Computationally Efficient Induction of Classification Rules with the PMCRI and J-PMCRI Frameworks

Frederic Stahl and Max Bramer

Frederic Stahl, Bournemouth University, School of Design, Engineering & Computing, Poole House, Talbot Campus, BH12 5BB Poole, fstahl@bournemouth.ac.uk

Frederic Stahl, Max Bramer University of Portsmouth, School of Computing, Buckingham Building, Lion Terrace, PO1 3HE Portsmouth, Max.Bramer@port.ac.uk

Abstract

In order to gain knowledge from large databases, scalable data mining technologies are needed. Data are captured on a large scale and thus databases are increasing at a fast pace. This leads to the utilisation of parallel computing technologies in order to cope with large amounts of data. In the area of classification rule induction, parallelisation of classification rules has focused on the *divide and conquer* approach, also known as the Top Down Induction of Decision Trees (TDIDT). An alternative approach to classification rule induction is *separate and conquer* which has only recently been in the focus of parallelisation. This work introduces and evaluates empirically a framework for the parallel induction of classification rules, generated by members of the Prism family of algorithms. All members of the Prism family of algorithms follow the *separate and conquer* approach.

Keywords:
Parallel Computing, Parallel Rule Induction, Modular Classification Rule Induction, PMCRI, J-PMCRI, Prism

1. Introduction

Many application areas are confronted with the problem of applying classification rule induction algorithms or data mining technologies in general on very large datasets. Such application areas include bioinformatics and chemistry which are confronted with large data sets, for example data generated in molecular dynamics simulation experiments. Researchers in this area

need ways to manage, store and find complex relationships in the simulation data [1]. The molecular dynamics datasets can reach 100s of gigabytes of data for a single simulation and the community is just starting to be able to store these massive amounts of simulation data [2]. A further area confronted with massive amounts of data is astronomy. Here some databases consist of terabytes of image data and are still growing as further data are collected in relation to the GSC-II [3] and the still ongoing Sloan survey [4]. Large international business corporations collect and share customer transactions in databases worldwide. Loosely speaking there is a significant need for well scaling knowledge discovery and data mining technologies for massive datasets for both the scientific and business world. Parallelisation seems to be one of the methods used in the data mining community to tackle the problem of scalability in computational terms [33, 10, 21].

One of the major challenges in data mining is the induction of classification rules on massive datasets. There are two general approaches to inducing classification rules, the *divide and conquer* and the *separate and conquer* approaches. The induction of classification rules can be traced back to the 1960s [5]. The *divide and conquer* approach induces classification rules in the form of a decision tree by recursively splitting the classification problem [6]. Its most popular representatives are the C4.5 [7] and C5.0 systems. Contrary to decision trees the *separate and conquer* approach induces classification rules directly that explain a part of the training data. *Separate and conquer* can be traced back to the 1960s [8]. Parallel classification rule induction has focused on the *divide and conquer* approach. A notable development here is SPRINT [9]. [10] points out that in some cases SPRINT may suffer from workload balancing issues and the ScalParC algorithm is proposed. However there are virtually no approaches to scaling up the *separate and conquer* approach.

The Prism [11] family of algorithms follows the *separate and conquer* approach and addresses some of the shortcomings of decision trees, such as the replicated subtree problem outlined in Section 2.1. More recent variations of Prism have demonstrated a similar classification accuracy compared with decision trees and in some cases even outperform decision trees [17, 15]. An implementation of Prism is also available in the WEKA data mining package[35]. This work proposes and evaluates the Parallel Modular Classification Rule Induction framework (PMCRI) which parallelises the Prism family of algorithms, in order to computationally scale up Prism algorithms to large datasets. PMCRI could potentially scale up further algorithms, that follow the *separate and conquer* approach, to large datasets, however, it may

not be applicable to all of them.

In the PMCRI framework the parallelisation is aimed at a network of computer workstations with the reasoning that modest sized organisations may not have the financial strength to afford a supercomputer but will most likely have a network of workstations in place which could be used to run parallel algorithms. PMCRI partitions the training data according to the features space and assigns equally sized subsets of the feature space to each computing node. Each computing node processes its part of the feature space and then cooperates with the other computing nodes in order to combine the computed results to classification rules. The computational performance of PMCRI is evaluated in terms of its execution time dependent on the number or data instances and the number of attributes/features. Furthermore PMCRI is evaluated to show how much computational benefit is gained by using p processors instead of one, dependent on the size of the datasets.

This paper is organised as follows: Section 2 introduces the Prism family of algorithms and compares them with decision trees; Section 3 discusses the PMCRI framework and Section 4 evaluates it. Section 5 introduces a version of PMCRI that incorporates a pre-pruning facility (J-PMCRI) and evaluates it in computational terms. Finally Section 6 closes the paper with a brief summary, concluding remarks and an outlook to future work.

2. The Prism Family of Algorithms

The Prism family of algorithms is a representative of the 'separate and conquer' approach outlined in Section 1 as opposed to the 'divide and conquer' approach.

2.1. The Replicated Subtree Problem

The 'divide and conquer' approach induces classification rules in the intermediate form of a tree whereas the 'separate and conquer' approach, and thus the Prism family of algorithms, induces modular rules that do not necessarily fit into a decision tree. Modular rules such as

$$IF\ A = 1\ AND\ B = 1\ THEN\ class = x$$

$$IF\ C = 1\ AND\ D = 1\ THEN\ class = x$$

will not fit into a decision tree as they have no attribute in common. In order to represent them in a decision tree, additional logically redundant

rule terms would have to be added. This can result in complex and confusing trees as Cendrowska shows in [11] and is also known as the replicated subtree problem [12]. Cendrowska's example illustrates the replicated subtree problem for the two rules listed above. She assumes that all attributes can have 3 possible values and only the two rules above classify for class x. Figure 1 shows the simplest possible decision tree expressing the two rules above, all remaining classes that are not class x are labeled y.

Cendrowska's claim that decision tree induction algorithms grow needlessly complex is vindicated by extracting the rules for class x from the tree in Figure 1, which are:

IF A = 1 AND B = 1 THEN Class = x

IF A = 1 AND B = 2 AND C = 1 AND D = 1 THEN Class = x

IF A = 1 AND B = 3 AND C = 1 AND D = 1 THEN Class = x

IF A = 2 AND C = 1 AND D = 1 THEN Class = x

IF A = 3 AND C = 1 AND D = 1 THEN Class = x

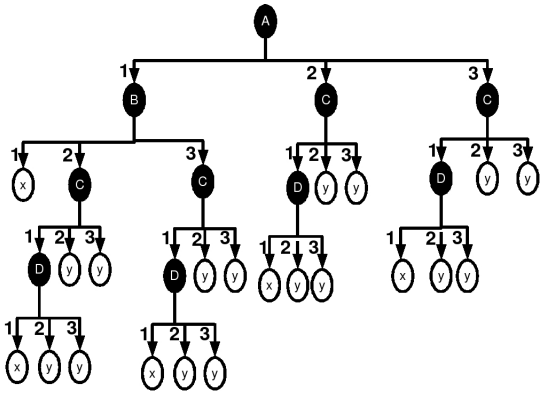

Figure 1: Cendrowska's replicated subtree example.

2.2. The 'Separate and Conquer' Approach

Algorithms induced by the 'separate and conquer' approach aim to avoid the replicated subtree problem by avoiding the induction of redundant rule terms just for the representation in a decision tree. The basic 'separate and conquer' approach can be described as follows:

```
While Stopping Criterion not satisfied{
```

```
    rule = Learn_Rule;
    Remove all data instances covered from Rule;
    add rule to the rule set;
}
```

The *Learn_Rule* procedure (or specialisation process) induces the 'best' rule for the current subset of the training set by searching for the best conjunction of attribute-value pairs (rule terms). The perception of 'best' depends on the heuristic used to measure the goodness of the rule, for example its coverage or predictive accuracy. The *Learn_Rule* procedure can be computationally very expensive, especially if all possible conjunctions of all possible rule terms have to be considered. After a rule is induced, all examples that are covered by that rule are deleted and the next rule is induced using the remaining examples until a *Stopping Citerion* is fulfilled. Different 'separate and conquer' algorithms implement different methods to reduce the search space of the *Learn_Rule* procedure and the *Stopping Criterion*. Some examples of algorithms that follow the 'separate and conquer' approach are [11, 36, 8, 37].

2.3. The Prism Approach

In the Prism approach, first a Target Class (TC), for which a rule is induced, is selected. Prism then uses an information theoretic approach [11] based on the probability with which a rule covers the TC in the current subset of the training data to specialise the rule. Once a rule term is added to the rule, a further rule term is induced only on the subset of the training data, that is covered by all the rule terms induced so far. This is done until the rule currently being induced only covers instances that match the TC. In Cendrowska's original Prism algorithm the TC is selected at the beginning by the user and only rules for the TC are induced. Alternatively the algorithm can be given a list of possible classes and is executed for each class in turn. Bramer's PrismTCS (Target Class Smallest first) algorithm [13] sets the TC for each new rule to be induced to the current minority class. Another member of the Prism family is PrismTC which sets the TC for each new rule to be induced to the current majority class. Unpublished experiments by Bramer revealed that PrismTC does not compete well with Cendrowska's original Prism and Bramer's PrismTCS. The advantage of PrismTCS is that it needs fewer iterations than the original Prism while maintaining a similar level of predictive accuracy and thus has a better computational performance [14]. The stopping criterion of Prism is to stop when there are either no more

examples left or the remaining examples all belong to the same class. The basic PrismTCS algorithm is outlined below where A_x is a possible attribute-value pair and D is the training dataset and i the minority class.

```
Step 1: D' = D;
Step 2: Calculate for each Ax in D' p(class = i| Ax);
Step 3: Select the Ax with the maximum p(class = i| Ax)
           and delete all instances from D' that do
           not match Ax;
Step 4: Repeat 2 to 3 until D' only contains
           instances of classification i.
           The induced rule then has as its antecedent
           the conjunction of all the selected Ax,
           with consequent class=i;
Step 5: Delete all instances from D that cover the
           induced rule;
Step 6: IF(D does not contain any instances of class i){
                  IF(D does not contain i){
                     recalculate minority class i;
                   }
                   GO TO Step 1;
        }
```

All methods outlined in this paper can be applied to any member of the Prism family, however we will focus here on the more popular and computationally more efficient PrismTCS algorithm.

2.3.1. Dealing with Clashes in Prism

According to [15] the best way to deal with clashes, which is also implemented in the Inducer data mining software [15, 13], is to check if the clashing instances have the TC as the majority class. If so, then the rule is taken into the rule set, otherwise the rule is discarded and the instances in the clash set that match the TC are deleted.

2.4. Dealing with Continuous Attributes in Prism

One way to deal with continuous attributes is discretisation of attribute values using techniques such as ChiMerge [16] before Prism is applied. In step 2 in the pseudocode above, Inducer calculates for each attribute value v (if attribute A is continuous) two tests. The two tests are the probabilities $p(class = i \mid A_x \geq v)$ and $p(class = i \mid A_x < v)$. The test with the highest probability for attribute A is selected and compared with those from the other attributes.

3. PMCRI: A Parallel Modular Classification Rule Induction Framework

This section reviews the Parallel Modular Classification Rule Induction (PMCRI) framework for inducing classification rules in parallel using the Prism family of algorithms. Any algorithm of the Prism family can be parallelised using PMCRI. Using PMCRI will produce exactly the same rules as the serial version of the parallelised Prism algorithm would, only PMCRI is able to cope computationally with much larger data volumes. A version of PMCRI that incorporates rule pruning, J-PMCRI will be introduced in Section 5.

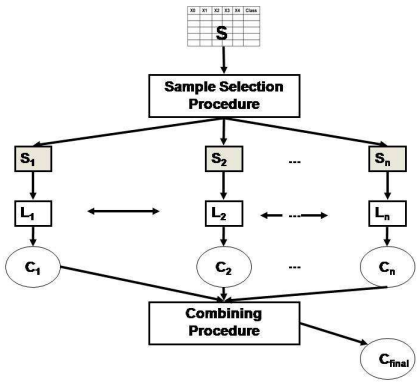

Figure 2: Cooperating Data Mining Model.

The scope of the research described in this paper is to derive a methodology that helps modest sized organisations to harvest the computational power of their standalone workstations interconnected in a network in order to avoid having to purchase expensive parallel computers. As Prism's computational complexity is directly dependent on the amount of data [19], data parallelisation is used for the PRISM framework. Data parallelisation can be achieved by partitioning the data into subsets and distributing these subsets evenly over n machines in a network of n separate computers.

The PMCRI framework is broadly based on the Cooperating Data Mining Model (CDM) [20] illustrated in Figure 2. CDM describes in general an approach for distributed and parallel data mining, it can be divided into three basic steps; a *sample selection procedure*, *learning local concepts* and

combining local concepts using a *combining procedure* into a final concept description.

- *sample selection procedure*: In the sample selection procedure samples S_1,..., S_n of the training data are taken and distributed over n machines. How the training data are partitioned is not further specified. For example, the samples may contain a subset of the instances or a subset of the attributes.

- *learning local concepts*: On each of the n machines there is a learning algorithm L running that learns a local concept out of the data samples locally stored on each machine. In general each L has a local view of the search space reflected by the data it holds in the memory. A global view of the search space can be obtained by communication between algorithms L_1,...,L_n. This can be done by exchanging parts of the training data or information about the training data. Where the latter is preferred, as information about data is usually smaller in size than the data itself and thus consumes less bandwidth. Subsequently each L will derive a concept description C from the locally stored data and the information that has been exchanged with other *Ls*.

- *combining procedure*: Eventually all *Cs* derived from all local *Ls* are combined into a final concept description C_f. How this is done depends on the underlying learning algorithm and its implementation.

3.1. The Sample Selection Procedure

The sample selection procedure is crucial for achieving optimal workload balancing. For PMCRI a similar approach to that in the SPRINT [9] algorithm has been developed. The training data are partitioned by building attribute lists for each attribute of the structure *<record id, attribute value, class value>*.

The left-hand side of Figure 3 shows how attribute lists are built in PMCRI. All lists are sorted and can be kept sorted during the whole duration of the algorithm's execution as identifiers (ids) are added to each list entry that allow the reconstruction of whole data records. When calculating the highest value of $p(class = i \mid A_x \geq v)$ or $p(class = i \mid A_x < v)$ for each continuous attribute A at step 2 in the algorithm given in Section 2 it is essential to use attribute values that have been sorted into numerical order. As pointed out in [21], the use of attribute lists and pre-sorting in the serial

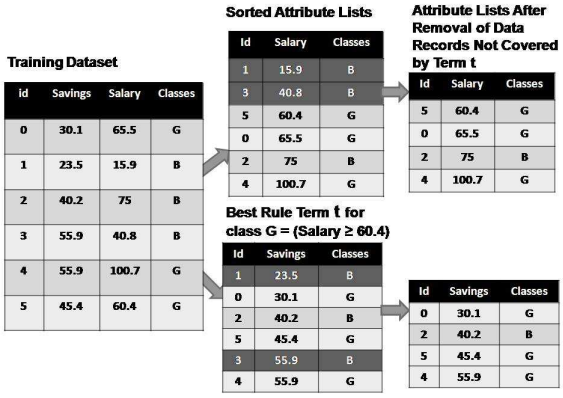

Figure 3: The left hand side shows how sorted attribute lists are built and the right hand side shows how list records, in this case records with the ids 1 and 3, are removed in Prism.

versions of Prism algorithms gives a substantial improvement in runtimes over earlier implementations in which each continuous attribute needed to be sorted afresh as each new rule term was induced. As Prism removes data instances that are not covered by previously induced rule terms (see step 3 in the PrismTCS algorithm in Section 2), the attribute list entries will have to be removed accordingly. For example in the data used in Figure 3, if Prism finds a rule term ($salary \geq 60.4$) for class G then Prism would remove the list entries matching ids 1 and 3 from all attribute lists as they are not covered by ($salary \geq 60.4$) in the attribute list 'salary'.

What is important to note is that after removing the list entries for uncovered instances, each attribute list has the same number of list entries and is still sorted. Thus an equal workload balance during the whole duration of Prism's execution in PMCRI can be achieved by distributing the attribute lists evenly over n computers. This is not the case for the SPRINT algorithm as SPRINT partitions each sorted attribute list into n equal parts and thus distributes each single attribute list evenly over n computers or processors. Compared with PMCRI, SPRINT may have a perfect workload balance at the beginning but the workload is likely to become imbalanced during SPRINT's execution [10]. In the PMCRI lists distribution strategy, a slightly imbalanced workload at the beginning may occur if $\frac{numberofattributes}{numberofComputers}$ is not an integer, however it is accepted as the workload balance will stay constant during the whole duration of Prism's execution.

3.2. Parallelising a System of Rule Term Learners Using a Distributed Blackboard Approach

The training data are distributed over p processors in the form of complete attribute lists. When this paper mentions processors in the context of PMCRI or J-PMCRI, then in fact workstations that have their own private memory are meant. The learner that each processor has to run for the Prism algorithm in order to derive rules is described below in five steps. However these steps can be easily adapted for any member of the Prism family.

Step 1: Each processor induces rule terms 'locally' on attribute lists it holds in memory by calculating all the conditional probabilities for the target class in all the attribute lists and taking the largest one. If there is a tie break, the rule term that covers the highest count of the target class is selected as the locally best rule term. The processors now need to communicate in order to find out which processor induced the globally best rule term using the probabilities and target class counts.

Step 2: After finding the best rule term, each processor needs to delete all attribute list records that are not covered by the globally best rule term. In order to do so all processors need to retrieve the ids from the processor that induced the globally best rule term.

Step 3: A rule term has been induced and Prism now needs to check whether the rule is completed or not. Each processor checks this independently. If the local attribute lists only contain the target class, then the rule is finished and the processor continues with step 4, otherwise the rule is incomplete and the processor jumps back to step 1.

Step 4: The current rule is completed and Prism needs to restore the data and delete all instances that are covered by the rules induced so far. For each processor the ids of the list records left after the final rule term has been induced are the ones that are covered by the completed rule. These ids are accumulated and remembered by the processor. The original attribute lists are restored and list records matching the accumulated ids are deleted.

Step 5: Each processor checks if there is still more than one list record left in each attribute list, and each of the attribute lists comprises records associated with more than just one class, then the next rule is induced, otherwise the stopping criterion of Prism is fulfilled and the processor will stop and exit.

Steps 1 to 5 show the basic operations a processor has to execute in order to derive Prism rules. Please note that the only communication takes

place in step 1 for communicating rule term properties and in step 2 for communicating the 'globally best' rule term covered ids. The learner that induced the globally best rule term only needs the rule term and the attribute lists from which it was derived in order to derive the ids, that match attribute list records, on all processors that are covered by the rule term. Thus it is not necessary to communicate the actual rule terms to all processors. The globally best rule term is retained in order to be able to extract the overall rule eventually and all other rule terms will be deleted. In general each processor can act independently over most of the tasks involved in inducing Prism rules.

Below is the pseudocode of a learning algorithm for a processor for steps 1 to 5 above. The parts that involve communication are numbered (1), (2) and (3). The pseudocode describes how rules for one target class are induced:

```
//Global Variables
TargetClass _tc;
List _rules;
Rule _rule;
List _fromRulesCoveredIds;
int[] _toDeleteIds;
while(while attribute lists are not empty and contain _tc){
 _rule = new Rule();
 while((local lists contain tc)AND
                          (_tc is not the only class)){
    induce locally best rule term t;
    exchange information about t with
     all other processors (1);
    if(t is globally best rule term){
       _rule.append(t);
       _toDeleteIds = Generate by t uncovered ids;
       communicate _toDeleteIds to
        remaining processors (2);
    } else{
       retrieve _toDeleteIds from winning processor (3);
    }
    delete all list instances matching _toDeleteIds;
    clear _toDeleteIds;
 }
 _fromRulesCoveredIds.add(ids present in
   current attribute lists);
 _rules.add(rule);
 restore all attribute lists to their initial size;
 delete all instances from all attribute lists that
 match ids in _fromRulesCoveredIds;
}
```

In the pseudocode above, the processor that induced the globally best rule term is referred to as the 'winning processor'. Also the pseudocode above can easily be adapted to any other version of Prism such as PrismTCS. In the case of PrismTCS there would be an additional procedure implemented that

counts the numbers of all classes that are currently presented in one of the attribute lists and sets the target class TC to the minority class. PrismTC could also be employed similarly to PrismTCS by just setting the majority class to TC.

A blackboard approach is used in the PMCRI framework for communication amongst learner algorithms which were described in the above. Blackboard systems are often explained with the metaphor of specialists or experts that are gathered around a blackboard and are confronted with a common problem to solve. The experts may have knowledge of different domains. The blackboard is used by the experts as an information space to derive a common solution. Each expert can observe the blackboard for knowledge or partial solutions that might help him to derive further parts of the solution [22], which the expert writes on the blackboard. In the blackboard system terminology, an expert is called a *Knowledge Source* (KS). Blackboard systems are generally useful for prototyping an application as they can easily be extended by further KSs [23]. Therefore we use a blackboard system to evaluate PMCRI.

In PMCRI, every learning algorithm, as described above, is a certain implementation of a KS that can adapt, interact and evolve within the environment in which it exists. Thus the terms learner and KS may be used interchangeably in the context of PMCRI. In PMCRI each learner KS is executed on a different processor. The specialist area of expertise is determined by the attribute list it uses in order to derive rule terms. A learner KS evolves by communication with other learners: by exchanging information about locally induced rule terms such as the rule term's conditional probability and by exchanging ids that are covered or uncovered by the induced rule term. Each learner KS will subsequently react on receiving such information by manipulating its local specialist knowledge encoded in the local attribute lists.

The usage of a blackboard system allows the structuring of communication as well as the information that is communicated logically by partitioning the blackboard in logically different panels or partitions. These panels or partitions can correspond to different levels of problem solution and thus simplify the management of the problem solving process [24].

The blackboard architecture derived for PMCRI is based on a blackboard server approach supporting asynchronous communication and the partitioning of the blackboard in several panels. Asynchronous communication is important in order to allow the concurrent execution of KSs while commu-

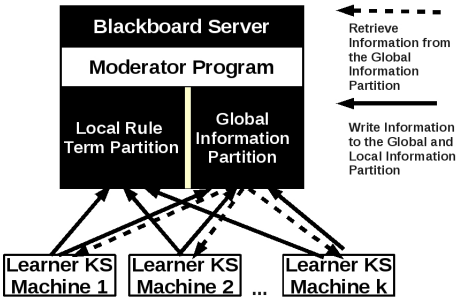

Figure 4: PMCRI's communication pattern using a distributed blackboard architecture.

nicating. The communication pattern of PMCRI is depicted in Figure 4. It consists of a blackboard server that is partitioned into two panels: the 'Local Rule Term Partition' and the 'Global Information Partition' and k learner KS machines. The *knowledge* of the KS machines is determined by the subset of the attribute lists they hold in memory on which they are able to induce the locally best rule term.

The moderator program in Figure 4 is a link between both partitions. It reads information from the 'local rule term partition' and writes information on to the 'global information partition'. The moderator program is in fact a separate KS, however it is hosted on the same machine as the blackboard system as it has very little computation to perform.

Learner KSs monitor the 'global information partition' and write information on the 'global' and 'local information partitions'. From the implementation point of view, the learner KS listening to the blackboard partitions would require constant polling and so constrain the blackboard's communication. For this reason the blackboard notifies KSs whenever something changes on the blackboard. Thus the KSs will only read the blackboard if there is new information available.

Basically the learner KS machines use the learner algorithm described in this section in order to derive the locally best rule term. It will be described below how the three lines of code in the learner pseudocode that involve communication are handled using the blackboard outlined in Figure 4.

When the first communication line in the learner pseudocode (1) is reached:

```
exchange information about t with all other processors (1);
```

the information about the rule term induced needs to be shared. For doing

that the learner KS will write the conditional probability and the target class count with which the rule term was induced together with its own name or identifier on the 'local rule term partition' on the blackboard.

The moderator will read the contents of the 'local rule term partition'. Once all learner KSs have contributed their locally best rule term information, the moderator will compare all the information in order to identify the learner KS that has induced the globally best rule term. This can be described by the pseudocode below. The method or function *'updateModerator'* is called whenever information about a new rule term is written on the blackboard:

```
//Global Variables
int _numberOfLearnerKS;
double _bestProbability = 0;
int _bestTargetClassCount = 0;
String _bestLearnerKSID;
int _counter = 0;
// Method called when information about a new rule
// term is written on the blackboard.
updateModerator(String name, double
 probability, double tc_count){
  _counter++;
  IF((probability>_bestProbability)OR
      ((probability==_bestProbability)AND
       (tc_count>_bestTargetClassCount))){
    _bestProbability = probability;
    _bestTargetClassCount = tc_count;
    _bestLearnerKSID = name;
  }
  IF(_counter==_numberOfLearnerKS){
    write _bestLearnerKSID on Global Information Partition;
    _bestProbability = 0;
    _bestTargetClassCount = 0;
    _counter = 0;
  }
}
```

From the learner KS's point of view, each learner KS will be informed that the 'global information partition' has changed and will read the information, which would be the *_bestLearnerKSID*. If *_bestLearnerKSID* matches the local learner KS's id then the learner KS has induced the globally best rule term. If it does not match the learner KS's id then the learner KS has only induced a locally best rule term. In both cases, the learner KS would continue accordingly as described in the learner KS's pseudocode after communication step (1).

The moderator is only involved in communication step (1). Next the learner KS machines will reach communication steps (2) and (3). The 'winning' learner KS that induced the globally best rule term will reach step (2) and the remaining learner KSs will reach step (3). In steps (2) and (3) the learner KSs will exchange information directly via the blackboard (using the 'global information partition'). At this stage only the 'winning' learner KS is able to derive the indices of the data instances that are uncovered by the last rule term induced. The remaining learner KSs will wait for the information to be available on the 'global information partition'. The 'winning' learner KS will provide this information by writing it on the 'global information partition'. From there on, the learner KSs can work autonomously until the next locally best rule term is induced and the whole communication pattern repeats.

3.3. The Combining Procedure

Regarding the basic learner pseudocode from Section 3.2, we can see that each learner builds a collection of rules. Each learner KS builds rule terms for the same rule simultaneously except that it only appends the rule terms that were locally induced and are confirmed to be globally the best ones in each iteration. Thus for each rule each learner KS will have a collection of rule terms, but will not have all rule terms that belong to the rule. In this sense the concept description induced by each learner is a part of the overall classifier.

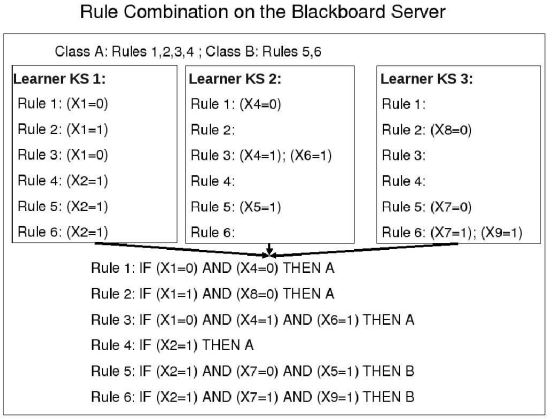

Figure 5: The Combining Procedure of PMCRI.

In the learner pseudocode all rules are stored in a list together with their target class, which is in the order in which the rules were induced. The combining procedure example highlighted in Figure 5 comprises three learner KS machines that induced six 'part-rules'. The part rules are listed in the learner KS's memory and are associated with the target class they were induced for. In order to create the final rule, the combining procedure simply collects all 'part rules' for each rule and appends the rule terms of the 'part rules' using 'AND' operators. It is important to note that the PMCRI framework reproduces exactly the same classifier as the serial version of the Prism algorithm would [14].

3.4. Dealing with Heterogeneous Hardware and Hardware Faults

Overloading the memory of PMCRI will lead the operating systems of each KS machine to buffer parts of the data onto the hard disk and thus would slow PMCRI down due to I/O overheads. In cases where there are fast processors but very little memory it is preferred not to exceed the memory capacity rather than to harvest the processors' power fully. A slight workload imbalance due to the fact that only complete attribute lists are used is accepted.

A hardware failure in PMCRI could be dealt with in two different ways. The first is to execute the rule term calculation and manipulation of attribute lists redundantly on two or more KS machines, which is also done in distributed data analysis systems such as Hadoop [34] and MapReduce. However if there are not enough computers available, then a failure can be recognised by the blackboard if the submission of the KS machines rule term probability times out. In this case a new KS machine could be started, loading the attribute lists of the failed KS machine into its memory. According to Section 3.2 in steps 2 and 3 of the pseudocode description of the learning algorithm, after the induction of each globally best rule term, all KSs exchange information about which attribute list records are uncovered by the globally best rule term and delete the matching attribute list records. Thus all KSs have the attribute list records with the same ids, hence the newly started KS can load its attribute lists and retrieve the ids of the list records from any other KS, keep matching list records in its memory and delete the remaining list records.

4. A Case Study of PrismTCS Parallelised Using the PMCRI Framework

This section examines PMCRI empirically in computational terms.

4.1. System Setup

For evaluation purposes a test implementation of the PrismTCS algorithm has been developed and the diabetes and yeast datasets from the UCI repository [25] were used. To create a larger and thus more challenging workload for PMCRI, each dataset was appended to itself in either a horizontal or a vertical direction. The basic diabetes and yeast datasets each comprise roughly 100,000 data records and 48 attributes. In order to increase the number of attributes the dataset is appended to itself in a horizontal direction, which is referred to as *landscape format*. In order to increase the number of data instances the dataset is appended to itself in a vertical direction, which is referred to as *portrait format*. The reason for replicating the whole dataset is that if the data are sampled, the number of rule terms and rules induced varies. In order to determine the effect of sorting with respect to the size of the data it is required that the same pattern is induced each time, which is the case if the data are appended to itself. Please note that in these experiments PMCRI's learning algorithm is based on PrismTCS and produces exactly the same rules as the serial version of PrismTCS would induce. Therefore there is no concern with issues relating to the comparative quality of rules generated by different algorithms. As all datasets were based on either the yeast or the diabetes dataset, the induced classifiers were identical for all dataset sizes based on yeast and also for all dataset sizes based on diabetes. In particular, the classifier induced on yeast produces 467 rules and the classifier on diabetes 110 rules. The machines used for all experiments in this section had a CPU speed of 2.8 GHz and a memory of 1 GB, of which 800 MB is allocated to PMCRI, the rest is allocated to the operating system. The operating system used was XUbuntu. In general, when this paper mentions processors or machines, learner KS machines are meant. All runtimes recorded for PMCRI comprise the time needed to transfer the training data to each learner KS from a single machine. For the serial version of PrismTCS there is no transfer of data involved.

4.2. Scalability with Respect to the Training Data Size and Capability Barriers of PMCRI

In order to investigate the scalability of PMCRI, with respect to the training data size, runtimes of a fixed processor (learner KS machine) configuration on an increasing workload have been examined. In this paper these experiments are called *size up experiments*. In general, achieving a linear size up is desired, meaning that the runtime is a linear function of the data set size. Usually the number of rules and rule terms induced would influence the workload. However, as the training data used in this section are appended to itself, the number of rules and rule terms induced is fixed for all sizes of the yeast and diabetes datasets in this case study. The fact that the number of rules and rule terms is fixed allows the examination of PMCRI's scalability with respect to the training data size. The term *capability barriers* in this work refers to the maximum workload that the system could cope with for a given number of processors in terms of the training data size in the form of number of instances or attributes. The maximum workload is limited by the amount of data that can be loaded in total into the memory of the network.

Figure 6 shows the runtimes plotted versus the number of training instances for both the yeast and diabetes datasets in portrait format using a different number of processors. In general a linear behaviour for all configurations on both datasets can be observed. The fact that the runtimes on the yeast dataset are generally longer than on the diabetes dataset can be explained by the fact that the yeast data generate more rules than diabetes; 467 versus 110 rules. The linear regression equations below support a linear size up behaviour of PMCRI on the yeast dataset, where x is the relative number of training instances and y the relative runtime in ms:

$Serial \quad PrismTCS(yeast): \quad y = 1.084x \quad (R^2 = 0.997)$
$2 \quad processors(yeast): \quad y = 0.992x \quad (R^2 = 0.995)$
$4 \quad processors(yeast): \quad y = 0.728x \quad (R^2 = 0.999)$
$6 \quad processors(yeast): \quad y = 0.515x \quad (R^2 = 0.998)$
$8 \quad processors(yeast): \quad y = 0.408x \quad (R^2 = 0.998)$
$10 \quad processors(yeast): \quad y = 0.336x \quad (R^2 = 0.998)$
$12 \quad processors(yeast): \quad y = 0.267x \quad (R^2 = 0.997)$

$Serial \quad PrismTCS(diabetes): \quad y = 1.056x \quad (R^2 = 0.998)$
$2 \quad processors(diabetes): \quad y = 0.887x \quad (R^2 = 0.999)$
$4 \quad processors(diabetes): \quad y = 0.675x \quad (R^2 = 0.998)$
$6 \quad processors(diabetes): \quad y = 0.483x \quad (R^2 = 0.997)$
$8 \quad processors(diabetes): \quad y = 0.389x \quad (R^2 = 0.994)$
$10 \quad processors(diabetes): \quad y = 0.322x \quad (R^2 = 0.996)$
$12 \quad processors(diabetes): \quad y = 0.262x \quad (R^2 = 0.992)$

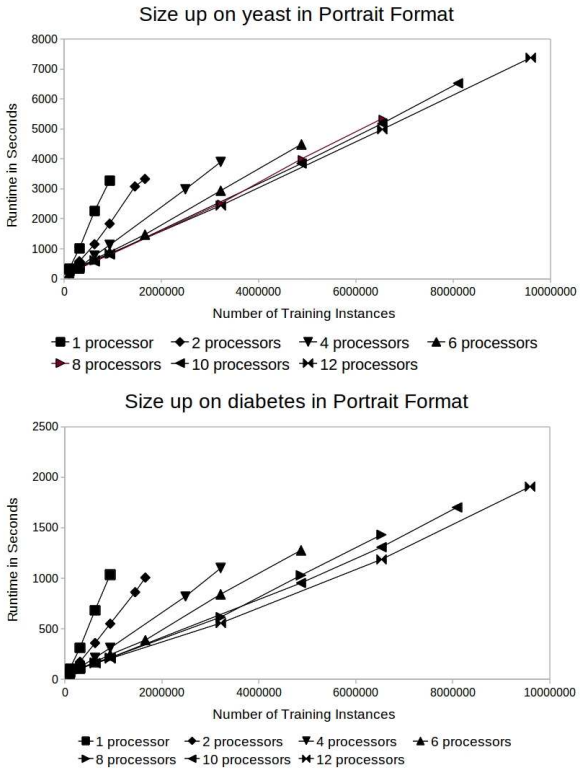

Figure 6: Capability Barriers of PMCRI and PrismTCS with respect to the number of training instances.

What can also be extracted from Figure 6 are the capability barriers of PMCRI. For a particular configuration of PMCRI the capability barrier is equivalent to the number of training instances the framework can cope with, meaning how many attribute lists can be loaded into the collective memory. The capability barrier of PrismTCS on one processor was reached after roughly 900,000 data instances. For PMCRI with two machines it was reached after roughly 1,600,000 data instances and for PMCRI with four machines after 3,200,000 instances, both for yeast and for diabetes in all cases. In PMCRI, the capability barriers can be widened by adding more machines and thus more memory. As Figure 6 shows, in PMCRI, if the amount of processors, and thus workstations with their own local memory, is doubled then the capability barriers also double meaning PMCRI will be

able to load double the amount of training data.

A similar experiment series as for Figure 6 is repeated on the yeast and diabetes datasets but this time the data were growing towards landscape format.

Figure 7 shows the runtimes plotted versus the number of attributes for the yeast and diabetes datasets in landscape format using different numbers of processors.

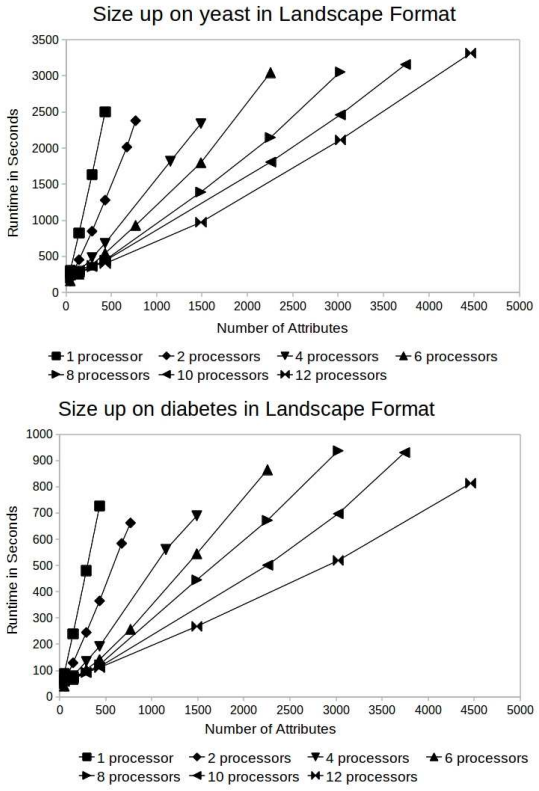

Figure 7: Capability Barriers of PMCRI and PrismTCS with respect to the number of attributes.

$Serial \quad PrismTCS(yeast) : y = 0.911x (R^2 = 0.999)$
2 $processors(yeast) : y = 0.751x (R^2 = 0.998)$
4 $processors(yeast) : y = 0.489x (R^2 = 0.997)$
6 $processors(yeast) : \quad y = 0.380 \quad (R^2 = 0.992)$
8 $processors(yeast) : \quad y = 0.238x \quad (R^2 = 0.992)$

10 $processors(yeast): \quad y = 0.167x \quad (R^2 = 0.989)$
12 $processors(yeast): \quad y = 0.121x \quad (R^2 = 0.979)$

$Serial \quad PrismTCS(diabetes): \quad y = 0.914x \quad (R^2 = 0.999)$
2 $processors(diabetes): \quad y = 0.766x \quad (R^2 = 0.999)$
4 $processors(diabetes): \quad y = 0.551x \quad (R^2 = 0.997)$
6 $processors(diabetes): \quad y = 0.438x \quad (R^2 = 0.995)$
8 $processors(diabetes): \quad y = 0.298x \quad (R^2 = 0.996)$
10 $processors(diabetes): y = 0.197x \quad (R^2 = 0.990)$
12 $processors(diabetes): \quad y = 0.135x \quad (R^2 = 0.998)$

Again, a closer look has been taken at the equations. Once more, a size up behaviour better than the theoretical linear one can be observed. We also observe the same behaviour for the capability barriers as for portrait data. We can see that the capability barrier of PrismTCS with PMCRI with 2 processors was reached after 768 attribute lists and for PrismTCS with PMCRI with four processors after roughly 1,488 attribute lists and so on.

4.3. Speed up of PMCRI

Standard metrics to evaluate a parallel algorithm or a parallel architecture are the speed up factors and the efficiency [26, 27]. With the speed up factors one can compare by how much the parallel version of an algorithm is faster using p processors than with 1 processor.

$$S_p = \frac{R_1}{R_p} \qquad (1)$$

Formula 1 represents the speed up factors S_p. R_1 is the runtime of the algorithm on a single machine and R_p is the runtime on p machines. In the ideal case, the speed up factors are the same as the number of processors used. For example, if two processors were used then a speed up factor of 2 means that we gained 100% benefit from using an additional processor [28]. In reality, the speed up factor will be below the number of processors for various reasons, such as a communication overhead imposed by each processor which would be, in our case, caused by communication of information about rule terms and indices of list records. Then there is also the synchronisation overhead. For example, in the case of PMCRI, if a processor has induced the locally best rule term it has to wait for the remaining machines to finish their rule term induction in order to receive or derive the indices that are covered by the globally best rule term. However, as stated in Section 3.1, the relative workloads of each processor stay constant; thus a synchronisation overhead will not be overwhelming.

Figure 8 shows the speed up factors of PrismTCS parallelised using PMCRI for different sizes of the yeast dataset (upper graph) and the diabetes dataset (lower graph), both in portrait format. Configurations of 1, 2, 4, 6, 8, 10 and 12 processors were used. It can be observed that, for a fixed dataset size with an increase of the number of processors, the speed up factors also increase, until a maximum speed up factor is reached and start to decrease again. It can also be observed that the larger the dataset size, the more processors are needed in order to reach the maximum speed up. For diabetes with 103,680 data records the maximum is reached with 4 processors. For diabetes with 311,040 records the maximum is reached with 8 machines. For the third and fourth data series of 622,080 and 933,120 records respectively, the maximum was not reached with all processors available. A similar behaviour can be observed for the speed up factors of the yeast dataset.

All datasets for the portrait format comprised 48 attributes and thus 48 attribute lists. It is not possible to distribute 48 attributes evenly on a 10 processor configuration. This is the slight downside of using complete attribute lists as already discussed in Section 3.1. However, in this case, the attribute lists have been distributed 'as evenly as possible' by assigning to 8 of the learner KS machines 5 attribute lists and to 2 machines only 4, which adds up to 48 attribute lists. However, as it can be seen in Figure 8, this workload imbalance is almost imperceptible.

Loosely speaking, regarding the speed up factors, it can be observed that the larger the data in terms of instances the more PMCRI benefits from using additional processors. In general, the speed up behaviour observed in this section is a normal behaviour for most parallel algorithms that need to communicate in order to share intermediate results or information [26].

Figure 9 shows the speed up factors of PrismTCS parallelised using PMCRI for different sizes of the yeast dataset (upper graph) and the diabetes dataset (lower graph), both in landscape format. Again, as for portrait format, configurations of 1, 2, 4, 6, 8, 10 and 12 processors were used.

The same tendency as for portrait data can be observed, which is that for a fixed data size an increasing number of processors also increases the speed up factors. The speed up factors again reach a maximum and then start to decrease again. Also, the larger the number of attributes, the more processors are required to reach the maximum speed up. For diabetes with 48 attributes the maximum is reached with 4 processors. For diabetes with 144 attributes the maximum is reached with 8 machines. For the third data series of 288 attributes the maximum seems to be reached with 12 machines however the

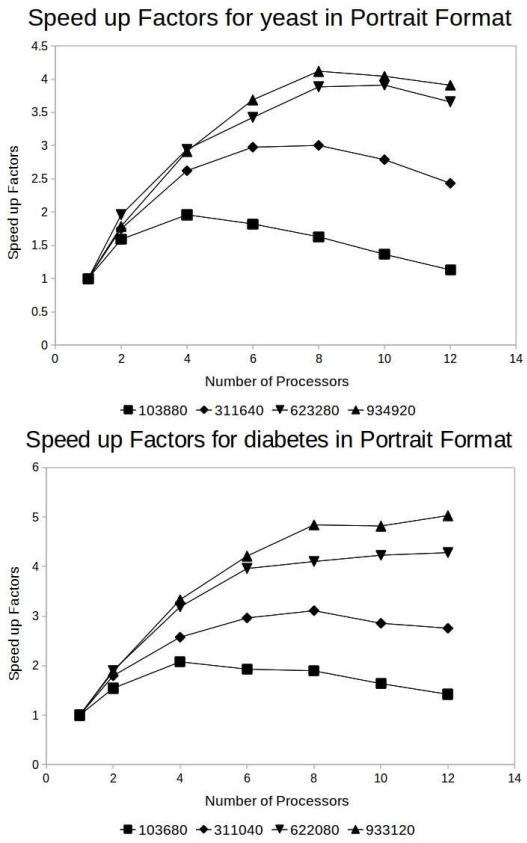

Figure 8: Speed up factors of PMCRI using PrismTCS on the yeast and diabetes dataset in portrait format.

maximum was not reached with all processors available for diabetes with 432 attributes. A similar behaviour can be observed for the speed up factors of the yeast dataset.

Again the speed up behaviour observed in this section is a normal behaviour, as for most parallel algorithms that need to communicate in order to share intermediate results or information.

4.4. Communication Efficiency of PMCRI

Efficiency is a performance metric represented by Formula 2:

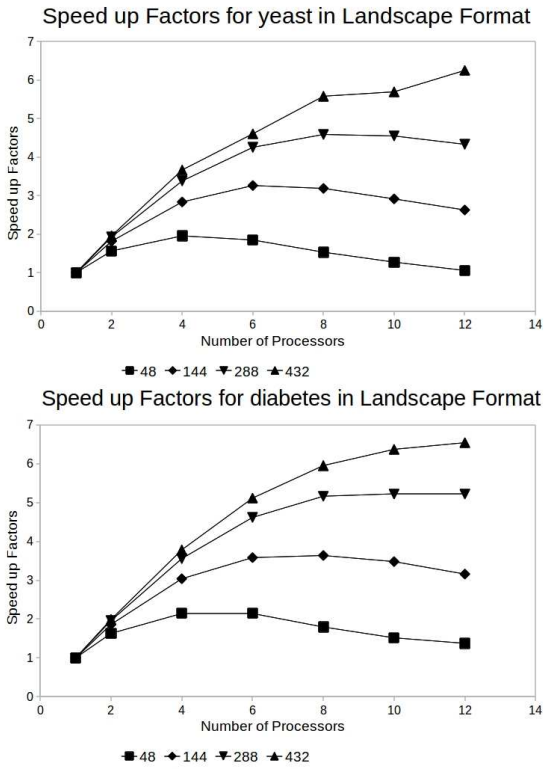

Figure 9: Speed up factors of PMCRI using PrismTCS on the yeast and diabetes dataset in landscape format.

$$E_p = \frac{S_p}{p} \qquad (1)$$

S_p is the speed up factor divided by the number of processors p. The efficiency ranges from 0 to 1. The efficiency is equivalent to the percentage with which PMCRI profits from each processor. In other words, the efficiency is the amount of speed up achieved per processor [28].

Comparing the efficiencies from portrait and landscape data for yeast, 48 attributes for landscape are equivalent in size to 103,880 instances in portrait format, 144 attributes in landscape are equivalent in size to 311,640 instances and so on for yeast and analogously for diabetes. Now the efficiencies for the yeast data in portrait and landscape format are plotted in the top half of Figure 10 and the efficiencies for the diabetes data in portrait and landscape format are plotted in the bottom half of Figure 10.

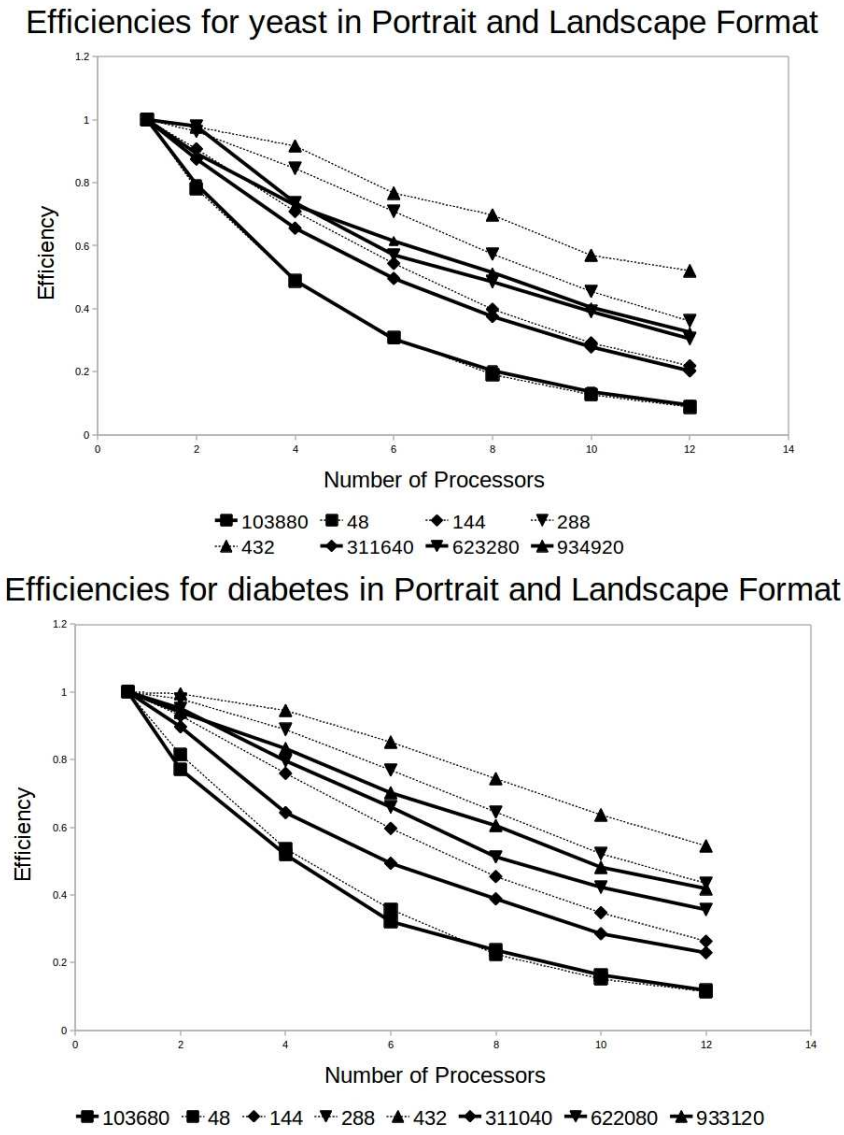

Figure 10: Efficiencies comparison of the data growing towards landscape and portrait format. The efficiencies for the yeast data are plotted in the top half of the figure and the efficiencies for the diabetes data are plotted in the bottom half of the figure.

Efficiency series for portrait format are plotted in solid lines whereas the ones for landscape format are plotted in dashed lines. For both, the diabetes

and the yeast data, it can be observed that the efficiencies for data in portrait format compared with the data in the equivalently-sized landscape format are always lower. By 'equivalently-sized' here is meant that the number of attribute values in the dataset, i.e. the number of instances multiplied by the number of attributes, is the same in both cases. For the yeast data, a dataset of 48 attributes is equivalent in size to one of 103880 instances, one of 144 attributes is equivalent to one of 311640 instances, one of 288 attributes is equivalent to 623280 instances and one of 432 attributes is equivalent to one of 934920 instances. And for the diabetes data, a dataset of 48 attributes is equivalent in size to one of 103680 instances, one of 144 attributes is equivalent to one of 311040 instances, one of 288 attributes is equivalent to 622080 instances and one of 432 attributes is equivalent to one of 933120 instances.

The higher efficiency of PMCRI on landscape formatted data, compared with portrait formatted data, can be explained by the fact that the number of indices of list records, that are covered or uncovered by the currently induced rule term, increases with the number of data records, but not with the number of attributes. Hence the more data records there are the more indices need to be communicated between the KSs and the blackboard. This is not the case for landscape formatted data.

5. J-PMCRI Case Study: PMCRI Incorporating J-pruning

Like decision tree induction algorithms, Prism also tends to have a higher potential to overfit on larger training datasets than on smaller ones. In previous work about parallel decision tree induction, pruning, in general, has been neglected as it seems to be computationally inexpensive when pruning is applied after the classifier has been induced. However, the results in [14] strongly suggest that pre-pruning would be a valuable asset in reducing the runtime of the classifier induction. This is because the fewer rule terms there are to be induced, the fewer iterations to derive the rule set are needed. This Section extends the PMCRI framework by a J-pruning facility that does the pruning during the classifier's induction.

5.1. J-PMCRI: A J-pruning Facility for PMCRI

J-pruning, a pre-pruning method has been developed for the Prism family of algorithms as well as for decision tree induction, in order to prevent Prism classifiers from overfitting on the training data [17]. J-pruning shows good

results on both Prism and decision tree induction algorithms [17, 32]. It is based on an information theoretic measure, the J-measure [18], that is used to quantify the theoretical information content of a rule. J-pruning is also interesting for computational efficiency considerations as it reduces the number of rules and rule terms induced considerably, as pointed out in [14, 19]. The theoretical average information content of a rule of the form *IF* $Y = y$ *THEN* $X = x$ can be measured in bits and is denoted by *J(X, Y=y)* [18].

$$J(X; Y = y) = p(y) \cdot j(X; Y = y) \qquad (1)$$

Equation (1) shows the product of the probability with which the left hand side of the rule will occur $p(y)$ and the *j-measure* $j(X; Y = y)$ (with a lower case j), also called cross entropy, which measures the goodness-of-fit of a rule and is defined in Equation (2):

$$j(X; Y = y) = p(x \mid y) \cdot log_2(\tfrac{p(x|y)}{p(x)}) + (1 - p(x \mid y)) \cdot log_2(\tfrac{(1-p(x|y))}{(1-p(x))}) \qquad (2)$$

Bramer bases J-pruning on the assumption that a high J-value of a classification rule also results in a high predictive accuracy [17]. J-pruning uses the J-value as a means to quantify if appending further rule terms is likely to improve a rule's predictive accuracy or is prone to overfit. Bramer's basic *J-pruning* is applied to Prism by calculating the J-value of the rule before the induction of a new rule term and the J-value that the rule would have after a newly induced rule term is appended. If the J-value goes up then the rule term is appended. In the case where the J-value goes down, the rule term is not appended, the rule currently being induced is truncated and is treated as if a clash had occurred, as described briefly in Section 2.3.1.

It is important to note that each learner KS machine is able to calculate the J-value of the locally best rule term induced solely from the attribute lists from which the rule term was induced.

The first factor *p(y)* is the probability with which the left hand side of the rule is covered. The number of instances that are covered by the induced rule term is equivalent to the number of instances covered by the currently induced rule. This number can easily be retrieved from the attribute list the term was induced from. The second factor is called the j-measure. The factors and

quotients needed to calculate the j-measure include the terms $p(x \mid y)$, which is the probability with which the rule covers the target class and *p(x)* which is the probability that the target class occurs in the training data before the rule has been induced. $p(x \mid y)$ is equivalent to the conditional probability with which the rule was induced and does not need to be calculated again and *p(x)* can easily be calculated before the induction of the present rule has been started using any attribute list.

Now only knowing the actual rule term that has been induced locally, each learner KS is able to calculate the J-value of the total rule if the rule term would be appended to the rule. Once the locally best rule term is induced, the J-value can be advertised on the blackboard together with the remaining information needed by the moderator in order to derive the globally best rule term. Thus there is no additional synchronisation step needed. The moderator can now also take the J-values of all 'candidate' rule terms into account. The moderator either writes the name of the winning learner KS on the 'global information partition' or if the globally best rule term would decrease the rule's J-value, it writes a message on the partition that informs the learner KS that 'nobody has induced an appropriate term' or in short *'Do Pruning'*. Displayed below is the modified code of the moderator that now also takes the J-values into account. The modified parts compared with the standard PMCRI moderator pseudocode are in lines 5, 14, 15, 19, 20, 21 and 25:

```
1  //Global Variables
2  int _numberOfLearnerKS;
3  double _bestProbability = 0;
4  int _bestTargetClassCount = 0;
5  double _bestJValue = 0;
6  String _bestLearnerKSID;
7  int _counter = 0;
8  //Method called when information about a new rule term
   //is written on the blackboard.
9  public void updateModerator(String name, double
     probability, double tc_count, double jvalue){
10   _counter++;
11 IF((probabilities>_bestProbability)OR
12       ((probability==_bestProbability)AND
13        (tc_count>_bestTargetClassCount))){
14     IF(jvalue>=_bestJValue){
15       _bestJValue = jvalue;
16       _bestProbability = probability;
17       _bestTargetClassCount = tc_count,;
18       _bestLearnerKSID = name;
19     }ELSE{
20       _bestLearnerKSID = Do Pruning;
21     }
```

```
22   }
23   IF(counter== numberOfLearnerKS){
24     write _bestLearnerKSID on Global Information Partition;
25     _bestJValue = 0;
26     _bestProbability = 0;
27     _bestTargetClassCount = 0;
28     _counter = 0;
29   }
30 }
```

So far for the learner KS machines, the source code is similar to that for standard PMCRI, except that the rule term information is enhanced by the J-value. After a learner KS has submitted the locally best rule term it awaits the name or id of the 'winning learner KS' to be advertised on the blackboard. However the name or id might actually be replaced by the information 'Do Pruning' and the learner KS has to handle this event as well. If the rule induced so far covers the target class as the majority class, then the rule is kept and all instances covered by the rule are treated as if they belong to the target class, otherwise the rule is discarded and clash resolution as described briefly in Section 2.3.1 is invoked. The pseudocode below shows the learner code from Section 3 modified to handle the information 'Do Pruning'. The modified parts are in lines 16 and 17:

```
1  //Global Variables
2  TargetClass _tc;
3  List _rules;
4  Rule _rule;
5  List _fromRulesCoveredIds;
6  int[] _toDeleteIds;
7  while(while attribute lists are not empty
         and contain _tc){
8     _rule = new Rule();
9     while((local lists contain tc)AND
            (_tc is not the only class)){
10       induce locally best rule term t;
11       exchange information about t with all other
         processors (1);
12       if(t is globally best rule term){
13          rule.append(t);
14          toDeleteIds = Generate by t uncovered ids;
15          communicate toDeleteIds to remaining
            processors (2);
16       } else if(Do Pruning){
17          invoke clash resolution.
18       } else {
19          retrieve toDeleteIds from winning processor (3);
20       }
21       delete all list instances matching toDeleteIds;
22       clear toDeleteIds;
23    }
```

```
24    _fromRulesCoveredIds.add(ids present in current
         attribute lists);
25    rules.add(rule);
26    restore all attribute lists to their initial size;
27    delete all instances from all attribute lists that
      match ids in _fromRulesCoveredIds;
28 }
```

Overall it can be seen that there is no further synchronisation step involved compared with PMCRI.

5.2. An Evaluation of J-PMCRI

For the evaluation two very large datasets, one with a moderate number of attributes and one with a large number of attributes have been used from the infobiotics repository [29], which comprises very large datasets for benchmarking purposes. The first dataset is called infobio2 in this paper. It comprises 60 attributes, 4 classifications and more than 2.3 million training instances. The second dataset used is called infobio3. Infobio3 comprises 220 attributes, 4 classifications and 2.5 million data records. This section focuses on the performance of PMCRI when pruning is incorporated. Thus with infobio2 and infobio3 two real large noisy datasets are used. The data are sampled and the number of rule terms induced is measured. The number of rules and rule terms will vary with different sample sizes, thus a direct comparison of the runtimes with respect to the data sample size (size up experiments) is not possible as the runtimes are influenced by the number of rules and rule terms induced. However what can be examined are the speed up factors for the same data size with a different number of processors, and the behaviour of the number of rules and rule terms induced with respect to the number of training instances. All runtimes recorded for J-PMCRI include the time needed to transfer the training data to each learner KS from a single machine. For the serial version of PrismTCS with J-pruning there is no transfer of data involved. It is expected that, for different amounts of training data, the number of rule terms induced will fluctuate but the trend to an increase of the number of rule terms is expected to be low or not present for Prism implementations using J-pruning.

Figure 11 shows the number of rule terms induced on the infobio2 and infobio3 datasets using PrismTCS parallelised with J-PMCRI. The number of rule terms stays relatively stable with minor fluctuations.

The fact that the number of rule terms does not increase considerably with an increasing number of training instances when using J-pruning strongly

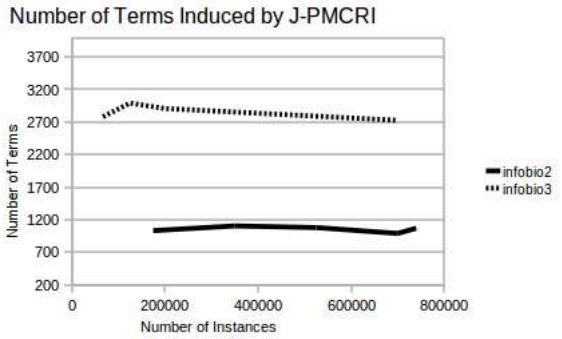

Figure 11: Number of rule terms induced using PrismTCS parallelised with J-PMCRI.

suggests the use of J-pruning not only for inducing more generalised rules but also for scaling purposes; the fewer rule terms have to be induced, the fewer iterations of the algorithm through the training data are required and thus the faster the rule induction process. The scalability of J-PMCRI is examined using the PrismTCS algorithm on the datasets infobio2 and infobio3. For this purpose the speed up factors are calculated.

Figure 12 shows speed up factors plotted versus the number of processors used on fixed numbers of training instances. What can be observed here is similar to the results of the first version of PMCRI. The speed up factors increase with an increasing number of processors then decrease again. This can be explained by the fact that using more processors will impose a larger communication overhead as well as managing overhead. However, what can also be observed is that the best speed up is reached for a larger number of processors if the number of training instances is large as well. Thus loosely speaking the larger the number of training instances, the more J-PMCRI benefits from using additional processors.

Please note that the experiments illustrated in Figure 12 do not include an experiment with all 2.3 million data records. The reason for excluding this run here is that it was not possible to perform this experiment on a 1 processor configuration due to memory constraints. There is a workload imbalance for an 8 processor configuration for infobio2. Also for the infobio3 dataset there are a number of cases where the workload is unbalanced, that is the case for the 6 processor configuration , the 8 processor configuration, and the 12 processor configuration. However the workload imbalance is hardly

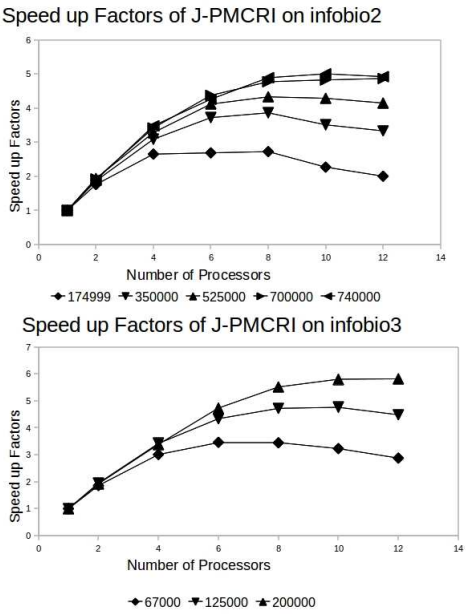

Figure 12: Speed up factors obtained for J-PMCRI with the PrismTCS algorithm.

noticeable in the results obtained.

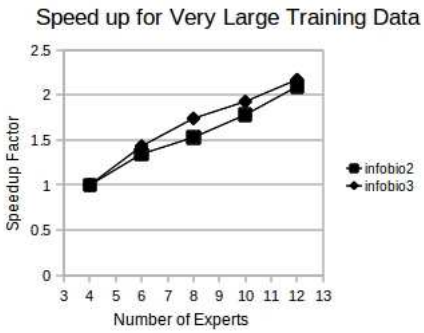

Figure 13: Speed up factors obtained using J-PMCRI on very large training data that cannot be handled by a single machine.

Figure 13 shows the speed up factors obtained on the total training dataset of the infobio2 and the infobio3 datasets. The experiments on info-

bio2 comprised a total of 2,338,121 instances and 60 attributes. For infobio3 there were 700,336 instances and 220 attributes. It would have been possible to run J-PMCRI on all 2.5 million instances of infobio3, but only with a 12 processor machine configuration due to memory constraints. However for the calculation of the speed up factors, runtimes of a PMCRI configuration with less than 12 processors were needed, thus the number of data instances needed to be reduced. The speed up factors are based on a 4 processor configuration and not on a single processor. The reason for this is that a minimum of 4 learner KS machines is needed in order to be able to process the large amount of training data due to memory constraints. Also the purpose of these experiments is to show that more than 12 learner KS machines are beneficial if the data are large enough. What can be seen in Figure 13 is that the speed up factors are still increasing with a total of 12 learner KS machines. This is different compared with the experiments outlined in Figure 12, where 12 learner KS machines were not any more beneficial. However the experiments in Figure 12 were conducted on much smaller data samples. The fact that the speed up factors are still growing for 12 processors highlights even more the observed behaviour that using more processors is more beneficial the larger the amount of training data.

Both frameworks, PMCRI and J-PMCRI are useful in their own right, just as Prism with and Prism without J-pruning are. PMCRI could be used in any case, however, for some data a Prism algorithm with pre-pruning might be better. For example, if there are a lot of noise or there are a lot of missing values in the dataset, in these cases Prism with J-pruning is likely to achieve a better classification accuracy [38], hence J-PMCRI should be used in order to reduce overfitting on the data if required. However if Prism does not tend to overfit on the data, then PMCRI will be sufficient. In the computational sense, both frameworks exhibit a similar speed up behaviour.

6. Conclusions

This work presents the first attempt to parallelise the Prism family of algorithms for modular classification rule induction. First the problem of data mining on massive datasets was discussed with the conclusion that parallel algorithms are needed in order to scale up data mining with the focus on parallelisation on networks of standard computer workstations. Parallelisation in the area of classification rule induction has so far focused on the induction of decision trees. Section 2 highlights the Prism family of algorithms as an

alternative to decision trees that performs better than decision trees in many cases as it avoids the replicated subtree problem. Section 3 introduces the blackboard based PMCRI framework that allows the parallelisation of any member of the Prism family in a network of computer workstations. PMCRI induces exactly the same rules as any serial Prism algorithm and is evaluated in Section 4. It has been found that PMCRI scales linearly with the increasing amount of data instances and attributes. Also the capability barrier, the maximum amount of data that can be loaded into the PMCRI system scales linearly with an increasing amount of memory. Furthermore it has been found that the larger the training data used, the more processors are beneficial. Section 5 discusses a variation of PMCRI, J-PMCRI that incorporates a pre-pruning facility, J-pruning. Pre-pruning is particularly interesting in computational terms as it prevents unnecessary rule terms being induced in the first place and thus lowers the computational demands of Prism algorithms in general, as less iterations for the rule term induction have to be performed. Section 5 also evaluates J-PMCRI in computational terms with similar results as for PMCRI. Loosely speaking PMCRI shows a nice scalability with respect to the data volume and number of workstations used. Ongoing work comprises the extension of the framework to induce general rules that do not predict a certain class but describe important relationships in the dataset. Also the integration of a further, only recently developed, pre-pruning version for Prism algorithms, Jmax-pruning [30, 31] is investigated for integration in the PMCRI framework. In general the development of PMCRI allows the application of Prism algorithms on a larger range of datasets that have previously simply been too large to be analysed using the Prism approach.

References

[1] D. Berrar, F. Stahl, C. S. G. Silva, J. R. Rodrigues, R. M. M. Brito, W. Dubitzky, Towards data warehousing and mining of protein unfolding simulation data, Journal of Clinical Monitoring and Computing 19 (2005) 307–317.

[2] F. Stahl, D. Berrar, C. S. G. Silva, J. R. Rodrigues, R. M. M. Brito, W. Dubitzky, Grid warehousing of molecular dynamics protein unfolding data, in: Proceedings of the Fifth IEEE/ACM Int'l Symposium on Cluster Computing and the Grid, IEEE/ACM, Cardiff, 2005, pp. 496–503.

[3] B. McClean, C. Hawkins, A. Spagna, M. Lattanzi, B. Lasker, H. Jenkner, R. White, New horizons from multi-wavelength sky surveys, in: Proceedings of the 179th Symposium of the International Astronomical Union held in Baltimore.

[4] A. Szalay, The Evolving Universe, ASSL 231, 1998.

[5] E. B. Hunt, P. J. Stone, J. Marin, Experiments in induction, Academic Press, New York, 1966.

[6] R. J. Quinlan, Induction of decision trees, Machine Learning 1 (1986) 81–106.

[7] R. J. Quinlan, C4.5: programs for machine learning, Morgan Kaufmann (1993).

[8] R. S. Michalski, On the Quasi-Minimal solution of the general covering problem, in: Proceedings of the Fifth International Symposium on Information Processing, Bled, Yugoslavia, pp. 125–128, 1969.

[9] J. Shafer, R. Agrawal, M. Metha, SPRINT: a scalable parallel classifier for data mining, in: Proc. of the 22nd Int'l Conference on Very Large Databases, Morgan Kaufmann, 1996, pp. 544–555.

[10] M. Joshi, G. Karypis, V. Kumar, Scalparc: a new scalable and efficient parallel classification algorithm for mining large datasets, in: Parallel Processing Symposium, 1998. IPPS/SPDP 1998. Proceedings of the First Merged International ... and Symposium on Parallel and Distributed Processing 1998, pp. 573–579.

[11] J. Cendrowska, PRISM: an algorithm for inducing modular rules, International Journal of Man-Machine Studies 27 (1987) 349–370.

[12] I. H. Witten, F. Eibe, Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations, Morgan Kaufmann, 1999.

[13] M. A. Bramer, Inducer: a public domain workbench for data mining, International Journal of Systems Science 36 (2005) 909–919.

[14] F. Stahl, Parallel Rule Induction, Ph.D. thesis, University of Portsmouth, 2009.

[15] M. A. Bramer, Automatic induction of classification rules from examples using N-Prism, in: Research and Development in Intelligent Systems XVI, Springer-Verlag, Cambridge, 2000, pp. 99–121.

[16] R. Kerber, Chimerge: Discretization of numeric attributes, in: AAAI, pp. 123–128, 1992.

[17] M. A. Bramer, An information-theoretic approach to the pre-pruning of classification rules, in: B. N. M Musen, R. Studer (Eds.), Intelligent Information Processing, Kluwer, 2002, pp. 201–212.

[18] P. Smyth, R. M. Goodman, An information theoretic approach to rule induction from databases, Transactions on Knowledge and Data Engineering 4 (1992) 301–316.

[19] F. Stahl, M. Bramer, M. Adda, Parallel rule induction with information theoretic pre-pruning, in: Twenty-ninth SGAI International Conference on Innovative Techniques and Applications of Artificial Intelligence, Springer, 2010, pp. 151–164.

[20] F. Provost, Distributed data mining: Scaling up and beyond, in: Advances in Distributed and Parallel Knowledge Discovery, MIT Press, 2000, pp. 3–27.

[21] F. Stahl, M. A. Bramer, M. Adda, PMCRI: A parallel modular classification rule induction framework, in: MLDM, Springer, 2009, pp. 148–162.

[22] L. Nolle, K. C. P. Wong, A. Hopgood, DARBS: a distributed blackboard system, in: Proceedings of the Twenty-first SGES International Conference on Knowledge Based Systems and Applied Artificial Intelligence, Springer, Cambridge, 2001, pp. 161–170.

[23] D. Corkill, Blackboard systems, AI Expert 6 (1991) 40–47.

[24] Y. C. Jiang, Z. P. Xia, Y. P. Zhong, S. Y. Zhang, An adaptive adjusting mechanism for agent distributed blackboard architectures, Microprocessors and Microsystems 29 (2005) 9–20.

[25] C. L. Blake, C. J. Merz, UCI repository of machine learning databases, Technical Report, University of California, Irvine, Department of Information and Computer Sciences, 1998.

[26] J. L. Hennessy, D. A. Patterson, Computer Architecture A Quantitative Approach, Morgan Kaufmann, USA, fourth edition, 2007.

[27] K. Hwang, F. A. Briggs, Computer Architecture and Parallel Processing, McGraw-Hill Book Co., first edition, 1990.

[28] C. Xavier, S. S. Iyengar, Introduction to Parallel Algorithms, John Wiley & Sons, Inc., 1998.

[29] J. Bacardit, N. Krasnogor, The Infobiotics PSP benchmarks repository, Technical Report, 2008.

[30] F. Stahl, M. Bramer, Induction of modular classification rules: Using jmax-pruning, in: Thirtieth SGAI International Conference on Innovative Techniques and Applications of Artificial Intelligence, Springer, 2011, pp. 79–92.

[31] F. Stahl, M. Bramer, Jmax-pruning: A facility for the information theoretic pruning of modular classification rules, Knowledge-Based Systems 19(0) (2012) 12–19.

[32] M. Bramer, Using J-pruning to reduce overfitting in classification trees, Knowledge-Based Systems 15(5-6) (2002) 301–308.

[33] A. Mutlu, P. Senkul, Y. Kavurucu, Improving the scalability of ILP-based multi-relational concept discovery system through parallelization, Knowledge-Based Systems 27(0) (2012) 352–368.

[34] http://hadoop.apache.org/ (last visited in August 2011).

[35] http://www.cs.waikato.ac.nz/ml/weka/ (last visited in December 2011).

[36] W. Cohen. Fast effective rule induction. In *Proceedings of the Twelfth International Conference on Machine Learning*, pages 115–123. Morgan Kaufmann, 1995.

[37] D. T. Pham and M. S. Aksoy. Rules: A simple rule extraction system. *Expert Systems with Applications*, 8(1):59–65, 1995.

[38] M. Bramer. Using J-Pruning to Reduce Overfitting of Classification Rules in Noisy Domains. In DEXA - Database and Expert Systems Applications, pages 433–442. Springer-Verlag, 2002.