, all pairs of riders will have met exactly once. This can also be seen by labeling the riders $\{0, \ldots, 15\}$ and constructing these heats: $H_1 = \{3, 6, 12, 15\}$, $H_2 = \{4, 5, 10, 13\}$, $H_3 = \{0, 4, 6, 7\}$, $H_4 = \{0, 10, 11, 15\}$, $H_5 = \{7, 10, 12, 14\}$, $H_6 = \{0, 8, 9, 14\}$, $H_7 = \{0, 1, 3, 13\}$, $H_8 = \{1, 6, 8, 10\}$, $H_9 = \{7, 9, 13, 15\}$, $H_{10} = \{1, 5, 14, 15\}$, $H_{11} = \{8, 11, 12, 13\}$, $H_{12} = \{5, 6, 9, 11\}$, $H_{13} = \{1, 4, 9, 12\}$, $H_{14} = \{3, 5, 7, 8\}$, $H_{15} = \{3, 4, 11, 14\}$, $H_{16} = \{2, 6, 13, 14\}$, $H_{17} = \{1, 2, 7, 11\}$, $H_{18} = \{0, 2, 5, 12\}$, $H_{19} = \{2, 4, 8, 15\}$, and $H_{20} = \{2, 3, 9, 10\}$.

Returning to the (conditional) mutual information calculations, the 16 riders correspond to variables and each heat represents a block consisting of four variables to be pairwise compared. Thus, rather than handling pairs of variables independently and having to make data access $\binom{16}{2} = 120$ times, we can instead make 20 blocks/heats of four variables each and thereby only having to access the data 20 times. Note that with the particular setup above, we are guaranteed not to make redundant calculations as the scores $I(X_i, X_j|C)$ and $I(X_i, X_j)$ are computed exactly once for all $1 \leq i, j \leq n$.

This approach of distributing features/riders into blocks/heats is an instance of a so-called *balanced incomplete block (BIB) design*; in fact the heat-system configuration employed by the Speedway World Championship correspond to a $(16, 4, 1)$-BIB design (see Definition 2). BIB designs have already been used in related contexts like edge labeling of the spanning tree associated to TAN classifiers [20] and organizing the conditional independence tests in the PC algorithm for learning the structure of a Bayesian network [21]. In the following sections, we will give a more detailed specification of the approach and demonstrate how it can be used to ensure the scalability of Algorithm 1.

As a final note on computational efficiency, we would like to highlight that the filter approach in Algorithm 1 is especially appropriate for Bayesian classifiers and, in particular, for the Naive Bayes classifier, where the probability distribution of the class variable given the features is modeled as

$$p(c|x_1, \ldots, x_N) \propto p(c) \prod_{i=1}^{N} p(x_i|c).$$

For this model we see that all the required distributions have already been estimated when computing the conditional mutual information in Step 6. It means that some of the calculations carried out during the feature selection phase can be re-used when learning the model parameters. The same result also applies to TAN classifiers [12]. Furthermore, from the perspective of scalability, it is also worth pointing out that the information-theoretic score functions employed above can be efficiently updated after the arrival of new data if the distributions involved belong to the exponential family. It is enough to store a set of sufficient statistics corresponding to the parameters of the model, which are just counts for discrete variables.

2.2 Balanced incomplete block designs

The use of block designs dates back to the statistical theory of design of experiments [11], motivated in its origin by agricultural experiments. In this scenario 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 compensate for the effect of the environment by setting up blocks of land small enough to assume uniform environmental conditions inside a block, and distribute the plant varieties among them. With space limitations inside each block, one may not be able to fit sufficient replications of all plant varieties inside a single block, and therefore rather required that each pair of plant varieties would be allocated at least once to the same block to facilitate a fair comparison between them. The relation to both the SWC and our calculation of (conditional) mutual information is evident.

BIB designs [26] can be applied to efficiently divide the computation of the (conditional) mutual information between all the pairs of features among a set of, for instance, threads on a shared memory system or processes on a distributed memory system. This section provides the necessary background information on BIB designs to follow the presentation of the method proposed. For ease of presentation, we focus on a shared memory system using threads to achieve parallelization.

Definition 1 (Design [26]) *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 non-empty subsets of X called blocks.*

In this paper, we only exploit cases where each block is a set, and not a multi-set (i.e., we do not allow multiple instances of the same element in the set). Nevertheless, some definitions will consider multi-sets. A BIB design is defined as follows:

Definition 2 (BIB design [26]) *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.*

The number of blocks in a design is denoted by b. Property 3 in the definition is the *balance* property that we will exploit. In Steps 6

and 7 of Algorithm 1, we want to compute the (conditional) mutual information between each pair X_i, X_j exactly once and therefore require $\lambda = 1$. A BIB design is *symmetric* when the number of blocks equals the number of points. This will not be the case in general.

Example 1 *Consider the $(7, 3, 1)$-BIB design. The blocks are (one out of a number of possibilities):*

$$\{1, 2, 3\}, \{1, 4, 5\}, \{0, 1, 6\}, \{2, 4, 6\}, \{0, 2, 5\}, \{0, 3, 4\}, \{3, 5, 6\}.$$

This BIB design is symmetric as the number of blocks equals the number of points.

There is no single efficient method to construct all BIB designs. First, it is important to know that they do not exist for all combinations of v, k, and λ. Second, the problem of finding a BIB design is NP-complete [7]. To efficiently utilize them we have therefore precalculated a number of BIB designs, and utilized those at run-time. Instead of storing the full designs, it is sufficient to store *difference sets* that can be used to generate some symmetric BIB designs:

Definition 3 (Difference Set [26]) *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 multi-set $[x - y : x, y \in D, x \neq y]$ 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 multi-set of all v translates of D is denoted $\text{Dev}(D)$ and called the development of D [26, page 42].

Theorem 1 ([26], Theorem 3.8 p. 43) *Let D be a (v, k, λ)-difference set in an Abelian group $(G, +)$. Then $(G, \text{Dev}(D))$ is a symmetric (v, k, λ)-BIB design.*

Example 2 *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:*

$$\{0, 1, 3\}, \{1, 2, 4\}, \{2, 3, 5\}, \{3, 4, 6\}, \{4, 5, 0\}, \{5, 6, 1\}, \{6, 0, 2\}.$$

Notice that the ith element of each block is unique across all blocks. This property will be used to assign blocks to threads in Section 2.3. This was not the case for the blocks presented in Example 1.

The concept of a difference set can be generalized to the concept of a *difference family*. A difference family is a set of base blocks. A difference family can be used to generate a BIB design similarly to how difference sets are used. Table 1 shows a set of difference families for BIB designs on the form $(q, 6, 1)$, which we will use later. Base blocks for generating BIB-designs are tabulated, e.g., [27], but can also be found computationally. The base blocks in Table 1 have been generated using `SageMath`[9]. The value $k = 6$ is chosen for practical reasons: First, difference families for generating the blocks need to be known to exist; second, we need to be able to store the count tables representing the joint distribution of the class and all the variables in a block in memory. The main idea for parallelization considered in this paper is to use the $(q, 6, 1)$ design to distribute the computations of the scores over a set of computing units such that each score is computed exactly once from a smaller intermediate table over six variables.

[9] www.sagemath.org

2.3 Computing the (conditional) mutual information

The computation of the mutual and conditional mutual information scores $I(X_i, X_j)$ and $I(X_i, X_j|C)$ for all pairs of features X_i, X_j should be divided into tasks of (approximately) equal size such that the score for each pair X_i, X_j is computed exactly once. This is achieved using BIB designs of the form $(q, 6, 1)$ where q is at least the number of features. That is, q is selected as the smallest value larger than the number of variables such that a $(q, 6, 1)$ BIB design is known to exist. The blocks of the BIB design are generated using a difference family (e.g., Table 1). Each block is used to compute the marginal counts of the features represented in the block (and the class variable). If all features have the same state space size, then the count tables will be of equal size.

The computation of the mutual and conditional mutual information scores is parallelized assigning blocks to the computing units available (we assume threads) as each thread can compute the scores corresponding to a block in parallel with other threads. Blocks are assigned to threads using the unique rank of each thread. A thread with rank r iterates over the block array and considers only blocks where the array index modulus t equals r where t is the number of threads (the uniqueness means that there is no need for synchronization). When a thread has selected a block, it computes the scores for all the pairs of features using a $(3, 2, 1)$-BIB design where the 6-block is marginalized to three blocks with four features each (in this case each point corresponds to two features). The tables of four variables are marginalized down to all pairs for computing the score where the first pair is ignored producing a total of $\binom{6}{2} = 15$ scores to be computed.

Figure 1 illustrates this principle, assuming an example with $q = 31$ features labelled as $X_0, \ldots, X_{30}$. The first block (second row in the figure) is $\{X_1, X_2, X_7, X_{19}, X_{23}, X_{30}\}$, corresponding to the difference family for design $(31, 6, 1)$, as given in Table 1. The second block would be obtained by adding 1 to the index of the variable in each coordinate, modulo 31, i.e., $\{X_2, X_3, X_8, X_{20}, X_{24}, X_0\}$. According to the same procedure, the third block would be $\{X_3, X_4, X_9, X_{21}, X_{25}, X_1\}$ and so on.

Taking the first block, we form three pairs of features, $P_1 = \{X_1, X_2\}$, $P_2 = \{X_7, X_{19}\}$ and $P_3 = \{X_{23}, X_{30}\}$ and compute the blocks of a $(3, 2, 1)$ design, where each block has two pairs. These blocks are actually all the possible pairings of P_1, P_2 and P_3, namely $\{P_1, P_2\}$, $\{P_2, P_3\}$ and $\{P_3, P_1\}$, placed on the third row of Figure 1. It can be seen that every three pairings we come up with $5 \times 3 = 15$ pairs of features for which the mutual and mutual information score is computed. In fact, each block corresponding to a pairing $\{P_i, P_j\}$ yields 6 pairs of features, but the first one is discarded in order to avoid repetitions. In Figure 1 it is indicated by marking both variables in red on the lower row.

Notice that $k = 6$ represents 15 pairs and the number of times we count is reduced by a factor of 15, but each count is a factor three more expensive (as we are counting six variables instead of two variables). In addition, there is the task of marginalizing the count tables to pairs. If the number of states for some features is high, then it may be more efficient to compute the score directly from the dataset instead of creating an intermediate table. More precisely, the table can become larger than the original dataset meaning that we would be computing counts from a table that is larger than the original dataset. In this case we may not obtain a time performance improvement.

Table 1. Examples of difference families for a set of $(q, 6, 1)$-BIB designs.

BIB design	Difference family	#(base blocks)	b = $v \cdot$ #(base blocks)
(31,6,1)	$\{(1, 2, 7, 19, 23, 30)\}$	1	31
(91,6,1)	$\{(0, 1, 3, 7, 25, 38),$ $(0, 5, 20, 32, 46, 75),$ $(0, 8, 17, 47, 57, 80)\}$	3	273
(151,6,1)	$\{(1, 32, 118, 7, 73, 71), \ldots\}$	5	755
(211,6,1)	$\{(0, 1, 107, 55, 188, 71), \ldots\}$	7	1477
(271,6,1)	$\{(1, 242, 28, 9, 10, 232), \ldots\}$	9	2439

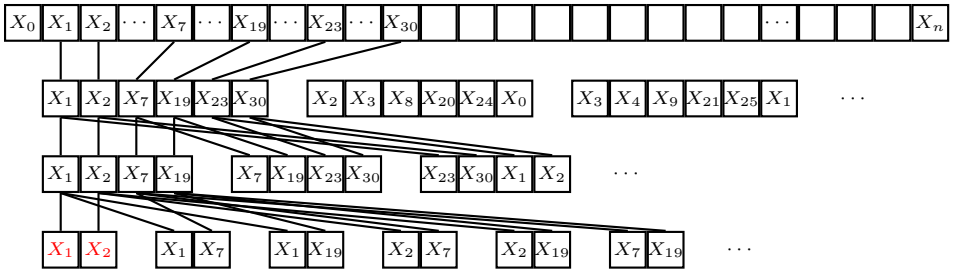

Figure 1. Example illustrating the use of $(q, 6, 1)$ and $(3, 2, 1)$ designs.

3 RESULTS

This section reports on the results of an empirical evaluation of the proposed parallel filter-based feature selection method based on BIB designs. The main focus of the empirical evaluation is to investigate the time performance improvement offered by the use of BIB designs to compute the score used by Algorithm 1. We consider the implementation of the proposed method on a shared memory computer with multiple cores such that threads can be used to achieve parallelization. This means that the entire dataset is loaded into the main shared memory of the computer where the process of the program is responsible for creating a set of POSIX threads to achieve parallelization.

Table 2. Networks from which datasets used in the experiments are generated.

Dataset	$\lvert\mathcal{X}\rvert$	$\lvert E\rvert$	Total CPT size
Munin1 [3]	189	282	19,466
Diabetes [2]	413	602	461,069
Munin2 [3]	1,003	1,244	83,920
SACSO [13]	2,371	3,521	44,274

The evaluation is performed using a total of fifteen datasets where twelve datasets are generated by sampling from known Bayesian network models [23, 9, 14, 16] and three are subsets of a real-world dataset over 1,823 variables from a Spanish bank. Random samples of data were generated from the four Bayesian networks of different sizes listed in Table 2 where $|\mathcal{X}|$ denotes the number of variables and $|E|$ denotes the number of edges in the Bayesian network. For each network a single variable is selected as target variable. Three datasets were generated at random for each network with 100,000, 250,000, and 500,000 cases. In addition, three samples from the bank dataset have been used. All datasets used are unless otherwise stated complete, i.e., there are no missing values in the data.

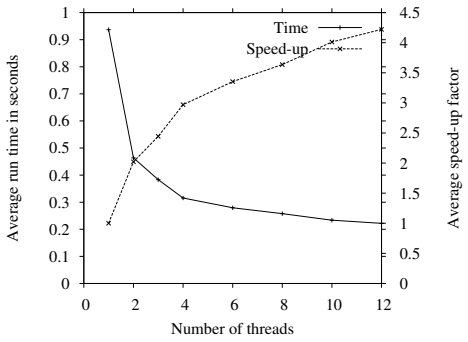

Figure 2. Munin1 with 100,000 cases.

The empirical evaluation is performed on a desktop computer running Red Hat Enterprise Linux 7 with a six-core Intel (TM) i7-5820K 3.3GHz processor and 64 GB RAM. The computer has six physical cores and twelve logical cores. For this reason, the number of threads used by the program is in the set $\{1, 2, 3, 4, 6, 8, 10, 12\}$ where the case of one thread is considered the baseline and corresponds to a sequential program.

The average computation time is calculated over ten runs with the same dataset. The computation time is measured as the elapsed (wall-clock) time of the algorithm. This means that other users of the computer may potentially impact the results. The speed-up factor is computed relative to the case of one thread.

Figures 2, 3, and 4 show the time performance and resulting speed-up factor achieved on the Munin1 dataset with 100,000, 250,000 and 500,000 cases, respectively.

The three performance figures for the Munin1 datasets clearly show an improvement in time performance as the number of threads

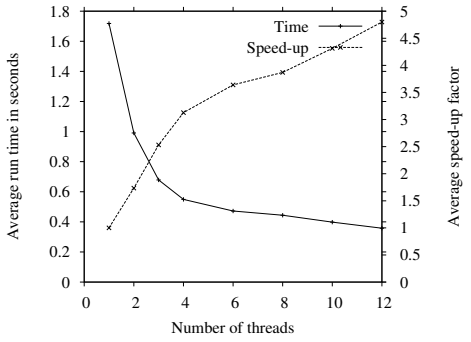

Figure 3. Munin1 with 250,000 cases.

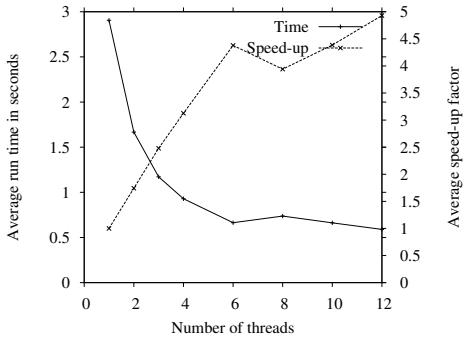

Figure 4. Munin1 with 500,000 cases.

used increases. The highest speed-up factor of approximately five is obtained using twelve threads and 500,000 cases. The most significant speed-up is obtained when going from one to two threads where the speed-up factor is close to two. After six threads there is in Figure 4 a small drop in performance. This is most likely related to the number of physical cores in the computer. Recall that the computer used in the test has six physical cores and twelve logical cores. We believe this is the explanation for the drop in performance going from six to eight threads.

Due to space restrictions, we do not show the time performance graphs for all tests. Instead we show the graphs for the largest datasets.

Figures 5, 6, and 7 show the time performance and resulting speed-up factor achieved on the Munin2 dataset with 500,000 cases, Bank with 500,000 cases, and SACSO with 500,000 cases, respectively.

The three performance graphs for Munin2, SACSO and Bank shown in Figures 5, 7, and 6, respectively show similar time performance improvements as Munin1. A highest speed-up factor of approximately six is obtained for all three networks.

Figure 8 shows the time performance and resulting speed-up factor achieved on the Diabetes dataset with 100,000 cases.

In this figure, we notice that the performance starts to deteriorate after six threads. Diabetes is a time-sliced model with a number of variables in each time slice having up to 21 states. This means that the use of $(q, 6, 1)$ designs could produce intermediate tables that are as large as or even are larger than the original dataset and the work could be unevenly distributed as a result of different table sizes.

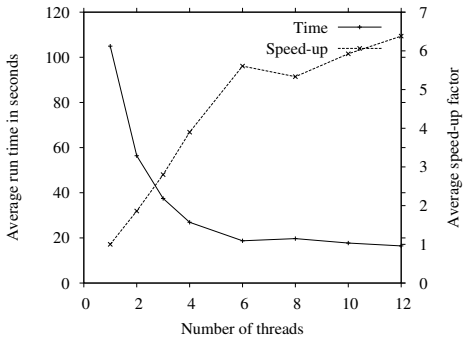

Figure 5. Munin2 with 500,000 cases.

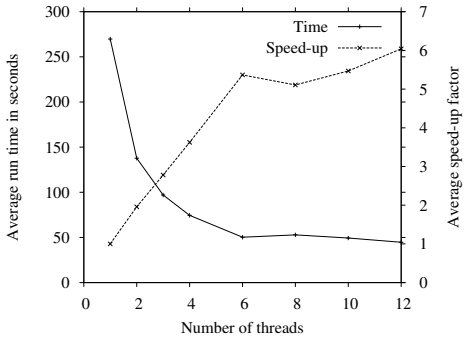

Figure 6. Bank with 500,000 cases.

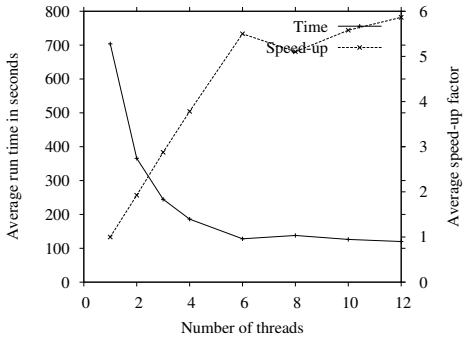

Figure 7. SACSO with 500,000 cases.

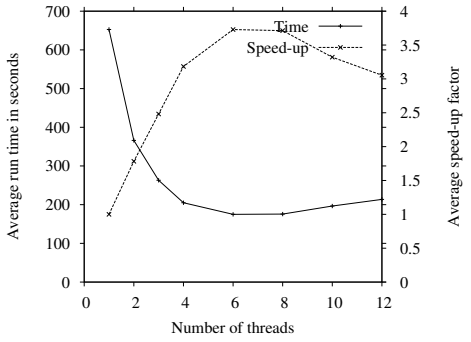

Figure 8. Diabetes with 100,000 cases.

Figure 8 shows the time performance and resulting speed-up factor achieved on the Diabetes dataset with one incomplete and 100,000 complete cases.

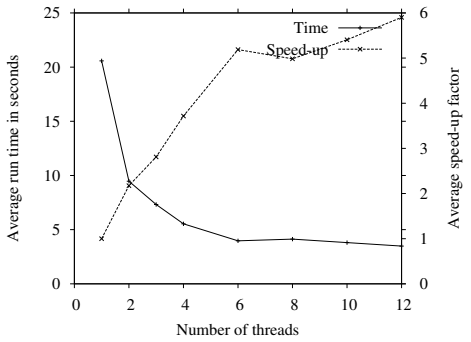

Figure 9. Diabetes with 100,000 complete and one incomplete cases.

Adding an incomplete case to the dataset means that $(q, 6, 1)$ designs are not used and the scores are computed directly from the original dataset. In this case, time performance is significantly better than in the case of complete data and the speed-up factor improves as the number of threads increases. Figure 10 shows the performance graphs for the Bank dataset with 100,000 complete cases while Figure 11 shows the performance for 100,000 complete cases and one incomplete case. It is clear from the two figures that the use of $(q, 6, 1)$ designs improves time performance significantly.

4 DISCUSSION

The results of the empirical evaluation reported in Section 3 demonstrates the scalability of the proposed approach to parallel filter-based feature selection using BIB designs. On the test computer the time performance clearly improves as the number of threads increases up to the number of physical cores. Thereafter, there is in some cases a minor drop in performance.

The proposed method uses BIB designs in two steps in order to ensure that the score is computed for each pair exactly once and such that the workload can be evenly distributed over the available computing units without any synchronization. In the first step a $(q, 6, 1)$ design is applied to create intermediate tables that are marginalised

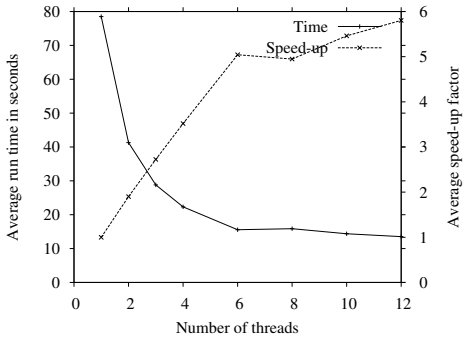

Figure 10. Bank with 100,000 complete cases.

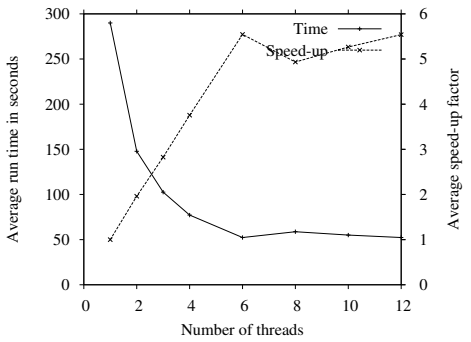

Figure 11. Bank with 100,000 complete and one incomplete cases.

to two or three variables to compute the score using a $(3, 2, 1)$ design. That is, the $(q, 6, 1)$ design is used to create intermediate tables from which the scores for a specific set of pairs are calculated. In some cases, when variables have a high number of states as in the Diabetes network, the size of the intermediate tables may become too large to manage efficiently and performance can deteriorate. We can avoid this step by determining all pairs directly from the entire set of features. This would mean that the scores for each pair are computed over the entire dataset. A upper limit equal to the size of the original dataset could be put on the size of the intermediate tables.

If data has missing values, then we cannot exploit the $(q, 6, 1)$ designs as an intermediate table for the variables with missing values and would compute the counts for each pair directly from the original dataset ignoring cases with missing values. The performance improvement offered by the intermediate tables creating using $(q, 6, 1)$ designs is in most cases substantial.

Future work includes horizontal parallelization of the algorithm where each computing unit holds a subset of the data over all variables and is responsible for computing partial counts over the data it holds. This will support parallelization on a distributed memory system.

5 CONCLUSION

In this paper, we have proposed a method for scaling up filter-based feature selection in classification problems using parallelization. The principal idea is to use the conditional mutual information as a filter measure distributing the computation of the required statistics over a set of computing units. The distribution of the computations is managed using BIB designs requiring no synchronization and ensuring that each score is computed exactly once. We have demonstrated the scalability of the proposed method using both synthetic and real-world datasets.

ACKNOWLEDGEMENTS

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. We thank BCC for providing access to an obfuscated sample dataset.

REFERENCES

[1] S. Ahmed, H. Narasimhan, and S. Agarwal, 'Bayes optimal feature selection for supervised learning with general performance measures', in *Proceedings of the UAI'2015 Conference*, (2015).

[2] S. Andreassen, R. Hovorka, J. Benn, K.G. Olesen, and E.R. Carson, 'A model-based approach to insulin adjustment', in *Proc. of the Third Conference on Artificial Intelligence in Medicine*, pp. 239–248, (1991).

[3] 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 *Computer-Aided Electromyography and Expert Systems*, chapter 21, Elsevier Science, (1989).

[4] P. Bermejo, L. de la Ossa, J.A. Gámez, and J.M. Puerta, 'Fast wrapper feature subset selection in high-dimensional datasets by means of filter re-ranking', *Knowledge-Based Systems*, **25**, 35–44, (2012).

[5] P. Bermejo, J.A. Gámez, and J.M. Puerta, 'A GRASP algorithm for fast hybrid (filter-wrapper) feature subset selection in high-dimensional datasets', *Pattern Recognition Letters*, **32**, 701–711, (2011).

[6] G. Brown, A. Pocock, M. Zhao, and M. Luján, 'Conditional likelihood maximisation: A unifying framework for information theoretic feature selection', *Journal of Machine Learning Research*, **13**, 27–66, (January 2012).

[7] D.G. Corneil and R.A. Mathon, 'Algorithmic techniques for the generation and analysis of strongly regular graphs and other combinatorial configurations', *Annals of Discrete Mathematics*, **2**, 1–32, (1978).

[8] T.M. Cover and J.A. Thomas, *Elements of Information Theory (Wiley Series in Telecommunications and Signal Processing)*, Wiley-Interscience, 2nd edn., 2006.

[9] R.G. Cowell, A.P. Dawid, S.L. Lauritzen, and D.J. Spiegelhalter, *Probabilistic Networks and Expert Systems*, Springer, 1999.

[10] W. Duch, 'Filter methods', in *Feature extraction. Fundations and applications*, eds., I. Guyon, S. Gunn, M. Nikravesh, and L.A. Zadeh, volume 207 of *Studies in Fuzziness and Soft Computing*, 89–117, Springer, (2006).

[11] R.A. Fisher, 'An examination of the different possible solutions of a problem in incomplete blocks', *Annals of Eugenics*, **10**, 52–75, (1940).

[12] Nir Friedman, Dan Geiger, and Moises Goldszmidt, 'Bayesian network classifiers', *Machine Learning*, **29**(2–3), 131–163, (1997).

[13] F. V. Jensen, C. Skaanning, and U. Kjærulff, 'The SACSO System for Troubleshooting of Printing Systems', in *In Proceedings of the Seventh Scandinavian Conference on Artificial Intelligence*, (2001).

[14] F.V. Jensen and T.D. Nielsen, *Bayesian Networks and Decision Graphs*, Springer, 2nd edn., 2007.

[15] G.H. John, R. Kohavi, and P. Pfleger, 'Irrelevant features and the subset selection problem', in *Machine Learning: Proceedings of the Eleventh International Conference*, pp. 121–129. Morgan Kaufmann, (1994).

[16] U.B. Kjærulff and A.L. Madsen, *Bayesian Networks and Influence Diagrams: A Guide to Construction and Analysis*, Springer, 2nd edn., 2013.

[17] Vipin Kumar and Sonajharia Minz, 'Feature selection: A literature review', *Smart Computing Review*, **4**(3), 211–229, (2014).

[18] T.N. Lan, O. Chapelle, J. Weston, and A. Elisseeff, 'Embedded methods', in *Feature extraction. Fundations and applications*, eds., I. Guyon, S. Gunn, M. Nikravesh, and L.A. Zadeh, volume 207 of *Studies in Fuzziness and Soft Computing*, 137–165, Springer, (2006).

[19] J. Li, K. Cheng, S. Wang, F. Morstatter, R.P. Trevino, J. Tang, and H. Liu, 'Feature selection: A data perspective', *arXiv preprint arXiv:1601.07996*, (2016).

[20] A.L. Madsen, F. Jensen, A. Salmerón, M. Karlsen, H. Langseth, and T. D. Nielsen, 'A new method for vertical parallelisation of TAN learning based on balanced incomplete block designs', in *PGM'2014. Lecture Notes in Artificial Intelligence*, volume 8754, pp. 302–317, (2014).

[21] A.L. Madsen, F. Jensen, A. Salmerón, H. Langseth, and T.D. Nielsen, 'Parallelization of the PC algorithm', *CAEPIA'2015. Lecture Notes in Artificial Intelligence*, **9422**, 14–24, (2015).

[22] L. Morán-Fernández, V. Bolón-Canedo, and A. Alonso-Betanzos, 'A time efficient approach for distributed feature selection partitioning by features', *CAEPIA 2015. Lecture Notes in Artificial Intelligence*, **9422**, 245–254, (2015).

[23] J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*, Series in Representation and Reasoning, Morgan Kaufmann, 1988.

[24] H. Peng, F. Long, and C. Ding, 'Feature selection based on mutual information: criteria of max-dependency, max-relevance, and min-redundancy', *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **27**, 1226–1238, (2005).

[25] C.E. Shannon, 'A mathematical theory of communication', *Bell Systems Technical Journal*, **27**(3), 379–423, (1948).

[26] D. Stinson, *Combinatorial Designs — Constructions and Analysis*, Springer, 2003.

[27] K. Takeuchi, 'A table of difference sets generating balanced incomplete block designs', *Review of the International Statistical Institute*, **30**(3), 361–366, (1962).

[28] H. Yang and J. Moody, 'Data visualization and feature selection: New algorithms for non-Gaussian data', in *Advances in Neural Information Processing Systems*, volume 12, (1999).