PMCRI: A Parallel Modular Classification Rule Induction Framework

Frederic Stahl[1], Max Bramer[1], Mo Adda[1]
[1]University of Portsmouth, Buckingham Building, Lion Terrace,
Portsmouth PO1 3HE, United Kingdom
{Frederic.Stahl, Max.Bramer, Mo.Adda}@port.ac.uk

Abstract. In a world where massive amounts of data are recorded on a large scale we need data mining technologies to gain knowledge from the data in a reasonable time. The Top Down Induction of Decision Trees (TDIDT) algorithm is a very widely used technology to predict the classification of newly recorded data. However alternative technologies have been derived that often produce better rules but do not scale well on large datasets. Such an alternative to TDIDT is the PrismTCS algorithm. PrismTCS performs particularly well on noisy data but does not scale well on large datasets. In this paper we introduce Prism and investigate its scaling behaviour. We describe how we improved the scalability of the serial version of Prism and investigate its limitations. We then describe our work to overcome these limitations by developing a framework to parallelise algorithms of the Prism family and similar algorithms. We also present the scale up results of a first prototype implementation.

1. Introduction

The growing interest and importance of commercial knowledge discovery and data mining techniques has led to a growing interest in the area of classification rule induction from data samples to enable the classification of previously unseen data. Research in classification rule induction can be traced back at least to the 1960s [1].A very widely used method to induce classification rules is TDIDT [2] which has given rise to a variety of implementations such as C5.0. However alternative algorithms exist such as the Prism algorithm [3]. Prism produces more generalised rules than TDIDT and thus tends to perform better on noisy datasets. As a result Prism has been used in areas where datasets are naturally noisy such as image recognition [4] or text classification. Prism is also the base for further data mining algorithms such as PrismTCS [5], N-Prism [6]. A free implementation of Prism can be found in the WEKA package [7] and also in the Inducer workbench[8]. The increase in performance of computer hardware such as CPU power and disc storage and sensors to record data such as CCTV cameras enables companies and researchers to generate and store larger and larger datasets to which they still wish to apply classification rule induction algorithms. This has led to the exploration of a new niche in data mining, parallel and distributed data mining. So far, work on distributed and parallel classification rule induction has been focused on the well-established TDIDT approach. Notable developments are the SLIQ [9] and its successor the SPRINT [10] algorithm. The latter achieves an almost linear scale up with respect to the number of CPUs and the sample size. However, very little work has been done on scaling up alternative algorithms such as Prism. One approach to scaling a data mining algorithm

is to sample the data before the algorithm is applied. Catlett's work [11] showed that sampling of data results in a loss of accuracy in the induced classifier. However Catlett's research was conducted 17 years ago and the datasets he used were fairly small compared with those used today. Frey and Fisher found in 1999 that the rate of increase of accuracy slows down with the increase of the sample size [12]. This resulted in seeking optimized methods for sampling massive datasets such as progressive sampling [13]. Whereas sampling might be an option for predictive modelling, scaling up data mining algorithms is still desirable in applications that are concerned with the discovery of new knowledge. Chan and Stolfo considered a way to scale up classification rule induction by dividing the data into subsets that fit in a single computer's memory and then generating a classifier on each data subset in parallel on several machines[14, 15]. The different classifiers generated are then combined by using various algorithms in order to achieve a final classifier. Despite the significant reduction of run times of the classification rule induction process, Chan and Stolfo's studies also showed that this approach does not achieve the accuracy of a single classifier induced on the same training data. In order to meet the need for a well scaling, more generalised and thus noise tolerant classifier, we investigate and improve PrismTCS's scaling behaviour and derive a parallel approach to inducing classification rules in parallel for algorithms based on the Prism family. We present a framework that induces modular classification rules in parallel based on the PrismTCS algorithm and evaluate its scaling behaviour.

2. Inducing Modular Classification Rules

The main drawback of the TDIDT approach, also often called the *divide and conquer* approach, lies in the intermediate representation of its classification rules in the form of a decision tree. Rules such as:

IF a = 1 AND b = 1 THEN class = 1
IF c = 1 AND d = 1 THEN class = 0

which have no attribute in common, could not be induced directly using the TDIDT approach. In such cases, TDIDT will first need to introduce additional tests that are logically redundant simply to force the rules into a form suitable for combining into a tree structure. This will inevitably lead to unnecessarily large and confusing decision trees.Cendrowska designed the original Prism algorithm to induce directly sets of 'modular' rules that generally will not fit conveniently into a tree structure, thus avoiding the redundant terms that result when using the TDIDT approach. Prism generally induces rule sets that tend to overfit less compared with TDIDT, especially if it is applied to noisy datasets or datasets with missing values [6]. Cendrowska's Prism algorithm follows the *separate-and-conquer* approach which learns a rule that explains a certain part of the training data. It then separates the data explained by the rule induced and induces the next rule using the remaining data. Thus it recursively "conquers" until no training data is left. This strategy can be traced back to the AQ learning system [16]. The basic separate and conquer algorithm can be described as follows:

```
Rule_Set = [];
While Stopping Criterion not satisfied{
    Rule = Learn_Rule;
    Remove all data instances covered from Rule;
}
```

The algorithm specific procedure *Learn_Rule* learns the best rule for the current training data subset. After each induced rule all data instances that are not covered are deleted and the next rule is learned from the remaining data instances. This is done until a *Stopping Criterion* is fulfilled. Also the *Stopping Criterion* is an algorithm specific one that differs from algorithm to algorithm. PrismTCS (**Prism** with **T**arget **C**lass, **S**mallest first) a version of Prism that attempts to scale up Prism to larger datasets has been developed by one of the present authors [5]. Whereas in PrismTCS the *separate-and-conquer* approach is applied only once, in the original Prism algorithm it is applied for each class in turn. PrismTCS has a comparable predictive accuracy to that of Prism [5].

Our implementation of PrismTCS for continuous data only is summarised in the following pseudo code:

```
(a) working dataset W = restore Dataset;
    delete all records that match the rules that have
    been derived so far and select the ;
    target class i = class that covers the fewest
    instances in W;
(b) For each attribute A in W
    - sort the data according to A;
    - for each possible split value v of attribute A
      calculate the probability that the class is i
      for both subsets A < v and A ≥ v;
(c) Select the attribute that has the subset S with
    the overall highest probability;
(d) build a rule term describing S;
(e) W = S;
(f) Repeat b to e until the dataset contains only
    records of class i. The induced rule is then
    the conjunction of all the rule terms built at
    step d;
(g) restore Dataset = restore Dataset – W;
    Repeat a to f until W only contains instances of
    class i or is empty;
```

The following approaches and the parallel classification rule induction algorithm presented in this paper are explained in the context of PrismTCS. However, our approaches can be applied to Prism and all its descendants analogously.

2.1 Speeding up PrismTCS

We identified two major overheads in Prism and PrismTCS that lower its computational efficiency considerably. The overheads comprise sorting for continuous

attributes in step b of the algorithm and the frequent deletion of data instances and resetting of the training dataset in step a, e and g of the algorithm. With respect to the sorting overhead we removed the innermost loop of sorting in step b by employing a pre-sorting strategy. The training dataset is pre-sorted by building attribute lists of the structure <*record id, attribute value, class value*> similar to the SPRINT algorithm [10, 17]. These attribute lists can be sorted before the first iteration of the Prism algorithm and remain sorted during the whole duration of the algorithm. With respect to frequent restoring of the training dataset an efficient data compression algorithm has been developed. When we talk about compression we mean an efficient way to delete and restore instances of the training dataset, while maintaining the sorted nature of the attributes, which is needed frequently in the algorithms of the Prism family. For example regarding the PrismTCS pseudo code, data instances are deleted in step e and g and restored in step a. The challenge here is to find a way of data efficient compression that takes account of the pre-sorted attribute lists. One way to implement this would be to keep two copies of each attribute list in memory, one list for resetting purposes and one list to work with, analogously to the "*working dataset W*" and the "*restore Dataset*" in the PrismTCS pseudo code in section 2. Attribute lists would be restored by replacing the working set of lists with a restore set of lists. However this approach involves a considerable overhead of memory usage by keeping two copies of the whole training dataset in the form of attribute lists in memory. A further overhead in processing time is caused by frequently creating deep copies of attribute list records.

We derived a more memory and time efficient algorithm for deleting and restoring data which involves having the dataset only stored once in the memory. We do that by only working with the *record ids* of each attribute list record which are stored in an integer array. This array is used to reference *attribute values* and *class values* which are stored in separate double precision and character arrays. Thus when pre-sorting the attribute list we only need to sort the integer array with record ids. Also when deleting list records we only need to delete the references in the *record ids* array. Thus the attribute values array and class values array are left untouched during the whole duration of the Prism algorithm. However we also need to avoid expensive resizing of the record ids array due to deletion and resetting of ids. We do that by replacing ids that need to be deleted in the array by the next id that does not need to be deleted and thus the size of the actual array stays the same. If the number of ids that are not deleted is *n* then PrismTCS will only take record ids stored between indices *0* and *n-1* into account and ignore the rest. Thus PrismTCS is required to update n whenever ids are deleted or reset.

The pseudo code below shows the basic compression algorithm:

```
int numbRelevant;// number of relevant ids in the array
boolean[] remove;// each index in the array corresponds
                 //to a actual id value that needs to be
                 //deleted
removal(numbRelevant, remove){
  int i,j;
  j = 0;
```

```
  FOR(i=0; i<numRelevant; i++){
     IF(remove[i]){
       recordIdsArray[j] = recordIdsArray[i]
       j++;
     }
   }
   numRelevant = j;
}
```

This algorithm is implemented in steps e and g of the PrismTCS pseudo code. We implemented three versions of PrismTCS, the original algorithm, as described in section 2, PrismTCS with attribute lists and PrismTCS with attribute lists and data compression.

There are three general factors that determine the time taken to process a dataset using PrismTCS or any other algorithm that generates a model from data:

(1) the volume of data (number of instances and attributes)
(2) the attribute types (continuous attributes generally take much longer to process than categorical ones)
(3) the complexity of the model produced in the chosen representation (in the case of PrismTCS this is the number of Disjunctive Normal Form rules and the number of terms they comprise).

The complexity of the model can vary considerably from one dataset to another, depending on the appropriateness of the chosen representation for representing the underlying causality of the domain. Although in the case of PrismTCS large datasets tend to lead to larger rulesets this is not necessarily or always the case. It is entirely possible for a very large dataset to be modelled by a small number of rules or (in an extreme case) for a small dataset to require as many rules as there are instances. It depends on the suitability of the representation. In all our experiments we have fixed factors (2) and (3) to enable us to focus on the effect of factor (1). All the attributes are continuous and all the rulesets derived from the yeast dataset from the UCI repository [18] lead to precisely the same model being output. We increased the size of the yeast data by appending it to itself, thus ranging from 1,500 up to 23,000 data instances, where each instance comprises 8 attributes. As illustrated in figure 1, pre-sorting has a positive impact on the scaling behaviour of PrismTCS, However both PrismTCS with and without pre-sorting are scaling with second order polynomial behaviour. On the other hand PrismTCS with pre-sorting and our data compression strategy scales up with a linear behaviour.

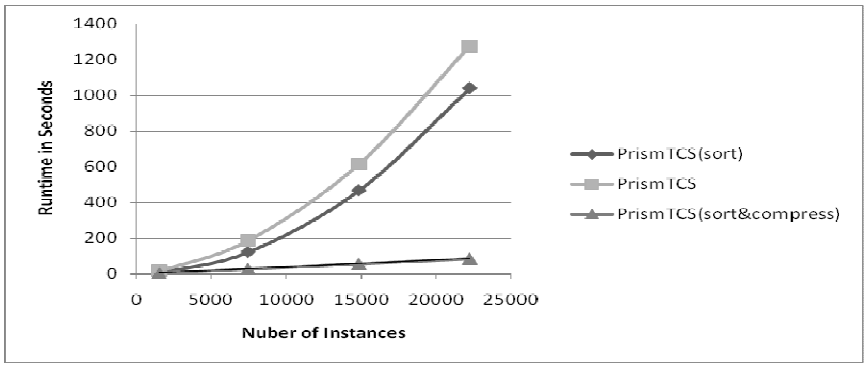

Fig. 1. Scale up of PrismTCS.

We were able to scale up our version of PrismTCS to a far larger number of attributes and data instances as shown in section 4. The following regression equations represent the scale ups of all three versions of PrismTCS where *x* is the number of training instances and *y* is the runtime in ms:

PrismTCS: $y = 0.002x^2 + 9.036x \ (R^2 = 1)$
PrismTCS(sort): $y = 0.002x^2 + 1.158x \ (R^2 = 1)$
PrismTCS(sort&compress): $y = 3.620x \ (R^2 = 0.998)$

3. PMCRI: A Parallel Modular Classification Rule Induction Framework

There have been several attempts to scale up classification rule induction via parallelisation. In the area of TDIDT we have already mentioned the SPRINT [10] algorithm. We can distinguish two types of parallel processing in data mining: fine grained parallelisation and loosely coupled distributed data mining [19]. Whereas fine grained parallelisation makes use of "shared memory multiprocessor" machines (SMP), loosely coupled distributed data mining makes use of "shared nothing" or "massively parallel processors" (MPP) [20]. We will focus here on parallelising modular classification rule induction using an MPP approach. Our reasoning is that MPP can be represented by a network of workstations and thus is a cheap way of running parallel algorithms for organisations with limited budgets. We want to use an MPP infrastructure to parallelise the modular classification rule induction of Prism by using the Cooperating Data mining Model (CDM) [19]. The CDM model is illustrated in figure 2. It is partitioned into different sections which we call *layers*. In the first layer of the CDM model is the sample selection procedure which partitions the data sample *S* into *n* sub samples where *n* is the number of workstations available. There are n Learning algorithms $L_1,\ldots,L_n$ in the second layer that run on the corresponding subsets and generate concept descriptions. C_1, …. C_n. In the third layer these concept descriptions are then merged by a combining procedure to a final concept description C_{final}. The final concept description in the case of classification rule induction would be a set of classification rules.

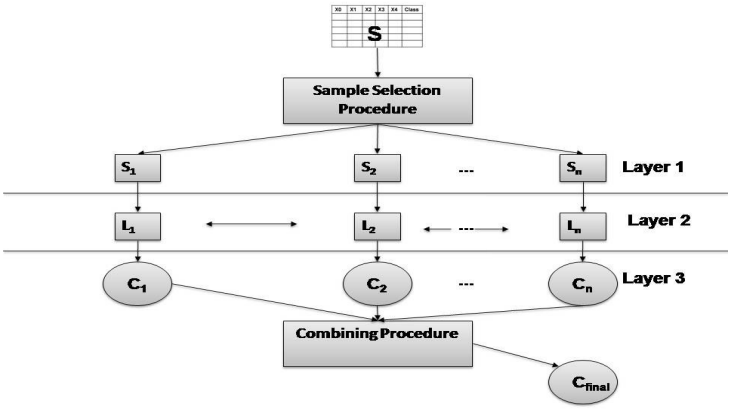

Fig. 2. Cooperating Data Mining.

We developed a parallel modular classification rule induction framework for the Prism family and tested it on PrismTCS, the PMCRI (Parallel Modular Classification Rule Induction) framework [21], which applies to the CDM model. Parallelisation in the first layer is achieved by distributing all attribute lists evenly over n processors and processing them locally by algorithms L_1 to L_n, which induce rule terms. To implement the second layer in the CDM model we used a distributed blackboard system architecture based on the DARBS distributed blackboard system [22].

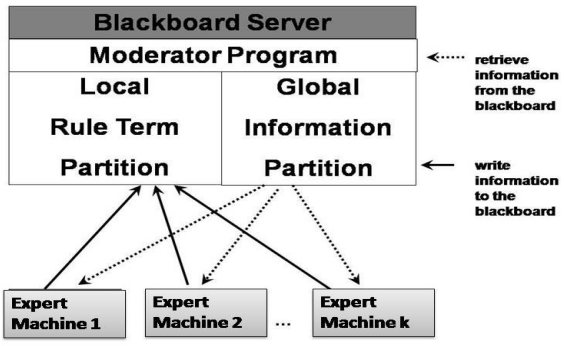

Fig. 3. The architecture of the PMCRI framework using a distributed blackboard system in order to parallelise the induction of modular rule terms. It is divided into two logical partitions: one to submit local rule term information and one to retrieve global information about the algorithm's status.

A blackboard system can be imagined as a physical blackboard which is observed by several experts with different knowledge domains, having a common problem to

solve. Each expert will use its knowledge domain plus knowledge written on the blackboard in order to infer new knowledge about the problem and advertise it to the other experts by writing it on the blackboard. In the software model such a blackboard system can be represented by a client server architecture. Figure 3 shows the basic communication pattern of PMCRI. The expertise of each expert machine is determined by the attribute lists it holds. Thus loosely speaking, each expert can induce the "locally best rule term". The experts then use then the "Local Rule Term Partition" on the blackboard to exchange information in order to find the "globally best rule term". The winning expert then will communicate the ids of the instances that are uncovered by this rule term to the other waiting experts using the "Global Information Partition" on the blackboard system. Now the next rule term can be induced in the same way. In PMCRI the attribute lists decrease in size at the same rate, thus the workload on each expert machine stays in the same proportion for all remaining experts.

The following steps describe how PMCRI induces one rule in the context of PrismTCS[23]:

```
Step 1 Moderator (PrismTCS) writes on "Global
    Information Partition" the command to induce locally
    best rule terms.

Step 2 All Experts induce the locally best rule term and
    write the rule terms plus its covering probability
    and the number of list records covered on the "local
    Rule Term Partition"

Step 3 Moderator (PrismTCS) compares all rule terms
    written on the "Local Rule Term Partition"; adds
    best term to the current rule; writes the name of
    the Expert that induced the best rule term on the
    Global Information Partition

Step 4 Expert retrieves name of winning expert.
    IF Expert is winning expert {
       derive by last induced rule term uncovered ids
       and write them on the "Global Information
       Partition" and delete uncovered list records
    }
    ELSE IF Expert is not winning expert {
        wait for by best rule term uncovered ids being
        available on the "Global Information Partition",
        download them and delete list records matching
        the retrieved ids.
    }
```

In order to induce the next rule term, PMCRI would loop back to step one. For PMCRI to know when to stop the rule it needs to know when the remaining list records on the expert machines are either empty or consist only of instances of the current target class. This information is communicated between the winning expert and the moderator program using the Global Information Partition. It is possible to

implement all the descendants of the original Prism algorithm simply by adapting the learner algorithm within this framework.

In layer 3 at the end of the PMCRI execution each expert machine will hold a set of terms for each rule. The implementation of the combining procedure in layer three in the CDM model is realised by communicating all the rule terms locally stored at the expert machines to the blackboard.

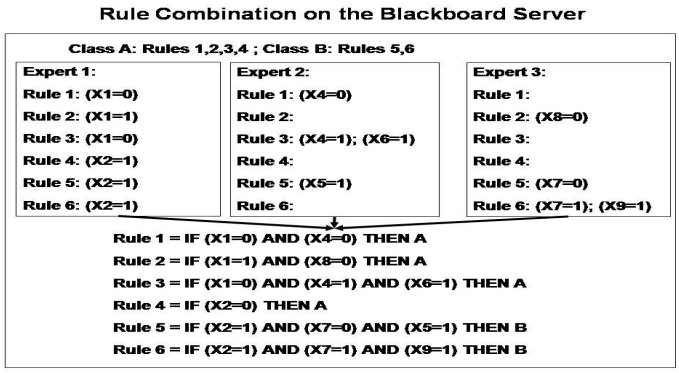

Fig. 4. The combining procedure of the CDM is realised by the moderator program, which will assemble the complete rules after each expert submits its globally best rule terms together with information about the rule for which they were induced.

Each rule term is associated with information about the rule and the class for which the terms were induced. The moderator program then simply appends each rule term to its corresponding rule as illustrated in figure 4.

4. Performance of PMCRI

To evaluate PMCRI we used the yeast dataset from the UCI repository [18]. To create a larger dataset and thus a higher (more challenging) workload for our system, we first appended it to itself in a horizontal direction until the dataset comprised a total number of 50 attributes. We used this base dataset to evaluate the system's performance. We then appended the data to itself in a vertical direction in order to increase the number of instances and thus increase the system's workload. Please note that the learner algorithms of PMCRI are based on PrismTCS and produce exactly the same rules as PrismTCS would induce. There is therefore no need to concern ourselves with issues concerning the comparative quality of rules generated by the different algorithms. Also please note that all datasets used were based on the yeast dataset, thus the classifiers induced in each experiment were identical and issues relating to differing numbers of rules and rule terms do not arise. This enables us to focus on issues of workload only. We made scale up experiments to evaluate PMCRI's performance with respect to its number of processors, speed up experiments to evaluate its performance with respect to the number of processors together with the processors' workload and size up experiments in order to evaluate its performance concerning the system's total workload. The hardware we used comprised ten identical

machines where each machine had one Pentium processor with 2.8 GHz and one GB memory. We run lightweight xUbuntu Linux Systems on each machine.

4.1 Scale up

Scale up is used to observe the system's ability to be enlarged. For examining the scale up of PMCRI we observe how the response time changes if the number of processors is increased while the workload per machine stays constant. In the ideal case the response time of PMCRI would stay constant since the total workload of each processor stays the same.

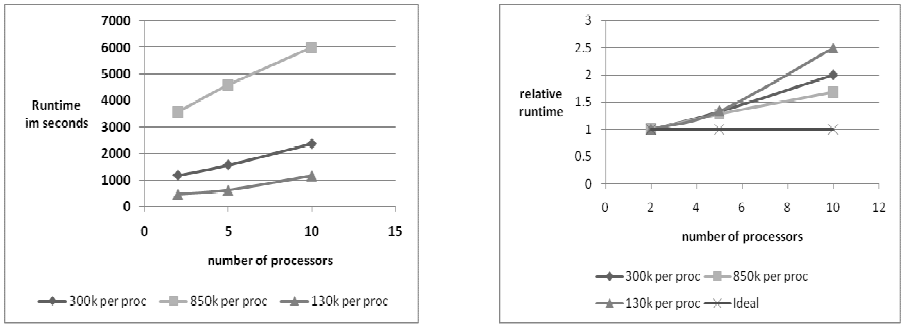

Fig. 5. Scale up of PMCRI

We studied three scale up experiments with a workload corresponding to 130,000, 300,000 and 850,000 instances per processor. Please note due to the distribution of the data via attribute lists each processor only holds a part of each data instance, for example the 130k instances workload for each processor in the case of two processors are actually 260k instances with half the attributes of each instance. In order to simplify matters we just refer to 130k training instances. The results of the scale up experiments are shown in figure 5. The results show a nice scale up. The drop in scale up with adding more processors can be explained by the additional communication overhead in the LAN, as more processors need to synchronise by communicating information about covered and uncovered list ids, while the amount of data per processor stays constant, the number of ids that need to be communicated increases with an increasing number of total instances. However as figure 5 also shows the effect of communication overhead can be lowered by increasing workload per processor. Loosely speaking the higher the overall workload the more the system profits from using additional processors.

4.2 Speed up

Speed up is used to compare how much a parallel algorithm is faster than the serial version of it. However we are limited with workload of the serial version of PrismTCS by the size of the memory of the computer used. However as can be seen in section 4.3 all parallel versions of PrismTCS using PMCRI are faster than the

serial version thus an absolute speed up compared with the serial PrismTCS would be positive. However we are able to determine the relative speed up of PMCRI in the context of PrismTCS by basing it to a two processor configuration. By examining the speed up characteristics of PMCRI we observe how the response time changes with the number of processors while the total workload stays constant. We studied three speed up experiments with a total number 600k, 1,000k and 2,000k data instances for configurations of 2, 5 and 10 processors. The results of the speedup experiments are shown in figure 6.

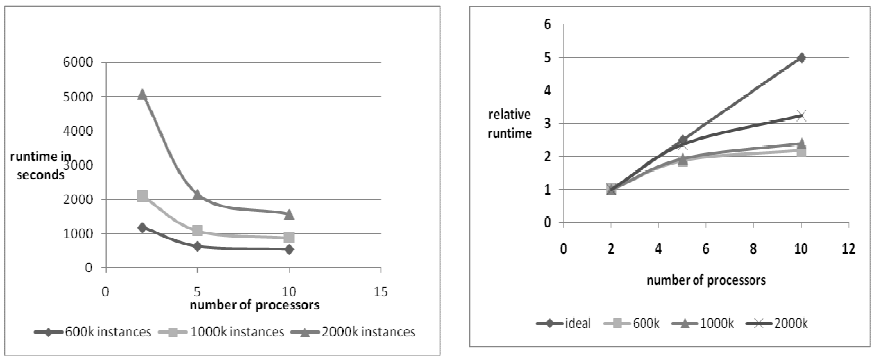

Fig. 6. Speed up of PMCRI

As we can expect the speedup increases with the size of the dataset. The ideal speedup would be if the amount of processors is doubled then the response time is reduced by half. However this behaviour is contradicted for the same reason as the scale up behaviour, by a communication overhead. Especially for small datasets, the additional communication overhead contradicts the benefit of more processors considerably, but for larger datasets the benefit from having more CPU power strongly outweighs the communication overhead. Again we observe that PrismTCS parallelised using PMCRI is faster the more processors we use especially for higher data workloads.

4.3 Size up

In size up experiments we examine how PMCRI performs on a fixed processor configuration. We do that by increasing the size of the data and leaving the number of processors constant. Figure 7 shows the size up for three different processor configurations of PMCRI and our serial version of PrismTCS. We increased the dataset size from 17k up to 8,000k training instances.

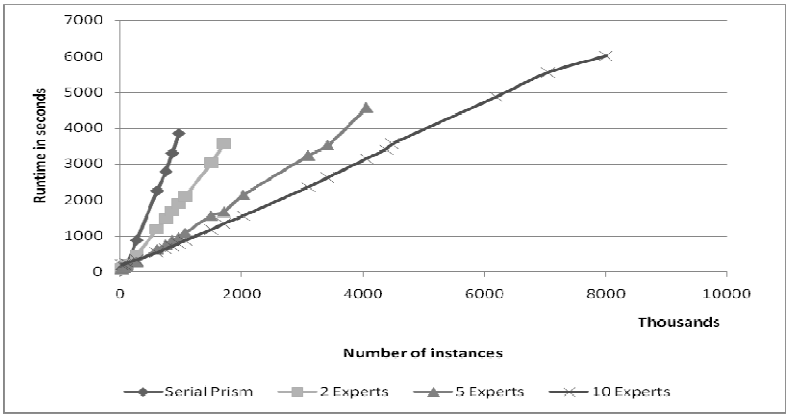

Fig. 7. Size up of PMCRI

Generally we observe a linear size up for PrismTCS and all PMCRI configurations, thus the processing time is a linear function of' the size of the dataset. The equations of a linear regression that prove the linear behaviour are shown below where *x* is the number of training instances and *y* the runtime in ms:

PrismTCS: $y = 3.840x$ ($R^2 = 0.994$)
PMCRI with 2 processors: $y = 2.019x$ ($R^2 = 0.997$)
PMCRI with 5 processors: $y = 1.065x$ ($R^2 = 0.995$)
PMCRI with 10 processors: $y = 0.775x$ ($R^2 = 0.997$)

Please note that all experiments from figure 7 were sized up to their maximum number of training instances. The maximum number of data instances is limited by the total amount of memory in the system. Buffering of attribute lists to the hard disc would be possible in order to overcome memory limitations; however frequent I/O operations are unwanted due to their time expensive behaviour.

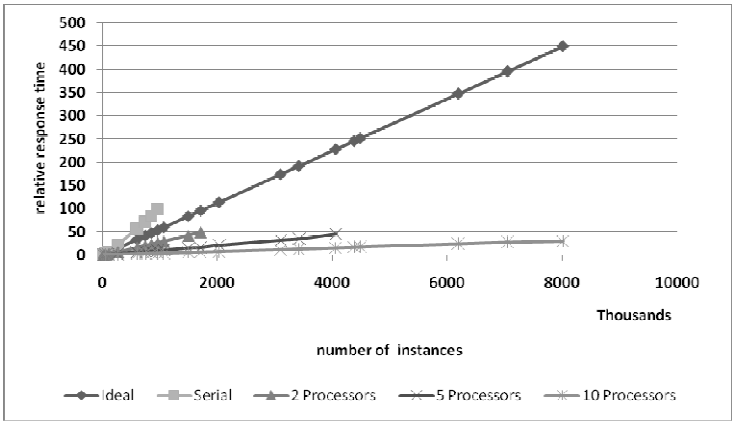

Fig. 8. Size up using relative response time.

We can clearly see that if we use double the amount of memory we can hold roughly double the amount of training instances into memory. Figure 8 shows the

size up using the relative response time for the same configurations as in figure 7, however this time we also added the ideal size up into the figure. The ideal size up would be that if we have double the amount of training instances on the same number of processors then we will need double the amount of time to train our classifier. We can observe that the serial version of PrismTCS clearly scales worse than the ideal runtime however for all parallel versions we observe a better sizing up behaviour than ideal, in particular the more processors we use the better the size up result compared with the ideal size up behaviour. Thus we can say PMCRI shows a superior size up performance.

5. Conclusions

We presented the work and first results of the PMCRI framework, a Parallel Modular Classification Rule Induction algorithm. PMCRI harvests the computational power of a network in order to make modular classification rule induction scaling better on large datasets. We started the paper by discussing the limitations of the more popular TDIDT algorithm and then we discussed the Prism algorithm family as an alternative that tries to overcome TDIDT's limitations naturally. However Prism's downside is that it does not scale well on large datasets. We discovered the frequent deletion of and restoring of data records and frequent sorting operations for continuous attributes as bottlenecks of Prism. The latter bottleneck has been tackled by using attribute lists in order to keep the training data sorted during the whole duration of the algorithm. We addressed the bottleneck of the frequent deletion and restoring of data records by proposing an algorithm that efficiently deletes attribute list instances without having to resize the underlying data structure. We achieve a linear scaling behaviour of PrismTCS by implementing these two approaches. We then described our work on a parallel classification rule induction algorithm based on the same rule generalisation method as PrismTCS, the PMCRI algorithm. The parallelisation in PMCRI is achieved by distributing attribute lists evenly over the machines in the network. Each machine induced rule terms independently that are locally the best. A global view of the algorithm is achieved by each machine by exchanging information about the local status of each machine using a distributed blackboard architecture. As both PMCRI and PrismTCS employ the same rule generalisation strategy the rule sets produced by both on the same training data are identical. We then experimentally analysed PMCRI's performance. With respect to its scale up behaviour we observed that the more processors we use the higher the synchronisation overhead due to communicating local information between processors. However the synchronisation overhead is contradicted by the processors' workload, thus the higher the workload the closer PMCRI scales up to its ideal behaviour. With respect to PMCRI speed up we again observed a speed up below its ideal speed up, which again can be explained by a communication overhead between processors, however we could observe that the higher the workload the closer the actual speed up performance is to its ideal. With respect to PMCRI size up we observed a linear behaviour on a fixed processor configuration with respect to its workload. Generally for PMCRI we observed superior size up behaviour to its ideal. We also stretched the boundaries in order to find the maximum workload of the system. The maximum workload was limited by the total

memory available thus the more machines we used the more data we were able to use in order to train our classifier. However we can generally say that the benefit of using more processors in PMCRI slows down due to an additional communication overhead per processor. However the communication overhead can be outweighed by adjusting the total workload of the system. Thus, loosely speaking, the user of the PMCRI system should balance the number of processors used with the total workload, thus for a smaller workload a configuration with less processors or even the serial version might be more beneficial concerning the system's runtime. However for large workloads a configuration with more processors is more likely to be beneficial. We are currently developing a more intelligent workload balancing strategy for PMCRI that takes into account that computers might have different CPU speeds and memory sizes available.

References

1. Hunt E. B., Marin J., and Stone P. J., *Experiments in Induction*. 1966: Academic Press.
2. Quinlan J. R., *Induction of decision trees. Machine Learning*. Vol. 1. 1986. 81-106.
3. Cendrowska J., *PRISM: an Algorithm for Inducing Modular Rules.* International Journal of Man-Machine Studies, 1987. **27**: p. 349-370.
4. Shu-Ching C., Mei-Ling S., and Schengcui Z., *"Detection of Soccer Goal Shots Using Joint Multimedia Features and classification Rules*, in *Fourth International Workshop on Multimedia Data Mining*. 2003: Washington DC, USA. p. 36-44.
5. Bramer M., *An Information-Theoretic Approach to the Pre-pruning of Classification Rules*. Proceedings of the IFIP Seventeenth World Computer Congress - TC12 Stream on Intelligent Information Processing. 2002: Kluwer, B.V. 201-212.
6. Bramer M., *Automatic Induction of Classification Rules from Examples Using N-Prism.* Research and Development in Intelligent Systems XVI, 2000.
7. Garner S., *Weka: The Waikato Environment for Knowledge Analysis*, in *New Zealand Computer Science Research Students Conference*. 1995. p. 57-64.
8. Bramer M., *Inducer: a public domain workbench for data mining.* International Journal of Systems Science, 2005. **36**(14): p. 909-919.
9. Metha M., Agrawal R., and Rissanen J., *SLIQ: A Fast Fcalable Classier for Data Mining*. International Conference on Extending Database Technology EDBT'96), 1996.
10. Shafer J. C., Agrawal R., and Mehta M., *SPRINT: A Scalable Parallel Classifier for Data Mining.* Twenty-second International Conference on Very Large Data Bases, 1996.
11. Catlett J., *Megainduction: Machine learning on very large databases*. 1991, University of Technology, Sydney.
12. Frey L. J. and Fisher D. H, *Modelling Decision Tree Performance with the Power Law.* eventh International Workshop on Artificial Intelligence and Statistics, 1999.
13. Provost F., Jensen D., and Oates T., *Efficient Progressive Sampling*, in *Knowledge Discovery and Data Mining*, Geoffrey I., Editor. 1999. p. 23-32.

14. Chan P. K. and Stolfo S.J., *Experiments on Multistrategy Learning by Meta Learning*, in *Second International Conference on Information and Knowledge Management*. 1993. p. 314-323.
15. Chan P. K. and Stolfo S.J., *Meta-Learning for Multistrategy and Parallel Learning*, in *Second International Workshop on Multistrategy Learning*. 1993. p. 150-165.
16. Michalski R.S., *On the quasi-minimal solution of the general covering problem*, in *Proceedings of the Fifth International Symposium on Information Processing*. 1969: Bled, Yugoslavia. p. 125-128.
17. Zaki M. J., Ho C.T., and Agrawal R., *Parallel Classification for Data Mining on Shared Memory Multiprocessors.* Fifteenth International conference on Data Mining, 1999.
18. Blake C. L. and Merz C. J, *UCI repository of machine learning databases*. 1998, University of California, Irvine, Department of Information and Computer Sciences.
19. Provost F., *Distributed Data Mining: Scaling up and Beyond*, in *Advances in Distributed and Parallel Knowledge Discovery*, P.C. H. Kargupta, Editor. 2000, AAAI Press / The MIT Press.
20. Kamath C. and Musik R., *Scalable Data Mining through Fine-Grained Parallelism*, in *Advances in Distributed and Parallel Knowledge Discovery*, P.C. H. Kargupta, Editor. 2000, AAAI Press / The MIT Press.
21. Stahl F. and Bramer M., *P-Prism: A Computationally Efficient Approach to Scaling up Classification Rule Induction*, in *IFIP International Conference on Artificial Intelligence*. 2008, Springer: Milan.
22. Nolle L., Wong K. C. P., and Hopgood A., *DARBS: A Distributed Blackboard System.* Twenty-first SGES International Conference on Knowledge Based Systems, 2001.
23. Stahl F. and Bramer M., *Parallel Induction of Modular Classification Rules*, in *Twenty-eighth SGAI International Conference on Innovative Techniques and Applications of Artificial Intelligence*. 2008, Springer: Cambridge.