*The Knowledge Engineering Review*, Vol. 00:0, 1–24. © 2004, Cambridge University Press
DOI: 10.1017/S000000000000000 Printed in the United Kingdom

# Scaling Up Classification Rule Induction Through Parallel Processing

FREDERIC STAHL, MAX BRAMER

*University of Portsmouth, School of Computing, Buckingham Building, Lion Terrace, PO1 3HE Portsmouth, UK*
*E-mail: Frederic.Stahl@port.ac.uk, Max.Bramer@port.ac.uk*

**Abstract**

The fast increase in the size and number of databases demands data mining approaches that are scalable to large amounts of data. This has led to the exploration of parallel computing technologies in order to perform data mining tasks concurrently using several processors. Parallelisation seems to be a natural and cost effective way to scale up data mining technologies. One of the most important of these data mining technologies is the classification of newly recorded data. This paper surveys advances in parallelisation in the field of classification rule induction.

## 1 Introduction

There are many areas confronted with the problem of mining massive datasets. For example, in bioinformatics and chemistry there are large datasets which are generated in different kinds of experiments and simulations, and there is considerable desire to find ways to manage, store and find complex relationships in this generated data (Berrar et al., 2005). Examples for such large scale simulations are Molecular Dynamics simulations that are conducted in order to find rules to describe the folding, unfolding or malfolding of proteins. Researchers in this area are just beginning to be able to manage and store these massive amounts of data (Stahl et al., 2005), which can reach 100s of gigabytes for a single simulation. Another example for massive datasets is the data that is generated by the NASA System of earth orbiting satellites and other space-borne probes (Way & Smith, 1991) launched in 1991 and still ongoing. This NASA system sends back approximately one terrabyte of data a day to receiving stations. In the business area, major multi-national corporations record customer transactions, and they often have thousands of establishments around the world that are all collecting data and store it in centralised data warehouses. In astronomy there exist databases that comprise terrabytes of image data and they are increasing in size as further data is collected for the GSC-II (McClean et al., 1998) and the Sloan survey (Szalay, 1998). Furthermore, the data sample size is not only dependent on the number of instances which could be sampled but also on the number of attributes and therefore, for example, the size of gene expression data is strongly dependent on the number of genes observed in the experiment. The number of attributes, for data generated in gene expression experiments, can reach tens of thousands for human genomes. In the case of 'market basket analysis', the number of attribute lists is strongly dependent on the number of products a store sells which can easily reach tens of thousands.

Not only the sheer size but also where and how this data is stored is problematic. For example in the area of molecular dynamics simulations, researchers keep their simulation data in in-house databases but also want to share their simulation data with other research groups, that may also store their locally generated data in their own local databases. This geographically distributed data also needs to be mined. Problems that arise there are the bandwidth consumption

of transferring vast amounts of data via the Internet and the fact that the database schemas might be customised to the individual research group that generated the data.

Loosely speaking, the commercial and the scientific world are confronted with increasingly large and complex datasets to be mined, and therefore tools and techniques that are scalable are required.

The problem of geographically distributed data sources that need to be mined is also known as *Distributed Data Mining (DDM)*. An overview of Distributed Data Mining can be found in (Park & Kargupta, 2002). Also ensemble learners are naturally suited for learning from distributed data sources. Park's survey also briefly outlines the Collective Data Mining framework for predictive modelling from heterogeneous data sources (Kargupta, Byung-Hoon, Hershberger, & Johnson, 1999). Another framework that is concerned with learning from distributed heterogeneous data sources is described by Caragea (Caragea, Silvescu, & Honavar, 2003) and has successfully been demonstrated for classification rule induction in the form of decision trees. Also the DataMiningGrid.org project is concerned with all sorts of data mining applications on geographically distributed data sources (Stankovski et al., 2008). One major concern of the DataMiningGrid.org project is the transfer of individual data mining algorithms to data sources, rather than transferring the data itself, with the aim to avoid a too intense bandwidth consumption.

However the scope of this survey is not concerned with geographically distributed or remotely located data sources but the scalability of the data mining task itself with the focus on classification, in particular classification rule induction. By scalable we refer to the ability to train a classifier on large training data and still generate the classification rules in a reasonable amount of time. Additional computational resources in the form of multiprocessor architectures can be used to mine increasingly large datasets.

The rest of the survey is organised as follows. Section 2 will discuss some general approaches to parallel data mining and classification rule induction. Section 3 will discuss data reduction as an approach to scaling up classification rule induction. Section 4 will introduce some general distributed data mining models and Section 5 will discuss concrete parallel formulations of classification rule induction algorithms based on decision trees. Section 6 highlights a recently developed methodology to parallelise modular classification rule induction algorithms, an alternative to decision trees. Section 7 will finish this survey with a brief summary and some concluding remarks.

## 2 Parallel Data Mining

Multiprocessor architectures can be used to host data mining algorithms. In general the workload is distributed to several computing nodes by assigning different portions of the data to different processors. Multiprocessor computer architectures can be divided into *tightly-coupled* and *loosely-coupled* architectures.

- A *tightly-coupled* architecture is constructed of multiple processors sharing one common memory, and physical data distribution is not required. Implementations of tightly-coupled architectures are also often referred to as *'Shared memory Multiprocessor machines'* (SMP).
- A *loosely-coupled* architecture is a collection of multiple computers in different locations. Each computer and thus each processor has its local private memory. Here data distribution is required along with data communication and accumulation mechanisms. Implementations of loosely-coupled architectures are also often referred to as shared nothing or *'Massively Parallel Processors'* (MPP).

The effectiveness of the data distribution mechanism is dependent on the architecture of the underlying network, also called the *host system*. Figure 1 illustrates both types of multiprocessor computer architectures. In the tightly-coupled architecture data is communicated using a memory bus system, whereas the loosely-coupled system utilises a communication network. Unfortunately,

as the number of processors increases in a SMP system where the processors share the bus, the bandwidth available per processor decreases. This can be overcome by using a MPP system. The advantage of loosely-coupled architectures is that the application components of the system can be hosted on different hardware. Hosting application components on different computers across a network makes the application more robust so that computer failure does not bring down the entire application. The disadvantage of loosely-coupled systems is that the application requires communication and collaboration between components. The communication requirement introduces an additional overhead for the application. Compared with loosely-coupled systems tightly-coupled systems are usually more efficient at processing as they avoid data replication and transfer of information between application components. However, the main disadvantage of tightly-coupled systems is often seen in the high cost of hardware to scale tightly-coupled architectures. An advantage of a loosely-coupled system compared with a tightly-coupled system is that the processing components can be reused as independent processing units or workstations once the multiprocessor system is not needed anymore. A further advantage of loosely-coupled systems is that if they are hosted on a network (that comprises independent workstations), the hardware used in the system is naturally upgraded as old computers are replaced by new ones. Upgrading a tightly-coupled system usually requires replacing the entire hardware.

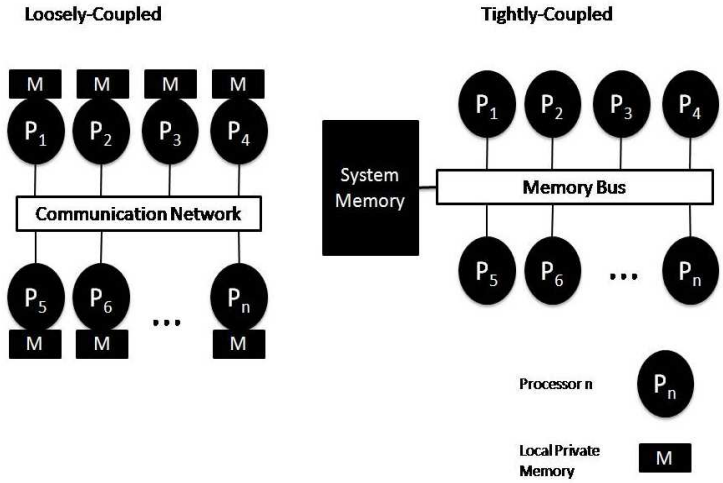



**Figure 1** Tightly- and loosely-coupled multiprocessor computer architectures

However, research in the area of classification rule induction has just begun to utilise the advantages of both loosely-coupled and tightly-coupled multiprocessor architectures. Due to the fast increase of data repositories and the massive size of single datasets the data mining community is called on to investigate the advantages of multiprocessor architectures, their suitability and benefits to scale up data mining systems to large data volumes.

The rest of this Section reviews two principal forms of parallelism often used to parallelise data mining tasks and algorithms, *task* and *data parallelism*. In *task parallel* algorithms the actual code is executed concurrently on several computing nodes or processors. In the data mining literature task parallelism is also often referred to as *control parallelism* (Freitas, 1998). In contrast to *task parallelism, data parallelism* relates to the concurrent execution of the same task on different subsets of the dataset on several computing nodes or processors (Hillis & Steele, 1986). The latter form of parallelism is very popular as it naturally achieves parallelisation of the workload of most data mining applications, as the workload of data mining applications is directly related to the amount of data used to mine. Data parallel data mining algorithms are sometimes referred to as 'distributed data mining' algorithms, because of the fact that the data is distributed to several computing nodes. This is confusing as 'distributed data mining' also refers

to the mining of geographically distributed data sources which is in contrast to data parallel data mining algorithms not necessarily concerned with achieving a shorter execution time due to parallelisation. However this survey reviews parallel classification rule induction algorithms in the context of achieving shorter runtime using multiple processors.

## 3 Scaling up Data Mining: Data Reduction

Data reduction is usually applied before the actual data mining algorithm and is therefore a preprocessing step of data mining. Data Reduction in the context of this work is aimed at reducing the overall data size in order to reduce the runtime of data mining algorithms. There are two major techniques of data reduction applied in this context; *feature selection*, which is also often called *attribute selection* and *sampling*.

### *3.1 Feature Selection*

Feature selection strategies that can be applied for classification tasks are described in this Section. A frequently used metric to evaluate attributes for relevance for the classification task is information gain, based on Shannon's entropy (Shannon, 1948). The basic procedure is to use the information gain to calculate how much information can be gained about a classification if the value of a certain attribute is known. Only the collection of attributes that have the largest information gains are used to run the classification rule induction algorithm. There are three main steps in order to filter out the relevant attributes:

1. Calculation of the information gains of all attributes
2. Deleting the attributes with the lowest information gains
3. Use the reduced data set in order to induce classification rules

According to Bramer (Bramer, 2007), there are many strategies that can be used at step 2, for example:

- Keep the $x$ attributes with the highest information gain.
- Keep the $x\%$ of attributes with the highest gain.
- Keep only those attributes that have an information gain of $x\%$ or more of the highest information gain of any attribute in the dataset.
- Keep only those attributes that reduce the initial entropy of the dataset by at least $x\%$.

Han and Kamber (Han & Kamber, 2001) describe similar approaches that can be used to select the best $x$ attributes. They do not suggest a specific metric, but the information gain could be used with it:

- *Stepwise forward selection*: Firstly a working dataset is built which is initially empty. The best attribute from the original dataset is determined, added to the empty dataset and deleted from the original one. In every subsequent iteration again the best of the remaining attributes are transferred from the original to the working dataset. This method implies that a threshold is defined of how many attributes the working dataset should hold.
- *Stepwise backward elimination*: This procedure iteratively calculates the goodness of all attributes and deletes the worst attributes in each step from the dataset. Again, this method implies that a threshold is defined of how many attributes should be removed.
- *Combination of forward and backward elimination*: A combination of the two latter methods may select the best attribute in each step and delete the worst one. This method implies that a threshold is defined of how many attributes should be removed and kept.
- *Decision tree induction*: A decision tree is induced on the original dataset using an algorithm such as ID3 (Quinlan, 1979a) or C4.5 (Quinlan, 1993). Here it is assumed that attributes that do not appear in the decision tree are irrelevant and so are not used for the data mining

algorithm. For the purpose of scaling up classification rule induction, this method would not be suitable due to the processing time needed to induce a decision tree just for the feature selection process.

A different approach to reduce the number of features is the Principal Component Analysis (PCA). Each instance can be described as a vector of $m$ features of dimensions. PCA searches for $k$ m-dimensional orthogonal vectors ($k \leq m$) that can be used to present the training data. The difference compared with feature selection described above is that PCA creates a new smaller set of features on which the initial data can be projected, whereas feature selection selects a subset of the original features. Implementations of PCA can be found in many statistical software kits such as (*Minitab*, 2010; *SAS/STAT*, 2010).

### *3.2 Sampling*

Sampling is the representation of a large dataset by a smaller random sample; a subset of the data. Assuming that dataset $D$ contains $N$ examples, then there are some general strategies for building the sample. For example:

- *Simple random sample without replacement (SRSWOR)* of size $s$: Here $s$ examples are randomly drawn from $D$.
- *Simple random sample with replacement (SRSWR)* of size $s$: Similar to SRSWOR, except that drawn examples are not deleted from $D$, so may be drawn again.

The difficulty in sampling is to determine the right sample size. There are several algorithms that attempt to converge the sample size towards a size that achieves a good classification accuracy. For example Quinlan's *Windowing* algorithm attempts to achieve a good sample size for the ID3 decision tree induction algorithm (Quinlan, 1983). Windowing initially takes a random sample, the *window*, from the dataset. The initial size of the window is specified by the user. The window is used to induce a classifier. The induced classifier is then applied to the remaining instances. Instances that are misclassified are added to the window. The user can also specify the maximum number of instances to add to the window. Again a classifier is induced using the new window and tested on the remaining instances. Windowing also applies testing first to instances that have not been tested yet and then to the already tested ones. This is repeated until all remaining instances are correctly classified. Windowing has been examined empirically in (Wirth & Catlett, 1988), where windowing did not perform well on noisy datasets. Noisy data is when there is irrelevant information in the data. However a further development of windowing has been developed for the C4.5 algorithm (Quinlan, 1993). The essential difference from the initial windowing algorithm is that it aims to achieve as uniformly distributed a window as possible. It also takes at least half of the misclassified instances in the new window and it may stop, even though not all instances are classified correctly, in cases where no higher accuracy is achievable. According to (Catlett, 1991) the fact that windowing in C4.5 aims for a uniformly distributed window can improve accuracy gains in the case of skewed distributed datasets. In general windowing imposes an additional computational overhead as many runs of the classification rule induction algorithm have to be learned in order to determine the optimal window or sample to train the classifier on.

*Integrative Windowing* (Fuernkranz, 1998) addresses this problem. Integrative windowing builds the initial window as in Quinlan's windowing algorithm, but does not only add misclassified instances to the window but also deletes instances that are covered by consistent rules. Consistent rules in Integrative Windowing are rules that did not misclassify negative examples during testing. Consistent rules are remembered but have to be tested again in the next iteration, as they may not have been tested on all remaining instances due to the possibility that the maximum number of instances that can be added to the window was reached in the previous iteration. (Fuernkranz, 1998) implemented Integrative Windowing for 'separate and conquer' algorithms only.

A further technique developed for finding a good sample size is *progressive sampling* (Provost, Jensen, & Oates, 1999). Progressive sampling makes use of the relationship between the sample size and the accuracy of the data mining model, for example, the predictive accuracy of a classifier. This relationship can be depicted using a learning curve as shown in Figure 2.

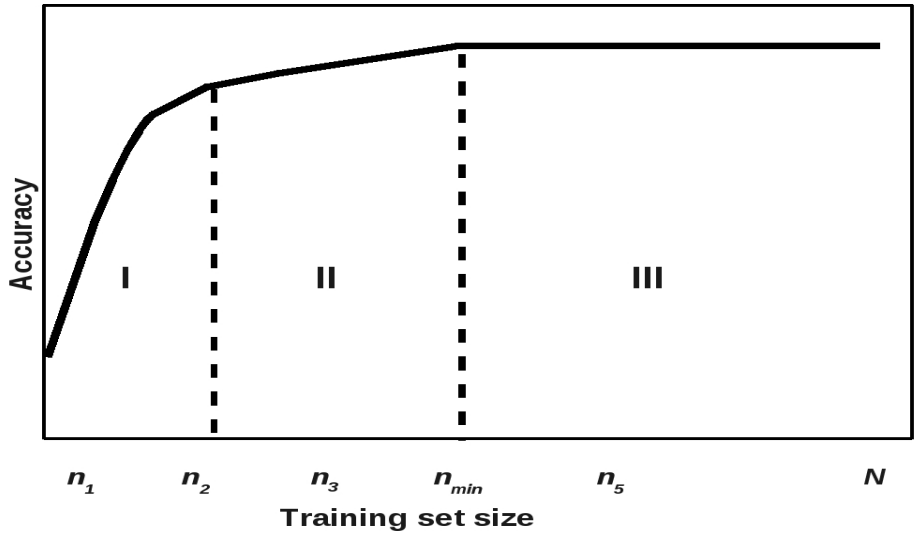



**Figure 2** Learning Curve.

Figure 2 shows the typical behaviour of a learning curve, which usually starts off steeply sloping (I). In the middle it is more gently sloping (II) and there is a plateau late in the curve (III). On the horizontal axis $N$ represents the total number of data instances and $n$ is the number of instances in the sample. The smallest sufficient sample size is denoted by $n_{min}$, a smaller sample causes a lower accuracy and a larger sample would not achieve a higher accuracy.

The slope of the learning curve indicates the rate of increase of accuracy of a data mining algorithm applied to two samples from a given dataset. Typically the slope of the learning curve decreases and almost levels off in the plateau portion. The rate of increase is then used as an indicator to find $n_{min}$. In order to obtain the slope, several runs of the data mining algorithm on different sizes of the sample are necessary. For a sample size the accuracy of the induced data mining model is measured. Progressive sampling assumes a relationship between the accuracy that the data mining model achieves and the sample size as depicted in Figure 2. However this learning curve is dependent on the ability of the used data mining algorithm to represent the regularities hidden in the data. So for some algorithms the plateau might be reached with small sample sizes and for others with very large data sizes.

## 4 Scaling Up Data Mining: Distributed Data Mining for Parallel Processing

Distributed Data Mining (DDM) is concerned with decomposing data mining tasks into many subtasks. There are two major approaches to doing so. *Finer-grained parallelisation* is used for tightly-coupled architectures or massively parallel processors as described in Section 2 and *coarser-grained parallelisation* is used for loosely-coupled architectures, for example, collections of standalone computers. This Section will review coarser grained distributed data mining approaches, and will initially introduce several distributed data mining models. The terminology 'distributed data mining approaches' in this Section comes from the terminology used in the literature reviewed here, however distributed data mining can be substituted by data parallel data mining in the context of this work. Given the constraint of utilising a loosely-coupled architecture, data parallelisation can be achieved by partitioning the data into subsets and assigning the subsets

to $n$ machines in a network of separate computers. This partitioning can happen with respect to selecting subsets of instances or subsets of features. This partitioning is done with the aim of also distributing the computational cost over $n$ machines in the network. (Provost, 2000) categorises distributed data mining approaches into several distributed data mining models of which two will be discussed here. All models have in common that they 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 is partitioned is not further specified. For example, the samples may contain a collection of instances or a collection of attributes and it is up to the actual algorithms to define the size of each sample. The subsamples might be of the same size, or the size might reflect the individual speed of the CPUs, or the size of the individual memory of each of the $n$ machines.
- *learning local concepts*: On each of the $n$ machines there is a learning algorithm $L$ installed that learns a local concept out of the data samples locally stored on each machine. Whether these $n$ learning algorithms $L_1$,...,$L_n$ do or do not communicate depends on the model. In general each $L$ has a local view of the search space reflected by the data it holds in the memory. By communication between several $Ls$ a global view of the search space can be obtained. This can be done by exchanging parts for the training data or information about the training data. Exchanging parts of the training data might be too expensive, concerning the bandwidth consumption when the training data is very large, whereas information about data usually involves statistics about the local data and the data itself, and these statistics are usually very small in size. Subsequently each $L$ will derive a concept description $C$ derived from the locally stored data and information that has been exchanged with other $Ls$.
- *combining procedure*: Eventually all $Cs$ 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. For example, in the case of classification rule induction each $L$ might derive a set of rules that is locally the best set of rules and then use the combining procedure to evaluate all rules (if they are good rules) on the whole training and test data.

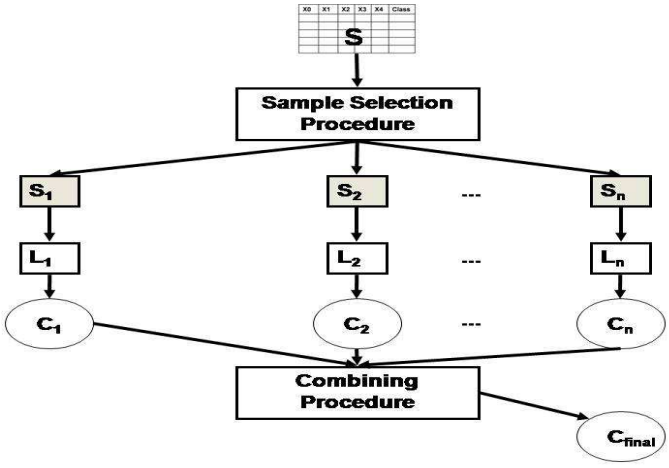



**Figure 3** Independent Multi Sample Mining.

Provost's DDM models do not make any assumptions about the underlying kind of learning algorithm. They should be seen more like a general way to describe parallel data mining algorithms

in a loosely-coupled environment. Figure 3 depicts the most basic DDM model; the *independent multi sample mining model*. The word 'independent' in the model name expresses that there is no communication or interaction between the different learning algorithms going on. Each algorithm forms its concept description independently. For example Meta Learning (Chan & Stolfo, 1993a) and the random forest approach (Breiman, 2001) follow the independent multi sample mining approach in a sequential fashion, but they could easily be run in parallel.

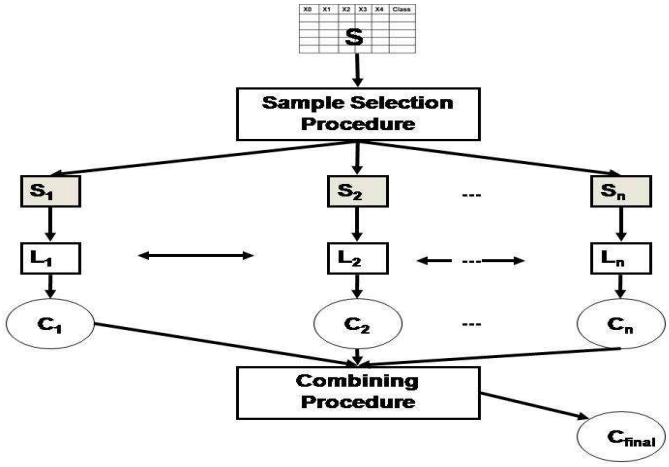



**Figure 4** Cooperating Data Mining Model.

However SMSM is not suited for running data mining algorithms in parallel, as each stage of the data mining process requires input from the previous one.

The probably most important model discussed here for parallelising data mining algorithms, is the *cooperating data mining model (CDM)*, which is depicted in Figure 4. CDM is based on a useful observation about certain evaluation metrics, in that every rule that is acceptable globally (according to a metric) must also be acceptable on at least one data partition on one of the $n$ machines (Provost & Hennessy, 1996, 1994). This observation is also known as the *invariant-partitioning property*. CDM allows the learning algorithms to cooperate in order to enable them to generate globally acceptable concept descriptions.

There are two more models discussed in (Provost, 2000), however they are not suitable for running data mining algorithms in parallel.

## 5 Parallel Formulations of Classification Rule Induction Algorithms

Several models for classification have been proposed in the past such as genetic algorithms (Goldberg, 1989), neural networks (Lippmann, 1988) and the induction of classification rules. Classification rule induction is the least computationally expensive and is also strongly competitive with genetic algorithms and neural networks. Research in classification rule induction can be traced back to at least the 1960s (Hunt, Stone, & Marin, 1966). The majority of classification rule induction algorithms can be categorised into 'divide and conquer' and 'covering' methods also known as 'separate and conquer' (Witten & Eibe, 1999) methods. The main differences between these methods are that they use different types of knowledge representations; 'divide and conquer' represents its classification rules in the form of decision trees and 'separate and conquer' in the form of rule sets.

'Divide and conquer' produces a decision tree by splitting an overly general rule into a set of specialised rules that cover separated subsets of the examples. Rules that cover examples of only one class are kept, while rules that cover several classes are further specialised in a recursive

manner. The 'divide and conquer' approach is also known as Top Down Induction of Decision Trees (TDIDT) (Quinlan, 1986). The following pseudo code describes the TDIDT algorithm.

```
IF   All instances in the training set belong to the
     same class
THEN return value of this class
ELSE (a) Select attribute A to split on
     (b) Divide instances in the training set
         into subsets, one for each value of A.
     (c) Return a tree with a branch for each non
         empty subset, each branch having a decendent
         subtree or a class value produced by applying
         the algorithm recursively
```

There have been many different proposals for attribute selection measures. However the two most frequently used ones are information gain, which is based on entropy, an information theoretic measurement that measures the uncertainty in a dataset associated with a random variable (Shannon, 1948) and the gini index which measures the impurity of a dataset regarding the classifications of the training instances (Breiman, Friedman, Olshen, & Stone, 1984).

The main drawback of the 'divide and conquer' approach lies in the intermediate representation of its classification rules in the form of a decision tree. Rules such as:

$$\text{IF a} = 1 \text{ AND b} = 1 \text{ THEN class} = 1$$

$$\text{IF c} = 1 \text{ AND d} = 1 \text{ THEN class} = 0$$

which have no attribute in common, could not be induced directly using the 'divide and conquer' approach. In such cases, TDIDT will first need to introduce additional rule terms 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. 'Separate and conquer' algorithms 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.

'Separate and conquer' produces a rule set by repeatedly specialising an overly general rule for the target class. At each iteration a specialised rule is generated that covers a subset of the positive examples (examples referring to the target class), which is repeated until all positive examples are covered by the rule set. Thus this approach is often referred to as the 'covering' approach. This strategy can be traced back to the AQ learning system (Michalski, 1969). The basic 'separate and conquer' approach can be described as follows:

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

The Learn_Rule procedure induces the best Rule for the current subset of the training set, and the perception of best depends on the heuristic used to measure the goodness of the rule, for example its coverage or prediction accuracy. 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 Criterion is fulfilled. The Stopping Criterion differs from algorithm to algorithm, but a quite common stopping criterion used is simply to stop when there are either no more examples left or the remaining examples are pure, meaning that all examples or instances are assigned to the same class. The most important part of the above algorithm is the Learn_Rule procedure, which searches for the best conjunction of attribute value pairs (rule terms). This process can be computationally very expensive, especially if all possible conjunctions of all possible rule terms have to be considered. There are two basic approaches to the Learn_Rule procedure according to which modular classification rule induction algorithms can be categorised. The first category considers all possible rule terms available such as CN2 (Clark & Niblett, 1989), RIPPER (Cohen,

1995) or Prism (Cendrowska, 1987) in order to specialise a rule. The other category uses a *seed example*, to form rule terms that only consist of attribute value pairs that match those of the seed example.

As discussed in Section 3 there are several approaches that aim to reduce the data to be mined. For example attribute selection aims to filter out the relevant attributes for the classification rule induction. An often used metric to evaluate attributes for relevance for the classification task is again the information gain. However even after attribute selection the dataset might still be very large, for example gene expression datasets can easily comprise tens of thousands of attributes, filtering for relevant attributes might still return thousands of attributes that could be relevant for the classification task. Sampling is probably the most popular method for data reduction. It is the representation of a large dataset by a smaller random sample of data instances. However Catlett's work (Catlett, 1991) showed that sampling of data results in a loss of accuracy in the induced classifier. Catlett's research was conducted almost 19 years ago and the data samples he used were fairly small compared with those used nowadays. In 1999 Frey and Fisher discovered that the rate of increase of accuracy slows down with the increase of the sample size (Frey & Fisher, 1999). Also classification rule induction algorithms often perform faster on categorical attributes than on continuous ones, thus discretisation of continuous attributes is a further way of making classification rule induction algorithms faster on large datasets (Kerber, 1992). In general it should be noted that data reduction techniques exist, however in the presence of very large datasets they are merely additional to parallelisation.

### *5.1 Parallel Formulations of Decision Trees*

This Section introduces some approaches to parallel TDIDT algorithms and Ensemble Learning. Parallel approaches to classification rule induction have focused on the TDIDT approach and very little work has been done on parallelising the 'separate and conquer' or covering approach. Like for most classification rule induction algorithms, the runtime of TDIDT is determined by the amount of training data used to induce the decision tree, thus data parallelism lends itself to parallelising TDIDT. (Sirvastava, Han, Kumar, & Singh, 1998) outlined two basic approaches to parallelising TDIDT, the *synchronous tree construction* and *partitioned tree construction* approaches, which will be discussed in this Section. Furthermore this Section discusses some actual parallel implementations of TDIDT and the Ensemble Learning approach to parallelisation.

#### *5.1.1 Synchronous Tree Construction*

The basic approach of 'Synchronous Tree Construction' is a data parallel approach, which constructs a decision tree synchronously by communicating class distribution information between the processors. Figure 5 shows the basic approach (Sirvastava et al., 1998), where the training instances are initially distributed to $n$ processors. When the term processor is used in this paper each processor also has its own private memory. So the approaches discussed here could be realised on a loosely-coupled system.

At any time each processor holds the whole decision tree induced so far in its memory. The top half of Figure 5 shows each processor holding the tree induced so far in the memory, the node that is going to be expanded next is coloured black. Next each processor gathers information about the local class distributions for the node to be expanded and communicates this information to the remaining processors. With this information each processor is able to calculate the best expansion of the current node. Displayed in the bottom part of Figure 5, is the decision tree in each processor's memory after the expansion, and the next node to be expanded is coloured black. Again, all processors cooperate by communicating class distribution information of the current node. Which node is expanded next depends on the tree induction algorithm. Some use a breadth-first and others a depth-first expansion.

The advantage of synchronous tree induction is that it does not require any communication of the training data, but it does suffer from high communication cost and workload imbalances.

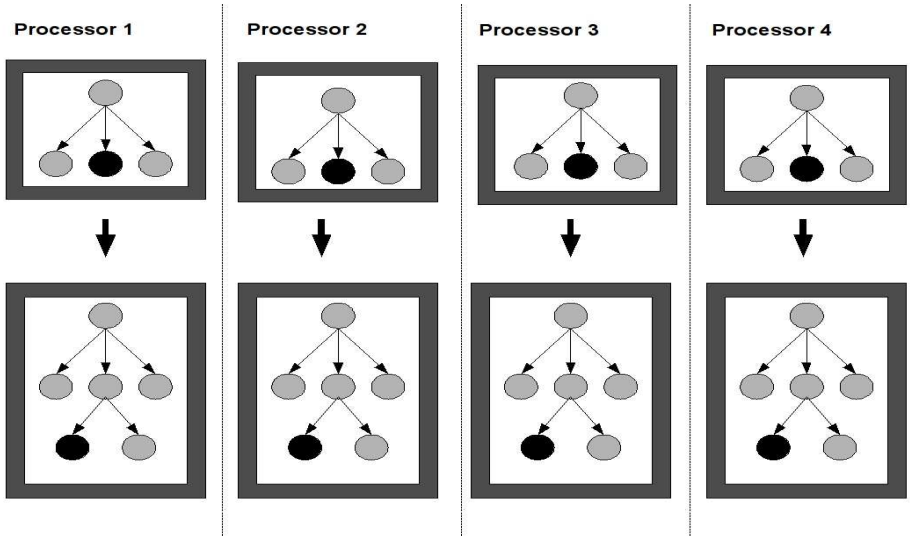



**Figure 5** Synchronous tree construction approach.

With respect to communication cost, for each node expansion the processors need to collect class distribution information and communicate it to the other processors in order to synchronise. At nodes in shallow depth of the tree the number of data records attached to each node is relatively large, but as the tree grows deeper, the number of training instances decreases and so does the time needed to compute class distribution information. On the other hand, the communication overhead does not decrease as much as the total workload as the tree grows deeper. Thus the communication overhead dominates the total processing time.

With respect to the workload imbalance, the number of training instances belonging to the same node of the tree can vary considerably amongst processors. For example, assume there are 2 processors and that processor 1 has all data instances on child node $A$ and there are none on $B$ and vice versa for processor 2. If $A$ is the current node then processor 2 would have no work to do and vice versa, processor 2 would have to do calculations for expanding node $B$.

### *5.1.2 Partitioned Tree Construction*

In the 'Partitioned Tree Construction' approach, whenever feasible each processor works on different subtrees of the decision tree. So if more than one processor is assigned to expand a node, then these processors are partitioned and assigned evenly (if possible) to the resulting child nodes. Figure 6 illustrates partitioned tree induction (Sirvastava et al., 1998). This approach can be seen as a data and task parallel hybrid. Different processors work on different parts of the tree, which is task parallelism. However as the data instances are attached to different parts of the tree and thus to different processors, this approach is also data parallel.

The top half of Figure 6 shows the expansion of the root node using three processors. Initially all processors follow the synchronous tree induction approach to expand the root node. In this case the expansion of the root node results in two child nodes and, if possible, the processors are assigned to different subtrees. As there are three processors and two subtrees, one processor is assigned to one of the subtrees and the remaining two processors are assigned to the remaining subtree. In this case processor 1 and 2 are assigned to the left child node and processor 3 to the right child node as illustrated on the bottom of Figure 6. Also the data instances are distributed according to the child node expansion. Processors 1 and 2 will continue following the synchronous tree construction approach. This is done until all processors work independently on different subtrees and once they all work solely on different subtrees, each processor can develop their subtrees independently without any communication overhead.

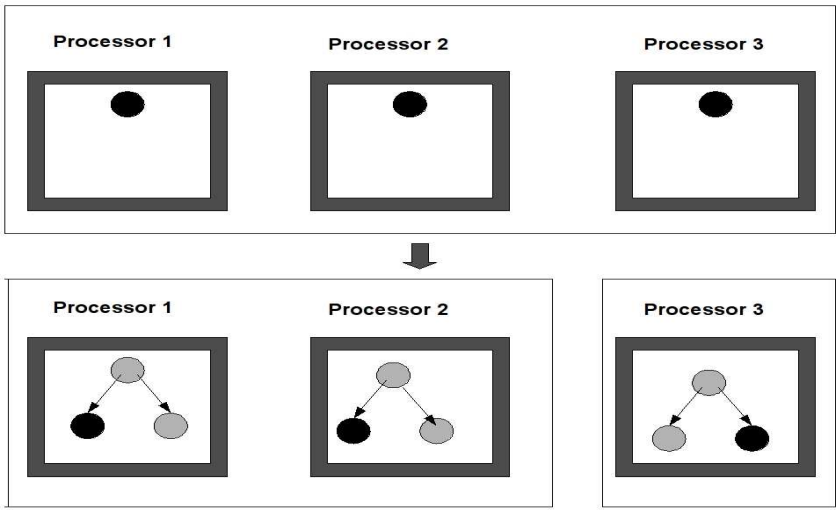



**Figure 6** Partitioned tree construction approach.

One of the disadvantages of this is the data movement in the first phase of the tree induction, which continues until all processors are solely responsible for an entire subtree. A further disadvantage is the communication overhead (as with the synchronous tree construction approach), but as with the data movement, this overhead drops to zero as soon as each processor works solely on one subtree. A further disadvantage of this approach lies in the workload balancing; the workload depends on the number of training instances assigned with the individual subtree of each processor. The workload on a certain processor drops to zero as soon as all child nodes are labelled as leaf nodes and this might happen at different points in time, especially if the induced tree is asymmetric, or the subtrees are attached to volumes of the training data that differ in size.

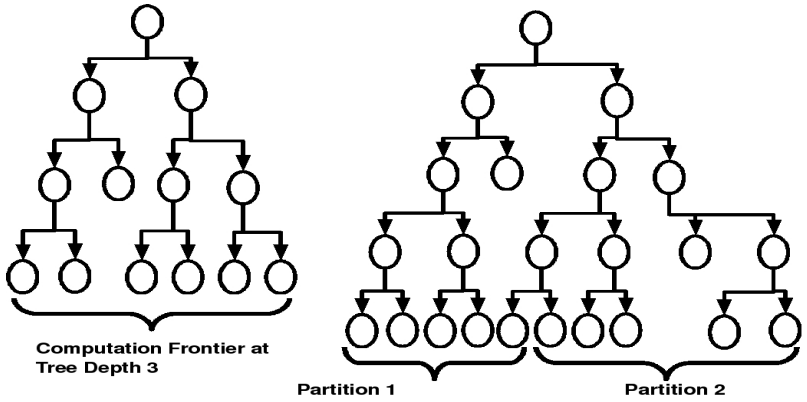



**Figure 7** The left hand side shows the computation frontier at depth 3 and the right hand side shows the binary partitioning of the tree to reduce communication costs.

(Sirvastava et al., 1998) developed a hybrid approach that attempts to minimise the disadvantages and maximise the advantages of both the synchronous and partitioned tree construction approaches. Their hybrid approach starts with the synchronous approach and keeps continuing until the communication becomes too high. Once the cost of communication reaches a defined threshold the processors and the current frontier of the decision tree are partitioned into two parts as illustrated in Figure 7. Partitioning may be repeated within a partition once the communication reaches the defined threshold again.

*5.1.3 Synchronous Tree Constructions by Partitioning the Training Instances Vertically*
The *Super Learning in Quest* (SLIQ) (Metha, Agrawal, & Rissanen, 1996) algorithm laid down a foundation that was used for several subsequent algorithm developments (Shafer, Agrawal, & Metha, 1996; Joshi, Karypis, & Kumar, 1998). Memory is saved by splitting the data vertically and by loading only one attribute list and one class list at any time into the memory in order to calculate the splits of the decision tree. The structure of an attribute list looks like <*attribute value, class index*> and the structure of the class list looks like <*class label, node*> as shown in Figure 8. The example data in Figure 8 is taken from (Metha et al., 1996).

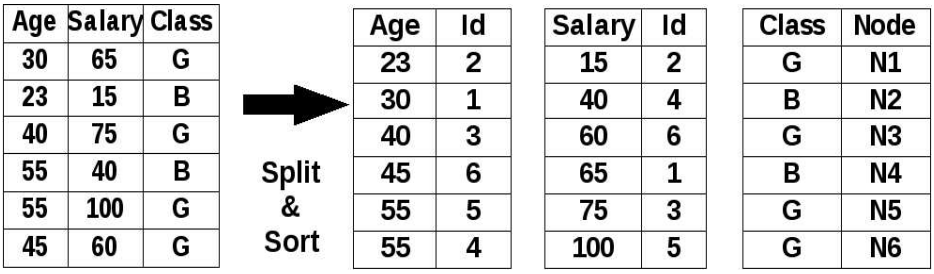
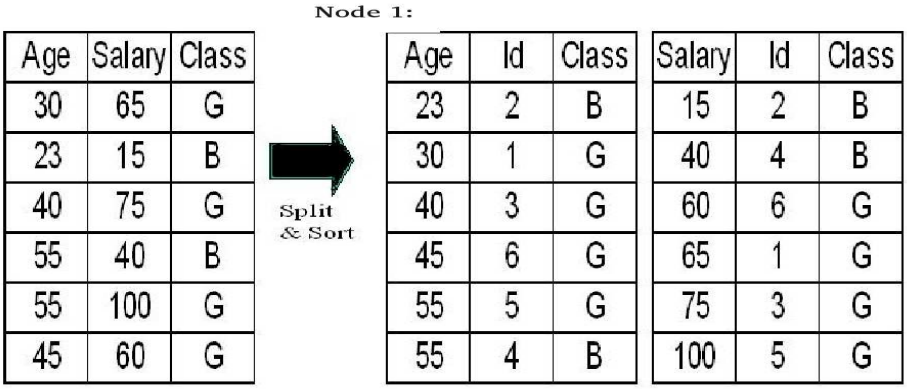



**Figure 8** Representing the data in the form of sorted attribute lists and a class list.

One contribution of SLIQ is that each attribute list is sorted right after the building of the attribute and class lists. Each attribute value holds a reference to the class value related to it in the class list, and thus also a reference to the current leaf. Single attribute lists need to be sorted only once and this saves valuable CPU time during the calculation of splits. The split evaluation metric used in the SLIQ implementation is the gini index, but it could easily be replaced by information gain or many other metrics. The split point in a sorted attribute list, for example the 'Salary' attribute list using the gini index can be found by scanning the attribute list for all possible binary splits, for example between ($Salary \leq 60$) and ($Salary > 60$). The gini index for each possible split is calculated and the split with the highest gini index is selected as split point for this particular attribute. After a split point has been found the split is performed by updating the class list accordingly and this is done by updating the 'Node' column, so each list record is referenced to its new node to which it is attached. SLIQ was initially developed to overcome memory constraints. In SLIQ only the class list and one attribute list need to be memory resident at the same time; the remaining attribute lists can be buffered to the hard disc. The disadvantage of SLIQ is that the usage of memory is directly determined by the size of the class list and therefore directly to the number of records in a data set; the class list plus one attribute list needs to be in memory at all times. Two possible approaches to parallelising SLIQ are discussed in (Shafer et al., 1996). The basic approach is to divide each sorted attribute list into $p$ sublists (if $p$ is the number of processors available) and distribute all sublists evenly over the $p$ processors. However the centralised and memory resident class list makes the parallelisation of SLIQ complicated, and one approach to parallelising SLIQ replicates the class list in the memory of each processor. This version is called SLIQ/R, where R stands for 'Replicated Class List'. Another approach is that the class list is partitioned and each processor holds a portion of it in its memory; this version is called SLIQ/D, where 'D' stands for 'Distributed Class List'.

The Scalable Parallelisable Induction of Decision Trees (SPRINT) algorithm (Shafer et al., 1996) represents a further development of the SLIQ algorithm by IBM. The aim of SPRINT is to remove memory limitations that are given by the class list in SLIQ. As in SLIQ, memory is saved by splitting the data set into attribute lists, but this time their structure looks like <*attribute value, tuple id, class*> and there is no class list.

Figure 9 shows how sorted attribute lists are created from the original data set. The data set used in Figure 9 is the same as the one used in Figure 8. As with SLIQ these attribute lists are pre-sorted so sorting in SPRINT is only performed once when the attribute lists are




| Age | Salary | Class |
|---|---|---|
| 30 | 65 | G |
| 23 | 15 | B |
| 40 | 75 | G |
| 55 | 40 | B |
| 55 | 100 | G |
| 45 | 60 | G |

| Age | Id | Class |
|---|---|---|
| 23 | 2 | B |
| 30 | 1 | G |
| 40 | 3 | G |
| 45 | 6 | G |
| 55 | 5 | G |
| 55 | 4 | B |

| Salary | Id | Class |
|---|---|---|
| 15 | 2 | B |
| 40 | 4 | B |
| 60 | 6 | G |
| 65 | 1 | G |
| 75 | 3 | G |
| 100 | 5 | G |

**Figure 9** Building sorted attribute lists in SPRINT.

created. SPRINT stores the information about the class within the attribute list. In SPRINT each processor induces the same decision tree and it attaches each attribute list to a certain node in the tree. At the beginning, all the attributes lists are attached to the root node *N1*.

Node 2 (Salary<60)

| Age | Id | Class |
|---|---|---|
| 23 | 2 | B |
| 55 | 4 | B |

| Salary | Id | Class |
|---|---|---|
| 15 | 2 | B |
| 40 | 4 | B |

Node 3 (Salary>=60)

| Age | Id | Class |
|---|---|---|
| 30 | 1 | G |
| 40 | 3 | G |
| 45 | 6 | G |
| 55 | 5 | G |

| Salary | Id | Class |
|---|---|---|
| 60 | 6 | G |
| 65 | 1 | G |
| 75 | 3 | G |
| 100 | 5 | G |

**Figure 10** Distribution of the attribute lists at the root node in SPRINT.

The next step is similar to SLIQ. Again the splits for all attribute lists are evaluated. The gini indices of all attribute lists at the current node are calculated. Again, the lowest gini index of all attribute lists at the concerning node is used to split all attribute lists, but the split in SPRINT is performed differently compared with SLIQ (as shown in Figure 10). The lists are logically split and list entries are not just referenced by different nodes using the class list. In this example the lowest gini index would be found for a split in the attribute 'Salary' for a split between values 40 and 60. Now all the attribute lists at the current node are split logically to form smaller lists and according to the splitting criteria they are then attached to the corresponding child nodes *N2* and *N3*.

The first step in the parallelised version of SPRINT is to distribute the data set over several processors. This is done by first building sorted attribute lists as in SPRINT's serial version, where each attribute list is then horizontally split in such a way that each processor gets an equal sized sorted Section of each attribute list.

Figure 11 shows the data distribution at node *N1* using 2 processors based on the example data set attribute lists shown in Figure 9. However SPRINT's scalability to large datasets (in terms of runtime and memory consumption) has been criticised by (Joshi et al., 1998). In the same paper a further evolution of SPRINT has been proposed, the ScalParC algorithm. The main criticism of SPRINT is that each processor determines the best split points for all the records it has. All processors then have to communicate in order to determine the best global split point, and the


**Figure 11** Data distribution in parallel SPRINT, using 2 processors.

next step in SPRINT is to split all lists according to the globally best split point. Therefore SPRINT builds a hash table that reflects a mapping of the attribute list records to nodes in the decision tree. This is used from all processors in order to split their own attribute lists, therefore this hash table has to be communicated as well. The size of the hash table is proportional to the number of instances of the tree at the current node on all processors. ScalParC proposes using a distributed hash table that does not need to be locally constructed.

### *5.2 Parallelisation Through Ensemble Learning*

Chan and Stolfo (Chan & Stolfo, 1993a, 1993b) considered partitioning the data into subsamples that fit into the memory of a single machine and developed a classifier in each subset separately. This can easily be run in parallel using the 'independent multi sample mining' approach described in Section 4. These classifiers are then combined using various algorithms in order to create a final classifier. There is no communication amongst the learning algorithms involved. This approach would reduce the runtime considerably. However the classifiers did not achieve the level of accuracy of a single classifier trained on the entire training data. In 'Meta Learning' the base classifiers that are computed on different portions of the training data are then collected and combined using a further learning process, a *'meta-classifier'*. Please note that Meta Learning does not specify the kind of classifier induced on each training subset, in fact it is possible to use different classifiers. The meta classifier's purpose is to integrate the separately induced classifiers. Initially Meta Learning was intended to improve the overall predictive accuracy rather than parallelisation.

Figure 12 (Chan & Stolfo, 1993b) illustrates the Meta-Learning framework using two data bases:

1. The base classifiers are trained using the initial training data sets.
2. Predictions are generated from a separate validation set using the learned classifiers from 1.
3. The Meta-Level Training Data set is constructed from the validation data and the predictions obtained from the validation data using the classifiers.
4. The meta or final classifier is generated using the Meta-Level data

Three main Meta-Learning strategies are used to generate the final classifier, *voting*, *arbitration* and *combining*:

- *Voting*: Each classifier gets a vote for a single prediction and the prediction with the majority is chosen. A variation of voting is the weighted voting where each vote from a certain classifier gets a weight according to their accuracy on the validation data.

- *Arbitration*: Includes a *judge classification algorithm* whose prediction is chosen if the participating classifiers do not reach a consensus prediction.
- *Combining*: Tries to coalesce the predictions of the initial classifiers based on their behaviour to each other. For example if classifier $A$ always predicts *class 1* correctly then it would make sense to use the prediction of classifier $A$ whenever $A$ predicts *class 1*.

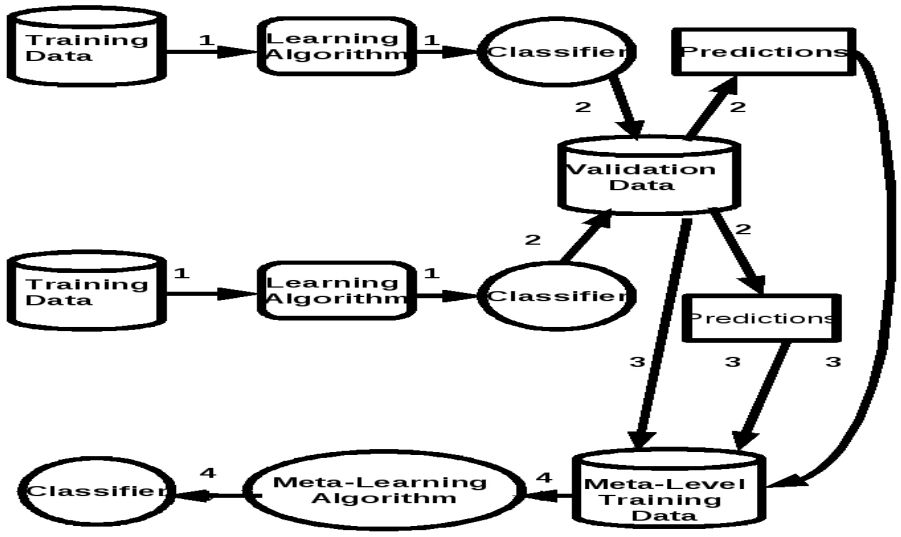



**Figure 12** Meta-Learning.

A further approach to ensemble learning methods is the Random Forests (RF) classifier from Breiman (Breiman, 2001). RF are inspired by the Random Decision Forests (RDF) approach from Ho (Ho, 1995). Ho argues that traditional trees often cannot be grown over a certain level of complexity without risking a loss of generalisation caused by overfitting on the training data. Ho proposes to induce multiple trees in randomly selected subsets of the feature space. He claims that the combined classification will improve, as the individual trees will generalise better on the classification for their subset of the features space. Ho evaluates his claims empirically. RDF has a potential to be parallelised using the independent multi sample mining approach discussed in Section 4.

RF makes use of the basic RDF approach by combining it with Breiman's bagging (**B**ootstrap **agg**regat**ing**) method (Breiman, 1996). Bagging is intended to improve a classifier's stability and classification accuracy. A classifier is unstable if a small change in the training set causes major variations in the classification. Inducing a classifier using bagging involves the following steps:

1. Produce $n$ samples of the training set by using sampling with replacement. Each sample has the same size as the initial training set. Some examples might appear several times in a sample; some might not appear at all.
2. Induce a classifier on each of the $n$ samples.
3. On each test example all classifiers are applied. The final classification is achieved using a voting schema.

Bagging can computationally be very expensive as several classifiers have to be induced in samples that are of the same size as the initial training set. However bagging has the potential to be parallelised using the independent multi sample mining approach illustrated in Section 4.

Finally RF combines the two, bagging and RDF. The basic approach is to learn several classifiers according to Breiman's bagging method, as outlined above. The RDF approach is incorporated in the splitting of each node in each tree by randomly choosing a subset of a defined

size of the overall feature space to base the splitting decision on. Again RF is computationally very expensive for the same reason as Breiman's bagging approach. Also RF can be parallelised using the independent multi sample mining approach. However recent work showed that in certain cases RF's classification accuracy is not able to compete with the accuracy of a serial classifier (Segal, 2004).

## 6 PMCRI and J-PMCRI: An Approach to Efficiently Parallelising Parallel Formulations of Modular Classification Rule Induction Algorithms

The last Section made the distinction between the 'separate and conquer' and 'divide and conquer' approach and outlined various approaches to parallelise algorithms that follow the 'divide and conquer' approach. On the other hand there have been virtually no attempts to parallelise algorithms that induce modular rules rather than decision trees. This Section outlines a recently developed PMCRI methodology to parallelise a family of algorithms that follow the 'separate and conquer' approach to classification rule induction.

### *6.1 Inducing Modular Classification Rules Using Prism*

Modular rules are rules that do not generally fit together naturally in a decision tree, although sometimes they might. Cendrowska's Prism algorithm induces modular rules.

In her paper Cendrowska (Cendrowska, 1987) describes the 'replicated subtree problem' of 'divide and conquer' approaches. However the name replicated subtree problem has been given by (Witten & Eibe, 1999). She argues that many rulesets do not fit naturally into a tree structure and the tree representation of rules is itself a major cause of overfitting. For example consider the following rules which have no attribute in common:

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

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

In Cendrowska's example all the attributes have three possible values and the two rules above cover all instances of class $x$. All remaining classes are labelled $y$. Again the root node has to split on a single attribute and there is no attribute in common in both rules.

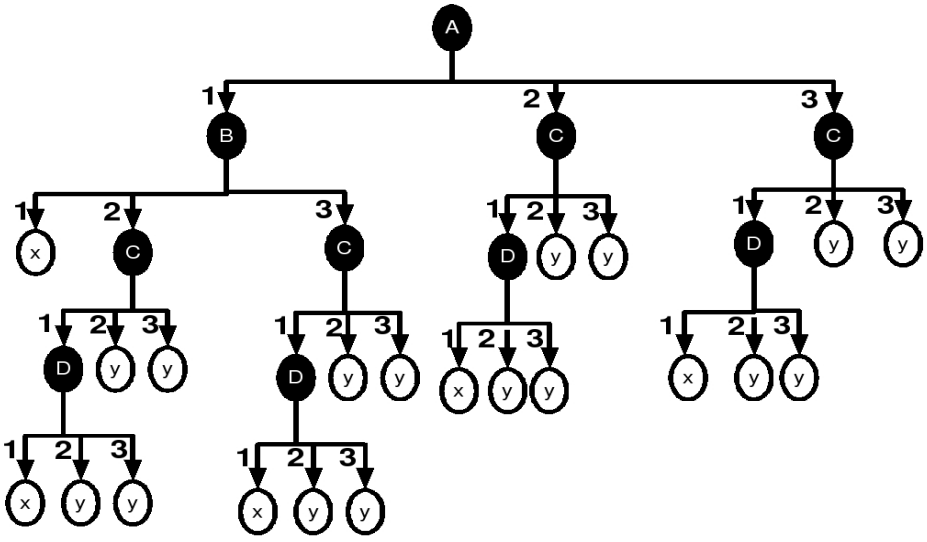



**Figure 13** Cendrowska's replicated subtree example.

The simplest tree representation that can be found to represent the two rules above is depicted in Figure 13. The rules that classify for class $x$ extracted from the tree 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*

The presence of such rules within a data set causes tree induction algorithms to induce unnecessarily large and confusing trees. Cendrowska also points out that decision trees are not always suitable for use in expert systems. To explain this she uses the example of a decision table for an optician for fitting contact lenses. This decision table is displayed in Table 1, where each data record represents a patient's attributes. The following is an explanation of Cendrowska's case study. The attributes are labelled $A$, $B$, $C$ and $D$ where $A$ corresponds to the age of the patient, $B$ is the patient's spectacle prescription, $C$ indicates whether the patient has astigmatism or not and $D$ is the tear production rate. The possible values of the attributes are listed below:

*A: 1 = young; 2 = pre-presbyopic; 3 = presbyopic*

*B: 1 = myope; 2 = hypermetrope*

*C: 1 = no; 2 = yes*

*D: 1 = reduced; 2 = normal*

The actual classes prescribe that either *no*, *soft* or *hard* contact lenses are suitable for the particular patient.

**Table 1** Opticians' decision table for fitting contact lenses.

| Record id | A | B | C | D | lenses? | Record id | A | B | C | D | lenses? |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | No | 13 | 2 | 2 | 1 | 1 | No |
| 2 | 1 | 1 | 1 | 2 | soft | 14 | 2 | 2 | 1 | 2 | soft |
| 3 | 1 | 1 | 2 | 1 | No | 15 | 2 | 2 | 2 | 1 | No |
| 4 | 1 | 1 | 2 | 2 | hard | 16 | 2 | 2 | 2 | 2 | No |
| 5 | 1 | 2 | 1 | 1 | No | 17 | 3 | 1 | 1 | 1 | No |
| 6 | 1 | 2 | 1 | 2 | soft | 18 | 3 | 1 | 1 | 2 | No |
| 7 | 1 | 2 | 2 | 1 | No | 19 | 3 | 1 | 2 | 1 | No |
| 8 | 1 | 2 | 2 | 2 | hard | 20 | 3 | 1 | 2 | 2 | hard |
| 9 | 2 | 1 | 1 | 1 | No | 21 | 3 | 2 | 1 | 1 | No |
| 10 | 2 | 1 | 1 | 2 | soft | 22 | 3 | 2 | 1 | 2 | soft |
| 11 | 2 | 1 | 2 | 1 | No | 23 | 3 | 2 | 2 | 1 | No |
| 12 | 2 | 1 | 2 | 2 | hard | 24 | 3 | 2 | 2 | 2 | No |

Now assuming a new patient needs some contact lenses fitted. The new patient's age is presbyopic *(A=3)*, the spectacle prescription is hypermetrope *(B=2)* and the patient has astigmatism *(C=2)*. Looking in the decision table it can be seen that there are two data records that match the patient's features which are the records with record ids 23 and 24. Both records are assigned to classification *No* regardless of the feature $D$ (tear production rate), in fact feature $D$ has different values for both cases and thus seems not to be relevant at all for this case. Thus the decision table strongly supports rule:

*IF A=3 AND B=2 AND C=2 THEN recommendation = No*

The decision tree that would be induced from this example using Quinlan's ID3 (Quinlan, 1979b, 1979a) decision tree induction algorithm is illustrated in Figure 14. Now assume that the decision tree in Figure 14 was used as the knowledge base for an expert system.

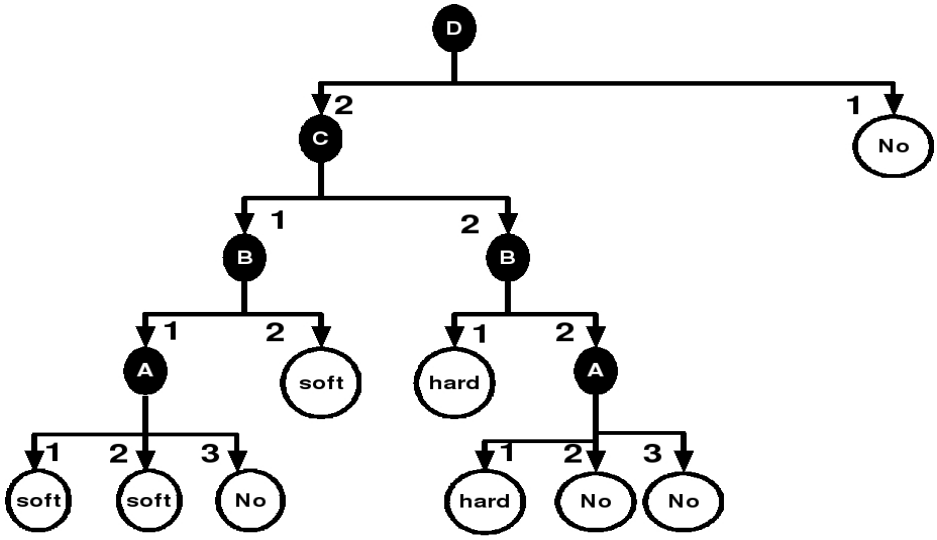



**Figure 14** Decision tree induced using ID3.

We can see that the expert system would not be able to make this decision without information about feature *D*, the tear production rate. So the optician would be required to make a tear production rate test. However tear production rate is restricted to two values *D=1* (reduced) and *D=2* (normal). If *D=1* then the expert system would make the decision that no contact lenses are suitable for the patient if *D=2* then the expert system would use the data the optician already knows about the patient which is *A=3 & B=2 & C=2* and also come to the conclusion that the patient is not suitable for contact lenses. So regardless of the value of *D* the decision made is that the patient is not suitable for the use of contact lenses. A test of the tear production rate will take some time and also may result in a fee to be paid by the patient. The patient would be very annoyed if he or she ever gets to know that the test was unnecessary and thus wasted his or her time and money.

### *6.2 The Prism Approach*

The general problem outlined above in Section 6.1 is further explained by Cendrowska in information theoretic terms. The argument is that the replicated subtree problem is caused by the fact that the attribute that is chosen at an intermediate node is more relevant to one classification and less for others. For example in Cendrowska's contact lenses example (Table 1) the initial entropy is 1.3261 bits. ID3 identifies that if the dataset *S* (Table 1) is split into subsets according to the values of attribute *D* then the average entropy of the resulting subtrees is minimised and thus the average amount of information gained is maximised. The resulting average entropy after performing the split on *D* is reduced to 0.7775. The entropy of the subset of *S* that covers all instances for which *D=1* is 0, however the subset that covers all instances for which *D=2* is 1.555 and so even higher than the initial entropy of *S*. Cendrowska's approach is to try to avoid the use of attribute values that are irrelevant for a certain classification. The Prism algorithm does that by maximising *'the actual amount of information contributed by knowing the value of the attribute to the determination of a specific classification'* (Cendrowska, 1987). Unlike decision trees, Cendrowska's algorithm looks at one *target class* at a time and specialises one rule at a time for this target class. In contrast, decision trees specialise several rules simultaneously when splitting on an intermediate node.

Cendrowska views Table 1 as a discrete decision system where attribute values including the classification are seen as discrete messages. She then gives a formula for the amount of information about an event in a message:

$$I(i) = log_2(\frac{probability\ of\ event\ after\ receiving\ the\ message}{probability\ of\ event\ before\ receiving\ the\ message})\ bits$$

Let us say that classification *lenses=No* is the classification of current interest (target class). Then the message *lenses=No* provides the following amount of information in the initial decision system about *lenses=No*:

$$I(lenses = No) = log_2(\frac{1}{p(lenses = No)}) = -log_2(\frac{15}{24}) = 0.678\ bits$$

The probability that *lenses=No* before receiving the message is $\frac{15}{24}$ and the probability that *lenses=No* when given the information that *lenses=No* is 1. In other words the maximum amount of information that can be achieved inducing a rule term on the current decision system for the concept *lenses=No* is 0.678 bits.

A concrete rule term *(D=1)* is considered for further specialisation of a rule for predicting *lenses=No*. Then the probability of the event *lenses=No* before the message *(D=1)* is *p(lenses=No)*=$\frac{15}{24}$*=0.625* and the probability of event *lenses=No* after the message *(D=1)* is the conditional probability $p(lenses = No \mid D = 1) = \frac{12}{12} = 1$. So the information about this message is:

$$\begin{aligned} I(lenses = No \mid D = 1) &= log_2(\frac{p(lenses = No \mid D = 1)}{p(lenses = No)}) \\ &= log_2(\frac{1}{0.625}) = 0.678\ bits \end{aligned}$$

The meaning of $I(lenses = No \mid D = 1) = 0.678$ bits is that the knowledge that *D=1* contributes 0.678 bits of information towards the concept that *lenses=No*. This value could be negative as well which means that knowing a certain attribute value makes it less certain that an instance belongs to a certain classification.

The information provided about *lenses=No* by knowing *D=1* is already the maximum amount of information we can achieve by inducing a rule term about the target class, which in both cases is 0.678 bits. So a further specialisation by adding more rule terms to the rule *IF(D=1) THEN lenses=No* would not increase the information about class *lenses=No*, also *D=1* covers only instances of the target class.

A further specialisation is justified if the information in the current rule about the target class is lower than 0.678 or if it covers further classes beside the target class. A further specialisation can be performed by repeating the process on the subset of Table 1 that contains only instances for which *D=1*. However, in this particular case the rule would be finished as the rule already contributes 0.678 bits of information. The next step now is to find the next rule for *lenses=No*, if there is one. This is done by deleting all instances from the initial training set that are covered by the rules induced so far for *lenses=No*. If there are still instances left with the classification *lenses=No* then the next rule is induced from this subset of the training data. If there are no instances left that belong to classification *lenses=No* then the whole process is repeated for one of the remaining classes using the initial training data from Table 1. A pseudocode description of Cendrowska's Prism algorithm can be found in Section 6.3.

### *6.3 The Prism Algorithms Family*

Cendrowska's basic Prism algorithm can be summarised as follows: where $A_x$ is a possible attribute value pair and D is the training dataset.

```
For each class i do {
  Step 1: Calculate  for each Ax  p(class = i| Ax)
  Step 2: Select the Ax with the maximum  p(class = i| Ax)
          and create a subset D' of D that comprises all instances
          that match the selected Ax.
  Step 3: Repeat 1 to 2 for D' until D'  only contains instances
```

```
          of classification i. The induced rule is then a
          conjunction of all the selected Ax and i.
  Step 4: Create a new D' that comprises all instances of D except
          those that are covered by the rules induced for
          class i so far.

  Step 5: Repeat steps 1 to 4 until D' does not contain any
          instances of classification i.
}
```

Note that instances are frequently deleted and again restored which causes a considerable overhead. An efficient implementation of dealing with frequent resetting and restoring data instances is essential but this is not outlined by Cendrowska.

### *6.3.1 Dealing with Clashes and Continuous Attributes*

Clashes occur whenever there are instances in a subset of the training set that are assigned to different classes but cannot be separated further. Such a subset is also called a *clash set*. This is inevitable whenever there are inconsistencies in the data, which is for example when there are two or more instances that have exactly the same attribute value pairs but are assigned to different classes. Cendrowska's original Prism does not take the existence of clashes into consideration. However the Inducer implementation of Prism addresses the problem of clashes (Bramer, 2000, 2005). When encountering a clash Inducer, by default, treats all instances in the clash set as if they belong to the target class of the rule that is currently being induced. However, according to (Bramer, 2000) the most effective approach whenever a clash set is reached is to check if the clash set's majority class is also the target class of the rule currently being generated. If this is the case, the rule is completed for the classification of the target class. However, if the target class is not the majority class of the clash set then the rule is discarded. The description in (Bramer, 2000) does not give any instructions how to deal with the clash set. However, if the current subset of the training data is not manipulated somehow, the same rule would be induced all over again and again discarded. Prism would be trapped in an endless loop. (Bramer, 2007) addresses the clash handling again and outlines a strategy to deal with the clash set. The strategy is to delete all instances in the clash set that are assigned to the discarded rule's target class. This keeps Prism from inducing the same rule all over again.

Cendrowska's initial version of Prism only works with discrete values. The problem of working with continuous data can be overcome by prior discretisation of the attribute values using algorithms such as ChiMerge (Kerber, 1992). However versions of Prism that deal with continuous data using local discretisation have been developed. For example the Inducer software (Bramer, 2005) provides Prism implementations that are able to deal with continuous data. The approach can be integrated in the pseudocode in Section 6.3 before the calculation of $p(class = i \mid A_x)$ (step 2 in the pseudocode in Section 6.3). If $A_x$ is continuous then the training data is sorted for $A_x$. For example, if $A_x$, after sorting, comprises values -3.45, -4.3, 5.3, 5.7 and 9.5 then the data is scanned for these attribute values in either ascending or descending order. For each attribute value a test such as $A_x < 5.3$ versus $A_x \geq 5.3$ is considered, using $p(class = i \mid A_x < 5.3)$ and $p(class = i \mid A_x \geq 5.3)$. In this scanning phase the term that has the largest conditional probability is retained and compared with the ones from the remaining attributes.

### *6.3.2 Variations of the Prism Algorithm*

The original version of Prism restores the training set to its original state before rules for each successive class are generated. Thus Prism has to process the full training set once for each class. The PrismTCS (**Prism** with **T**arget **C**lass, **S**mallest first) algorithm (Bramer, 2002) removes the outermost loop and thus can be seen as a first attempt to make Prism scale better. The difference is that, after the induction of each rule, Prism does not reset the dataset to its original state. Instead, PrismTCS deletes all instances from the training set that are covered by the rules induced so far and selects the minority classification as the new Target Class (TC) and induces the next rule for the minority class. This approach generally produces smaller rule sets whilst

maintaining a similar level of predictive accuracy. There is also an implementation of Prism available (Bramer, 2005) that selects the majority class as target class which is called PrismTC. Further versions of Prism exist in the Inducer software that induce the same rules as the original Prism but do so in a different order.

### *6.4 PMCRI: Parallel Modular Classification Rule Inducer*

A methodology for parallelising algorithms of the Prism family has been developed, the 'Parallel Modular Classification Rule Inducer' (PMCRI) which is reviewed here (Stahl, Bramer, & Adda, 2009b). The PMCRI framework is based on the Cooperative Data Mining (CDM) model (see Figure 4) (Provost, 2000) which is explained here briefly and then used to explain the overall PMCRI methodology.

The CDM model can be divided into three layers. The first is the sample selection procedure, where the training data and thus the workload is distributed in subsets $S_1$ ... $S_n$ if $n$ is the number of CPUs in a network. The second layer is the execution of algorithms $L_1$ ... $L_n$ on the training data subsets in order to produce local concept descriptions $C_1$ ... $C_n$. The learning algorithms have a local view of the local training data, however they may obtain a global view by communicating information between each other. In the third and last layer the local concepts are combined to a final concept description $C_{final}$.

#### *6.4.1 Distributing the Workload*

The basic idea of PMCRI is to build attribute lists similar to those in SPRINT (see Figure 9) of the structure *<attribute value, tuple id, class index>* and then distribute them evenly over $p$ processors. The learning algorithms then search for candidate rule terms and build the classifier in parallel.

However in contrast with SPRINT, attribute lists are not further split into $p$ part attribute lists if $p$ is the number of CPUs, instead whole attribute lists are distributed evenly over all CPUs. As pointed out by (Sirvastava et al., 1998), distributing part attribute lists will result in an initially perfect workload balance however it is likely that it will result later on in the algorithm in a considerable workload imbalance as part attribute lists may not evenly decrease in size. The same problem would be present for Prism algorithms as pointed out in (Stahl, 2009). Instead whole attribute lists are evenly distributed and a slight initial and predictable workload imbalance is accepted rather than an unpredictable workload imbalance during the algorithms execution.

The left hand side of Figure 15 shows the building of attribute lists from a training dataset. A rule term for class $B$ has been found ($Salary \leq 60.4$), which covers in the 'Salary' attribute list instances 5, 0, 2, 4. Thus the remaining list instances matching ids 1 and 3 need to be deleted in order to induce the next rule term. This deletion matches step 2 in the Prism pseudo code in Section 6.3. The attribute lists after the deletion of list records matching ids 1 and 3 are shown on the right hand side of Figure 15. What is important to note is that the resulting attribute lists are still equal in size, hence distributing complete attribute lists evenly over the CPUs will not cause a workload imbalance during the duration of the algorithm.

#### *6.4.2 Learning Rule Terms in Parallel using a Blackboard System*

Blackboard Systems are often described with the metaphor of experts gathered around a school blackboard confronted with a problem. Each expert can derive knowledge towards a solution of the problem using its own expertise and write this information on the blackboard. In turn an expert can read information written on the blackboard from other experts and use it in order to derive new knowledge. In a software system the blackboard is implemented as server and the experts as clients. Blackboard systems in AI can be dated back to the Hearsay II system (Erman, Hayes-Roth, Lesser, & Reddy, 1980) and experts are called Knowledge Sources (KS). The basic idea in PMCRI is that each CPU represents a particular KS and the expertise is determined by

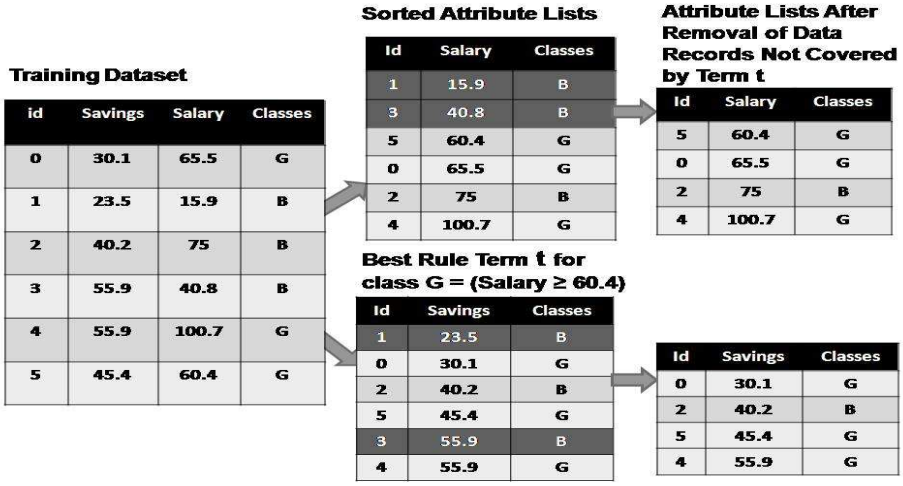



**Figure 15** The left hand side shows how sorted attribute lists are built and the right hand side shows how list records, in this case records with the ids 1 and 3, are removed in Prism, after a rule term has been found.

the subset of the attribute lists it holds in memory. Thus each KS is able to induce a rule term that is locally the best one on the attribute lists it holds in memory. The blackboard is used to coordinate the Prism algorithm and exchange information about locally induced rule terms. In this paper the terms KS, KS machine and expert are used interchangeably and denote a single CPU with its private memory hosting one KS.

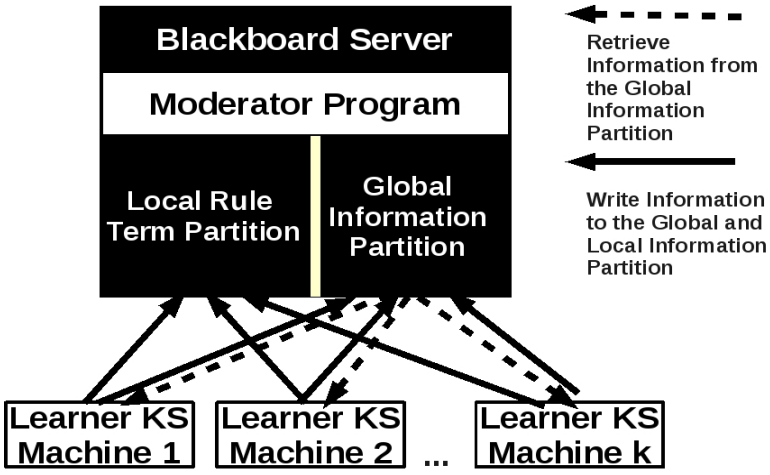



**Figure 16** PMCRI's communication pattern using a distributed blackboard architecture.

The communication pattern of PMCRI is depicted in Figure 16. 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. On the blackboard server there is also a moderator program. It reads information from the 'local rule term partition' and writes information on to the 'global information partition'. Learner KSs monitor the 'global information partition' and write information on the 'global' and 'local information partitions'. When a Learner KS detects that a pre-specified condition applies, a corresponding action is triggered. Thus the learner KS machines

can be seen as entities that contribute local knowledge and the moderator can be seen as a learner KS activator. The local rule term information exchanged using the blackboard is the probability with which the locally induced rule terms cover the target class.

All $k$ KS machines induce the locally best rule term and write the probability with which the rule term covered the target class in the attribute list on the 'local information partition'. The moderator program compares the submitted probabilities and advertises the name of the KS that induced the rule term with the globally highest probability on the global information partition. The winning KS then communicates the ids of the instances that are uncovered from its induced rule term to the remaining experts using again the blackboard. Next all KS continue by inducing the next locally best rule term.

The following steps listed below describe how PMCRI induces one rule (Stahl, Bramer, & Adda, 2008) based on the Prism algorithm:

```
Step 1 Moderator writes on Global Information Partition
       the command to induce locally best rule terms.
Step 2 All KSs induce the locally best rule term and write the
       rule terms plus its covering probability on the
       local Rule Term Partition
Step 3 Moderator compares all rule terms written on the
       Local Rule Term Partition and writes the name
       of the KS that induced the best rule term on the
       "Global Information Partition"
Step 4 KS retrieves name of winning expert.
       IF KS is winning expert {
          keep locally induced rule term and derive by last
          induced rule term uncovered ids and write
          them on the Global Information Partition and delete
          uncovered list records
       }
       ELSE IF KS is not winning KS {
           delete the locally induced rule term and
           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.
       }
```

### *6.4.3 Combining Procedure*

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, a rule is distributed over the network, thus the concept description induced by each learner is a part of the overall classifier.

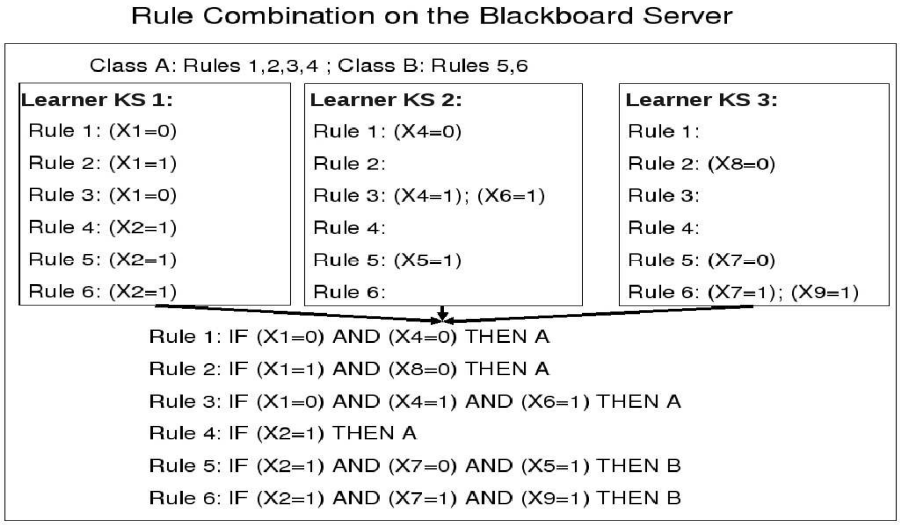



**Figure 17** Combining Procedure.

In the learner pseudocode all rules are stored in a list or, in general, in a 'Rule_Collection' which is in the order in which the rules were induced. Also each learner remembers the target class for which a 'part-rule' was induced. This information in the form of a collection of 'part-rules' needs to be collected at the terminal where the actual rules shall be printed or stored in some form. Finally all 'part-rules' need to be combined in order to build the 'globally complete' rule. How this information is collected is not specified in the PMCRI framework, also the combining procedure only appears at the end of the algorithm's execution and thus imposes only a small computational and bandwidth overhead. In the current implementation of PMCRI all 'part-rules' are written on the local rule term information partition and then picked up by the terminal that is supposed to store or output the rules to the user. The terminal could theoretically be any learner KS machine, or the blackboard itself or a separate KS to the blackboard.

The combining procedure of the learner KS that collects all 'part-rules' is very simple as illustrated in an example in Figure 17. The example 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. The combining procedure then simply appends each rule term to its corresponding rule. It is important to note that the PMCRI framework is able to reproduce exactly the same result as any serial Prism algorithm.

### *6.4.4 J-PMCRI: PMCRI with Information Theoretic Pre-Pruning*

Pruning is a commonly used technology to reduce overfitting of the induced classification rules. Post-pruning methods are applied to the already trained rules and pre-pruning is applied during the induction of the classification rules. Parallel versions of TDIDT often do not implement pre-pruning technologies with the reasoning that post-pruning is computationally inexpensive and thus easier to realise in parallel TDIDT implementations compared with pre-pruning which would have to be parallelised as well (Shafer et al., 1996; Sirvastava et al., 1998). This argument is true, however using post-pruning unnecessarily forces the induction of rule terms that will be removed by pruning anyway, where pre-pruning avoids inducing these rule terms in the first place. Hence it is useful to develop parallel pre-pruning technologies. (Bramer, 2002) developed a pre-pruning method for all algorithms of the Prism family. The pre-pruning method is based on the J-measure of Smyth and Goodman (Smyth & Goodman, 1992) and works also for TDIDT and shows on both, Prism and TDIDT, a good performance (Bramer, 2002) regarding the predictive accuracy and the number of rule terms induced. Hence the PMCRI methodology has been extended by a parallel J-pruning facility, resulting in the J-PMCRI methodology. For further reading and evaluation about J-PMCRI refer to (Stahl, Bramer, & Adda, 2010).

## *6.5 Computational Performance of the PMCRI and J-PMCRI Methodology*

The evaluation results summarised here can be found for PMCRI in (Stahl, Bramer, & Adda, 2009a) and for J-PMCRI in (Stahl et al., 2010). In order to investigate the scalability of PMCRI, with respect to the training data size, runtime of a fixed processor (learner KS machine) configuration on an increasing workload have been examined. These experiments are called *size up experiments* in contrast with *speed up experiments* that keep the workload constant and increase the number of processors. It has been observed that for PMCRI and J-PMCRI the workload is equivalent to the number of data records and attributes that are used to train a Prism classifier. For both, PMCRI and J-PMCRI a linear size up has been achieved , meaning that the runtime is a linear function of the data set size. With respect to speed up, for both, PMCRI and J-PMCRI it has been found that the larger the amount of data used, the more PMCRI and J-PMCRI benefit from using additional processors. Also it has been observed that the memory consumption of both, PMCRI and J-PMCRI is linear with respect to the total training data size.

## 7 Summary and Concluding Remarks

This paper starts with highlighting example applications that generate massive amounts of data. Data mining these massive amounts needs attention from a computational point of view. Often a single workstation is not enough to hold the training data in memory and would take too long to process. Hence the area of parallel and distributed data mining is more topical than ever before. Next this paper distinguishes between the terms 'parallel' and 'distributed' data mining. The research literature is often inconsistent with the usage of the term 'distributed data mining', they either mean geographically distributed data sources or distributing data over several workstations in order to divide the CPU time needed to process the data to several machines. This paper used the term parallel data mining to refer to the latter usage of the term 'distributed data mining'. In Section 2 two basic parallel system architectures are discussed that can be used to parallelise data mining algorithms, 'tightly-coupled' and 'loosely-coupled' systems. This paper focuses on 'loosely-coupled' systems as they can be realised by a network of workstations and are thus accessible to even modest sized organisations.

Section 3 discusses the data reduction approaches in order to reduce the data volume the classifier is being trained on, notably feature selection techniques and sampling. However the challenge with sampling is to determine the optimal sample size. Section 4 gives some general data mining models derived by Provost. These models are helpful in order to describe different approaches to distributed/parallel data mining.

Section 5 highlights two general approaches to classification rule induction, the induction of decision trees also known as the 'divide and conquer' approach and induction using covering algorithms also known as the 'separate and conquer' approach. Then some general parallel approaches to the 'divide and conquer' approach have been discussed, notably the 'Synchronous Tree Construction' approach and the 'Partitioned Tree Construction' approach. In the Synchronous Tree Construction approach the training data is split over $p$ CPUs and each CPU induces the same tree in parallel whilst exchanging statistics about the data located on different CPUs. The Partitioned Tree Construction approach outsources the induction of whole subtrees to separate CPUs. Both approaches have serious disadvantages. The Synchronous Tree Construction approach suffers from a high communication overhead and the Partitioned Tree Construction approach from workload imbalances. Also a hybrid approach has been discussed. However the 'Synchronous Tree Construction by Partitioning the Training Instances Vertically' approach has seen some concrete systems for parallel decision tree induction. Here the training data is partitioned attribute wise, each CPU holds a subset of the attribute lists in memory and all CPUs induce the same tree whilst exchanging statistics about the data located on different CPUs. Notable systems that have been successful were highlighted more closely, the SLIQ and the SPRINT systems. SLIQ and SPRINT build so called attribute lists out of each attribute and split these lists into $p$ equally sized chunks if $p$ is the number of CPUs. Then each chunk of each list is assigned to a different CPU. Next some ensemble learning strategies have been introduced and highlighted as a possibility to parallelise decision tree induction with a low communication overhead.

Section 6 then points out that there are virtually no approaches to speeding up the 'separate and conquer' approach with parallelisation. However the recent development of the PMCRI and J-PMCRI methodology has been highlighted as a first attempt to parallelise the 'separate and conquer' approach. PMCRI parallelises any algorithm of the Prism family. The research literature provides evidence that Prism outperforms decision trees in many cases because it does not represent rules in the form of trees but rather as 'IF THEN ELSE rules'. Also PMCRI uses attribute lists similar to SPRINT in order to partition and distribute the features space evenly over $p$ CPUs. PMCRI's central component is a blackboard architecture that is used to coordinate the communication. In contrast with SPRINT, PMCRI induces different rule terms on different CPUs and not synchronously. The research literature about PMCRI and J-PMCRI provides evidence that PMCRI scales well on large datasets. PMCRI may represent the only approach to

scale up the 'separate and conquer' approach with parallelisation, however it is not only restricted to Cendrowska's original Prism algorithm, but all members of the Prism family of algorithms or in general 'separate and conquer' algorithms that can process each attribute independently from the others can be parallelised. Future developments in PMCRI may comprise the parallelisation of algorithms that generate generalised rules where the right hand side of each rule is not restricted to a single classification attribute but any combination of attributes.

Although the technologies outlined here are focused on *loosely-coupled* architectures, as they can be realised with commodity hardware, hybrid technologies of *loosely-coupled* and *tightly-coupled* architectures are likely to come into focus in the coming years. This is because of the new generation of processors that comprise several processor cores that share one memory. Nowadays *dual-core* processors are standard technology, but in the near future we will have *multi-core* processors in our personal computers. Once APIs become available that allow researchers to conveniently develop data mining applications that make use of all cores simultaneously, then data mining researchers will be able to derive hybrid technologies that harvest the computational power of networks of *multi-core* processor computers. Or loosely speaking, future parallel data mining systems will be utilising whole networks of *shared memory multi processor* machines.

## References


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

Bramer, M. A. (2000). Automatic induction of classification rules from examples using N-Prism. In *Research and development in intelligent systems XVI* (pp. 99–121). Cambridge: Springer-Verlag.

Bramer, M. A. (2002). An information-theoretic approach to the pre-pruning of classification rules. In B. N. M Musen & R. Studer (Eds.), *Intelligent information processing* (p. 201-212). Kluwer.

Bramer, M. A. (2005). Inducer: a public domain workbench for data mining. *International Journal of Systems Science*, *36*(14), 909–919.

Bramer, M. A. (2007). *Principles of data mining*. Springer.

Breiman, L. (1996). Bagging predictors. *Machine Learning*, *24*(2), 123–140.

Breiman, L. (2001). Random forests. *Machine Learning*, *45*(1), 5–32.

Breiman, L., Friedman, J. H., Olshen, R. A., & Stone, C. J. (1984). *Classification and regression trees*. Belmont, California, U.S.A.: Wadsworth Publishing Company.

Caragea, D., Silvescu, A., & Honavar, V. (2003). Decision tree induction from distributed heterogeneous autonomous data sources. In *In proceedings of the conference on intelligent systems design and applications (isda 03* (pp. 341–350). Springer Verlag.

Catlett, J. (1991). *Megainduction: Machine learning on very large databases*. Unpublished doctoral dissertation, University of Technology Sydney.

Cendrowska, J. (1987). PRISM: an algorithm for inducing modular rules. *International Journal of Man-Machine Studies*, *27*(4), 349–370.

Chan, P., & Stolfo, S. J. (1993a). Experiments on multistrategy learning by meta learning. In *Proc. second intl. conference on information and knowledge management* (pp. 314–323).

Chan, P., & Stolfo, S. J. (1993b). Meta-Learning for multi strategy and parallel learning. In *Proceedings. second international workshop on multistrategy learning* (pp. 150–165).

Clark, P., & Niblett, T. (1989). The CN2 induction algorithm. *Machine Learning*, *3*(4), 261–283.

Cohen, W. W. (1995). Fast effective rule induction. In *Proceedings of the twelfth international conference on machine learning* (pp. 115–123). Morgan Kaufmann.

Erman, L. D., Hayes-Roth, F., Lesser, V. R., & Reddy, D. R. (1980). The Hearsay-II Speech-Understanding system: Integrating knowledge to resolve uncertainty. *ACM Computing Surveys (CSUR)*, *12*(2), 213–253.

Freitas, A. (1998). A survey of parallel data mining. In *Proceedings second international conference on the practical applications of knowledge discovery and data mining* (pp. 287–300). London.

Frey, L. J., & Fisher, D. H. (1999). Modelling decision tree performance with the power law. In *Proceedings of the seventh international workshop on artificial intelligence and statistics* (pp. 59–65).

Fuernkranz, J. (1998). Integrative windowing. *Journal of Artificial Intelligence Resarch*, *8*, 129–164.

Goldberg, D. (1989). *Genetic algorithms in search, optimization and machine learning.* Addison-Wesley.

Han, J., & Kamber, M. (2001). *Data mining: Concepts and techniques.* Morgan Kaufmann.

Hillis, W., & Steele, L. (1986). Data parallel algorithms. *Communications of the ACM*, *29*(12), 1170–1183.

Ho, T. K. (1995). Random decision forests. *Document Analysis and Recognition, International Conference on*, *1*, 278.

Hunt, E. B., Stone, P. J., & Marin, J. (1966). *Experiments in induction.* New York: Academic Press.

Joshi, M., Karypis, G., & Kumar, V. (1998). 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).

Kargupta, H., Byung-Hoon, Hershberger, D., & Johnson, E. (1999). Collective data mining: A new perspective toward distributed data analysis. In *Advances in distributed and parallel knowledge discovery* (pp. 133–184). AAAI/MIT Press.

Kerber, R. (1992). Chimerge: Discretization of numeric attributes. In *AAAI* (p. 123-128).

Lippmann, R. P. (1988). An introduction to computing with neural nets. *SIGARCH Comput. Archit. News*, *16*(1), 7–25.

McClean, B., Hawkins, C., Spagna, A., Lattanzi, M., Lasker, B., Jenkner, H., et al. (1998). New horizons from multi-wavelength sky surveys. In *Proceedings of the 179th symposium of the international astronomical union held in baltimore.*

Metha, M., Agrawal, R., & Rissanen, J. (1996). SLIQ: a fast scalable classifier for data mining. In *Proceedings of the 5th international conference on extending database technology: Advances in database technology* (Vol. 1057, pp. 18–32). Springer.

Michalski, R. S. (1969). On the Quasi-Minimal solution of the general covering problem. In *Proceedings of the fifth international symposium on information processing* (pp. 125–128). Bled, Yugoslavia.

*Minitab.* (2010, http://www.minitab.com/).

Park, B., & Kargupta, H. (2002). Distributed data mining: Algorithms, systems and applications. In *Data mining handbook* (pp. 341–358). IEA.

Provost, F. (2000). Distributed data mining: Scaling up and beyond. In *Advances in distributed and parallel knowledge discovery* (pp. 3–27). MIT Press.

Provost, F., & Hennessy, D. N. (1994). Distributed machine learning: scaling up with coarse-grained parallelism. In *Proceedings of the second international conference on intelligent systems for molecular biology* (pp. 340–347).

Provost, F., & Hennessy, D. N. (1996). Scaling up: Distributed machine learning with cooperation. In *Proceedings of the thirteenth national conference on artificial intelligence* (pp. 74–79). Menlo Park, CA: AAAI Press.

Provost, F., Jensen, D., & Oates, T. (1999). Efficient progressive sampling. In *International conference on knowledge discovery and data mining* (pp. 23–32). San Diego: ACM.

Quinlan, R. J. (1979a). Discovering rules by induction from large collections of examples. In *Expert systems in the micro-electronic age.* Edinburgh: Edinburgh University Press.

Quinlan, R. J. (1979b). *Induction over large databases* (Technical No. STAN-CS-739). Stanford University.

Quinlan, R. J. (1983). Learning efficient classification procedures and their applications to chess endgames. In *Machine learning: An AI approach* (pp. 463–482). Morgan Kaufmann.

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

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

*Sas/stat.* (2010, http://www.sas.com/).

Segal, M. R. (2004). *Machine learning benchmarks and random forest regression* (Tech. Rep.). San Francisco, CA: Center for Bioinformatics & Molecular Biostatistics, University of California.

Shafer, J., Agrawal, R., & Metha, M. (1996). SPRINT: a scalable parallel classifier for data mining. In *Proc. of the 22nd int'l conference on very large databases* (pp. 544–555). Morgan Kaufmann.

Shannon, C. E. (1948). A mathematical theory of communication. *The Bell System Technical Journal*, *27*.

Sirvastava, A., Han, E., Kumar, V., & Singh, V. (1998). Parallel formulations of Decision-Tree classification algorithms. *Data Mining and Knowledge Discovery*, 237–261.

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

Stahl, F. (2009). *Parallel rule induction*. Unpublished doctoral dissertation, University of Portsmouth.

Stahl, F., Berrar, D., Silva, C. S. G., Rodrigues, J. R., Brito, R. M. M., & Dubitzky, W. (2005). Grid warehousing of molecular dynamics protein unfolding data. In *Proceedings of the fifth IEEE/ACM int'l symposium on cluster computing and the grid* (pp. 496–503). Cardiff: IEEE/ACM.

Stahl, F., Bramer, M., & Adda, M. (2008). Parallel induction of modular classification rules. In *Sgai conf.* (p. lookup-lookup). Springer.

Stahl, F., Bramer, M., & Adda, M. (2009a). Parallel rule induction with information theoretic pre-pruning. In *Sgai conf.* (p. 151-164).

Stahl, F., Bramer, M., & Adda, M. (2010). J-PMCRI: A methodology for inducing pre-pruned modular classification rules. In *Artificial intelligence in theory and practice III* (pp. 47–56). Brisbane: Springer.

Stahl, F., Bramer, M. A., & Adda, M. (2009b). PMCRI: A parallel modular classification rule induction framework. In *MLDM* (pp. 148–162). Springer.

Stankovski, V., Swain, M., Kravtsov, V., Niessen, T., Wegener, D., Roehm, M., et al. (2008). Digging deep into the data mine with dataminninggrid. *IEEE Internet Computing*, *12*, 69-76.

Szalay, A. (1998). *The evolving universe*. ASSL 231.

Way, J., & Smith, E. A. (1991). The evolution of synthetic aperture radar systems and their progression to the EOS sar. *IEEE Transactions on Geoscience and Remote Sensing*, *29*(6), 962–985.

Wirth, J., & Catlett, J. (1988). Experiments on the costs and benefits of windowing in ID3. In *Proceedings of the fifth international conference on machine learning (ML-88)* (pp. 87–95). Ann Arbor: Morgan Kaufmann.

Witten, I. H., & Eibe, F. (1999). *Data mining: Practical machine learning tools and techniques with java implementations*. Morgan Kaufmann.