Mining Taxation Data with Parallel BMARS[*]

Sergey Bakin
Department of Mathematics
The University of Queensland
St. Lucia, Brisbane, Qld 4072, Australia
Sergey@maths.uq.edu.au

Markus Hegland
Computer Sciences Laboratory
Australian National University
ACT 0200, Australia
Markus.Hegland@anu.edu.au

Graham Williams
CSIRO Math and Info Sciences
GPO Box 664, Canberra
ACT 2601, Australia
Graham.Williams@cmis.csiro.au

Abstract

A new parallel version of Friedman's Multivariate Adaptive Regression Splines (MARS) algorithm is discussed. By partitioning the data over the processors of a parallel computational system one achieves good parallel efficiency. Instead of using truncated power basis functions of the original MARS, the new method (BMARS) utilises B-splines which improves numerical stability and reduces the computational cost of the procedure. In addition, the coefficients of the basis functions of a BMARS model provide quickly accessible information about the local behaviour of the function. The algorithm has a time complexity proportional to the number of data records. The method provides a new means for the detection of areas in the space of features which are characterised by the "interesting" patterns of response values. This is applied to searching for classes of incorrect tax returns using multiple predictor variables or features. The parallel algorithm makes it feasible to investigate very large databases, such as the taxation database.

[*]The authors acknowledge the support provided by the Cooperative Research Centre for Advanced Computational Systems (ACSys) established under the Australian Government's Cooperative Research Centres Program and the support provided by the Australian Taxation Office.

1 Introduction

Millions of dollars are at risk each year due to inappropriate tax practices. Thus there are large benefits if one is able to detect errors and fraud. While a detailed analysis of every tax return is infeasible one is interested in tools to detect classes of people which are more likely to be involved in fraud. The tax declarations of these people can then be further examined. An important tool used to find the different classes of people generates a predictive model from the data. This model provides an estimate of the risk that fraud might have been committed.

A major difficulty in finding a good risk model is posed by the size of the data which typically contains records from several years with millions of records per year and up to 100 features per record. It is extremely important that methods are used which scale well with the data size and are capable of handling high dimensional data. Ideally, the research into fraudulent behaviour is done with an interactive system and the short response time provided by parallel processing is essential. This scenario is typical for data mining applications [5].

From a statistical point of view the determination of risk from data can be addressed by logistic regression. As the investigation is of an exploratory nature, spline-based techniques are used which do not make limiting assumptions on the form of the functions. The Multivariate Adaptive Regression Splines method [6] successfully addresses high dimensionality and heterogeneous predictor variables. It is also scalable.

Typically, only a very small percentage of all records in a database represent fraud. If the records are indexed by n, the response variable y_n shall denote fraud ($y_n = 1$) or no fraud ($y_n = 0$). The other attributes of the record are collected in the predictor variable $\mathbf{x}_n$ with both numeric and categorical components. Then the model used is that the y_n are independent random variables and $p(\mathbf{x}_n)$ shall denote the probability that $y_n = 1$. Thus

$$y_n \sim \text{Bernoulli}[p(\mathbf{x}_n)], \quad n = 1, \dots, N.$$

The estimate $\hat{p}$ of p is modelled as

$$\hat{p}(\mathbf{x}) = \frac{\exp(\hat{f}(\mathbf{x}))}{1 + \exp(\hat{f}(\mathbf{x}))},$$

where $\hat{f}(\mathbf{x})$ is constructed by MARS in the following additive form [6]:

$$\hat{f}(\mathbf{x}) = \sum_{v=1}^{d} f_v(x_v) + \sum_{v_1 > v_2}^{d} f_{v_1,v_2}(x_{v_1}, x_{v_2}) + \cdots + \sum_{v_1 > \ldots > v_k}^{d} f_{v_1,\ldots,v_k}(x_{v_1}, \dots, x_{v_k}).$$

The highest order k of the interaction terms typically is chosen between 1 and 4 and x_v is the v-th component of $\mathbf{x}$. These models have also been called ANOVA splines [16], [3] due to their similarity with the additive models in analysis of variance. The interaction terms reveal essential information as they allow the determination of localised "pockets" in the data characterised by a higher number of occurrences of detected fraud.

MARS constructs $\hat{f}$ as a linear combination of basis functions $\hat{f}(x) = \sum_{j=0}^{J} a_j T_j(\mathbf{x})$. The basis functions $T_j(\mathbf{x})$ are products of one-dimensional basis functions which are characteristic (or indicator) functions in the case of categorical variables and truncated power functions in the case of numeric variables. The key to the method, which can be viewed as a successor of the regression trees [2] is that the $T_j(\mathbf{x})$ are generated in a recursive way and depend on the data. In particular, two new basis functions are generated at each step of the form

$$\begin{aligned} T_{J+1}(\mathbf{x}) &= T_j(\mathbf{x})[(x_v - s)]_+ \\ T_{J+2}(\mathbf{x}) &= T_j(\mathbf{x})[-(x_v - s)]_+, \end{aligned}$$

where the $T_j(\mathbf{x})$, $j = 0, \ldots, J$ are the parent basis function and the $[(x_v - s)]_+$ and $[-(x_v - s)]_+$ are univariate truncated power basis functions of an appropriately chosen variable x_v which does not appear in the parent basis function. The new basis functions are selected such that the residual sum of squares corresponding to the augmented model

$$\text{RSS[model]} = \min_{a_1,\ldots,a_{J+2}} \sum_{n=1}^{N} \left[y_n - \sum_{j=0}^{J+2} a_j T_j(\mathbf{x}_n) \right]^2 .$$

is minimal. This process is terminated if no further substantial reduction in RSS[model] can be achieved in this way. After this forward selection step a further step, called backward elimination, removes unnecessary basis functions using a GCV estimate of the expected error in order to reduce over-fitting. Having built the set of tensor product basis functions $\{T_j(\mathbf{x})\}_{j=0}^{J}$, MARS obtains values for the regression coefficients $a_0, \ldots, a_J$ via linear logistic regression using the Fisher Scoring Algorithm [10].

MARS has been a highly successful data mining tool. In this project it is discussed how this tool has been improved by

- Using compactly supported B-spline basis functions
- Utilising a new scale-by-scale model building strategy
- Introducing a parallel implementation

These modifications allow the stable and fast analysis of very large data sets which occur in taxation data.

While the focus of the previous discussion has been on the modelling of risk of fraud there is a good case to be made for investigating records which are "unusual". In order to find such records one can model one of the attributes, e.g., the deductions, as a function of the other attributes. Records would be unusual if the predicted value of this attribute is far from the observed value. As such unusual records are only few it does not cost much to investigate them further and this could be a good indicator to find someone who has been attempting to "trick the system".

In Section 2 the BMARS algorithm is introduced together with the parallel implementation. The performance of the algorithm is presented in Section 3. In Section 4 the application to taxation data risk rating is discussed. A final discussion of the results is in Section 5.

2 BMARS = B-splines + MARS + Parallelisation

The MARS algorithm has been implemented using truncated power basis functions. Utilisation of such a basis may lead to ill-conditioned linear systems of equations [1], [4] which, in turn, may result in poor quality of the fit. Some other bases such as, for instance, B-splines, have better numerical properties [14]. In this section we outline an algorithm called BMARS based on the principles similar to those of the original MARS but utilising B-spline basis functions of the second order. These are piecewise linear functions fully characterised by three knots and are often referred to as *hat* functions. Although B-splines of any order can, in principle be used in BMARS, we chose basis function of the second order for the following reasons:

- utilisation of the simplest continuous B-splines significantly simplifies implementation,
- approximation with tensor products of piecewise linear functions is more resistant to the so-called end-effects [6].

Note that, although they are constructed using different bases, models produced by MARS and BMARS belong to the same space of d-variate functions piecewise linear in each variable. As is well-known, B-splines are distinct from zero only within a compact interval (a compact support property) and the length of the interval can be controlled via selection of the knot locations. We exploit this fact to introduce a new modified forward stepwise procedure which builds models in a scale-by-scale manner, where the scale is the length of the support interval.

Like the original MARS, BMARS is comprised of two modules. The first one (the forward stepwise procedure) produces a model made up of a large number of tensor product basis functions some of which may be suboptimal. The second module (backward elimination) is intended to remove the suboptimal basis functions and, thereby improve the quality of the fit.

The forward procedure of BMARS is based on the combination of the original MARS algorithm with a new scale-by-scale approach to model building. To implement this approach we use B-splines of various scales. Such B-splines can be constructed using L nested sets of knots[1] for each numeric predictor variable. For ex-

[1]The number L of the sets can be either determined based on the coin-tossing argument outlined in [6] or specified by a user.

ample, the lth set can be comprised of $(100/2^l)$th percentiles of the empirical marginal distribution of the corresponding variable. For each knot set one can construct a basis comprised of B-splines (see Figure 1) which are to be used by the algorithm. In the case of uniformly distributed predictor data points, B-splines based on the lth set have twice as long support intervals as B-splines based on the $(l+1)$ set.

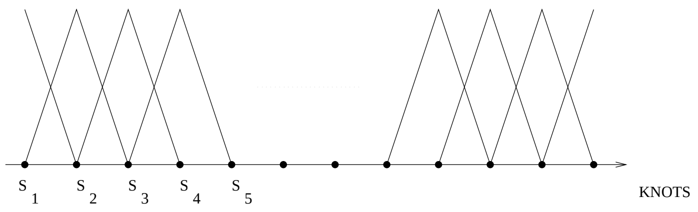

Figure 1: B-splines constructed based on a set of knots.

Like models by MARS, a model built by BMARS is comprised of tensor product basis functions

$$\hat{f}(x) = \sum_{j=0}^{J} a_j T_j(\mathbf{x}). \tag{1}$$

However, each tensor product basis function is a product of univariate B-splines rather than truncated powers:

$$T_j(\mathbf{x}) = \prod_{k=1}^{K_j} B_{s(k,j)}[x_{v(k,j)}]. \tag{2}$$

These basis functions are produced according to the following procedure (for now we assume that all the predictor variables are numeric). One starts with the model containing only the constant function

$$T_0(\mathbf{x}) = 1.$$

The next basis function is derived from $T_0(\mathbf{x})$ by

$$T_1(\mathbf{x}) = T_0(\mathbf{x}) B_s[x_v], \tag{3}$$

where $B_s[x_v]$ is a B-spline of the largest scale available for x_v, and the values for s and v are chosen to ensure the largest reduction in the value of the residual sum of squares resulting from the least squares fit of the current model to data. Note that B-splines of the largest scale only are allowed to form the new tensor product basis function $T_1(\mathbf{x})$. Having selected thus $T_1(\mathbf{x})$, the procedure continues to generate new basis functions in the same way. Specifically, after the J-th step there are $(J+1)$ functions

$$\{T_j(\mathbf{x})\}_0^J \tag{4}$$

in the model, each of the form (2) and the $(J+1)$-st step adds one new basis function such that

$$T_{J+1}(\mathbf{x}) = T_{j^*}(\mathbf{x}) B_{s^*}[x_{v^*}]. \tag{5}$$

Here $T_{j^*}(\mathbf{x})$ is one of the $(J+1)$ already chosen basis functions (4), $0 \le j^* \le J$; x_{v^*} is one of the predictor variables not present in $T_{j^*}(\mathbf{x})$; and $B_{s^*}[x_{v^*}]$ is a univariate B-spline basis function of the largest available scale. The values for the three parameters j^*, v^*, s^* defining T_{J+1} are chosen such that the residual sum of squares is minimal, i.e., as the solution of the minimisation problem

$$(j^*, v^*, s^*) = \arg\min_{j,v,s} \mathrm{RSS}[\mathrm{model} \cup T_{J+1}(\mathbf{x})], \tag{6}$$

where $T_{J+1}(\mathbf{x}) = T_j(\mathbf{x}) B_s[x_v]$ and

$$\mathrm{RSS}[\mathrm{model} \cup T_{J+1}(\mathbf{x})] = \min_{a_1,\cdots,a_{J+1}} \sum_{n=1}^{N} \left[y_n - \sum_{j=0}^{J} a_j T_j(\mathbf{x}_n) - a_{J+1} T_{J+1}(\mathbf{x}_n) \right]^2 .$$

In order to obtain the value of the functional $\mathrm{RSS}[\mathrm{model} \cup T_{J+1}(\mathbf{x})]$, one has to perform a least squares fit. The algorithm proceeds along these lines until the moment when the B-splines of the current (largest) scale are unable to improve the approximation. The splines of the largest scale are only able to approximate relatively coarse features of a regression surface. Therefore, at this stage, BMARS switches over to using B-splines of the second largest scale in (5). In order to determine when to carry out the switch, BMARS estimates the prediction error of the current model via the Generalised Cross-Validation (GCV) score [6] after adding each new basis function to the model and, as soon as the GCV score ceases to decrease, the switch occurs. The algorithm proceeds to produce new tensor product basis functions according to the formula (5), where $B_s[x_v]$ are B-splines of the second largest scale. When the splines of the second scale exhaust their potential, BMARS switches over to the third largest scale. This process of producing new basis functions as well as switching from scale to scale continues until the size of the model has reached a predetermined complexity. It is worth noting that, because each new basis function has been derived from an earlier basis function, B-splines of any available scale may, in principle, appear as factors in any basis function (2). This algorithm based on the scale-by-scale model building strategy is summarised by the pseudo-code 1.

Algorithm 1 BMARS algorithm

```
model ← {$T_0(\mathbf{x})$}, where $T_0(\mathbf{x}) = 1$
curr_scale ← largest_scale
for $j = 0$ to $J_{\max} - 1$ do
  $T_{j+1}(\mathbf{x}) = T_{j^*}(\mathbf{x})B_{s^*}[x_{v^*}]$, where $j^*, v^*, s^*$ are from (6); $B_{s^*}[x_{v^*}]$ is of curr_scale
  model ← model $\bigcup T_{j+1}(\mathbf{x})$
  if GCV fails to decrease then
    curr_scale ← next_largest_scale
  end if
end for
```

The backward elimination procedure of BMARS as well as the approach to determination of the regression coefficients $a_0, \dots, a_J$ in the final model (1) via the Fisher Scoring procedure are similar to those utilised in the original MARS [6], [7]. Note that, like the original MARS, BMARS allows for a straightforward handling of categorical variables via replacement of B-splines in (2) with appropriate indicator functions though the idea of the scale-by-scale model building is not applicable in this case. It turns out [1] that the computational complexity of the forward stepwise procedure can be estimated as

$$P_{\text{forwd}} \sim J_{\max}^2(\alpha J_{\max} + \beta)Nd. \tag{7}$$

where α and β are some parameters independent of the parameters of the problem in hand (d, N etc). Execution of the backward elimination procedure is less expensive. It is easy to show that the corresponding complexity can be evaluated as

$$P_{\text{backwd}} \sim \gamma J_{\max}^4 + \frac{1}{2} N J_{\max}^2. \tag{8}$$

where γ is a problem independent parameter. So, the total computational cost of BMARS is linear in the number of data points N as well as the number of predictor variables. This makes BMARS suitable for performing an efficient large scale data analysis often referred to as Data Mining.

Now we would like to contrast the MARS and BMARS algorithms with a standard statistical forward subset selection procedure (FSS) [11]. MARS builds a model by adding basis functions one at a step and so does the FSS algorithm. However, the former selects a new basis function from a relatively small subset of all available tensor product basis functions. Specifically, at each step such a subset is comprised of basis functions that can be derived from the ones selected earlier. Therefore, models produced by the forward stepwise procedure of MARS have a hierarchical structure. In contrast to this, the FSS algorithm may, in principle, add any tensor product basis function from the full set of basis functions at any step regardless of the type of basis functions added to the model before. As a consequence, the computational work required to carry out one step of MARS is considerably less than that of the FSS procedure and the reduction in the computational cost is especially dramatic in high-dimensional settings where the number of elements in the full set of tensor product basis functions is astronomical. Thus, the forward stepwise procedure of MARS can be regarded as an efficient version of an ordinary statistical forward subset selection (FSS) algorithm and this interpretation can be extended to BMARS as well. However, the way in which BMARS approximates the FSS algorithm is somewhat different from that of MARS. As was set out before, MARS selects its new basis functions from among the candidates derived according to the rule (1). Note that the univariate truncated powers $[\pm(x_v - s)]_+$ corresponding to all knot locations s are allowed to be the factors. The BMARS strategy goes even further in narrowing the set of the candidates. Although it uses the rule (5) (which is similar to (1)) to define the set of the candidate basis functions, at each step it restricts the scale of the B-splines allowed which, in the terms of MARS, would mean that some of the knots are skipped. Of course, using B-splines of only one scale would result in a poor accuracy of the produced models. Therefore, the algorithm was enabled to use different scales at different steps.

Although introduction of the scale-by-scale strategy resulted in a considerable reduction of the computational cost of running BMARS, it is important to explore possibilities for parallel distributed implementation of the algorithm because:

- It would allow for distribution of the computational load among several processors or computers. This would lead to a considerable reduction in the execution time of Data Mining applications.
- In Data Mining one often has to deal with large amounts of data. A parallel distributed implementation would allow for utilisation of the Random Access Memory of a number of systems which, in turn would allow one to solve larger problems than it would be possible using only one particular computational system.

Here we describe a parallel version of the BMARS algorithm intended to run on a cluster of workstations as well as on a multiprocessor system. As we saw earlier in (7), (8), the generation of tensor product basis functions accounts for the bulk of the computational cost. Therefore, we focus our attention on parallelisation of the forward stepwise module of BMARS though the similar approach can be applied to parallelisation of the backward elimination procedure as well.

In the BMARS strategy, each new basis function added to the model depends on the basis functions produced earlier. Thus one has a dependency which prohibits the parallel construction of basis functions. However, the determination of each new basis function can be done in parallel. Indeed, in order to generate a new tensor product basis function $T_{J+1}(\mathbf{x})$ (5), one has to solve the optimisation problem (6).

The solution of this problem amounts to performing a number of least squares fits for all appropriate values for the parameters l, v and s. Our parallelisation of the BMARS algorithm is based on parallel solution of each least squares fit. To perform a least squares fit we use the version of the Gram-Schmidt algorithm outlined in the Rejoinder section of [6]. According to this approach, new basis functions are orthonormalised as they are included with respect to all of those already contained in the model. Thus, the basis functions $\hat{T}_j(\mathbf{x}),\ j = 0, \ldots, J$ contained in the current model are kept orthonormal to each other. Given the current model comprised of J basis functions, the next $(J+1)$st function (5) selected is the one whose inclusion in the model results in the minimum value of the residual sum of squares (RSS). For each candidate basis function $T_{J+1}(\mathbf{x}) = T_j(\mathbf{x})B_s[x_v]$ the corresponding RSS value can be obtained as follows

$$\mathrm{RSS}[\mathrm{model} \cup T_{J+1}(\mathbf{x})] = \mathrm{RSS}[\mathrm{model}] - \frac{[(\mathbf{T}_{J+1}, \mathbf{y}) - \sum_{j=0}^{J}(\mathbf{T}_{J+1}, \hat{\mathbf{T}}_j)(\hat{\mathbf{T}}_j, \mathbf{y})]^2}{(\mathbf{T}_{J+1}, \mathbf{T}_{J+1}) - \sum_{j=0}^{J}(\mathbf{T}_{J+1}, \hat{\mathbf{T}}_j)^2}, \quad (9)$$

where RSS[model] is the RSS value for the current model comprised of J tensor product basis functions. Once the new candidate basis function $T_{J+1}(\mathbf{x})$ has been determined it is included in the model and orthogonolised with respect to the other basis functions

$$\hat{T}_{J+1}(\mathbf{x}) \leftarrow T_{J+1}(\mathbf{x}) - \sum_{j=1}(\mathbf{T}_{J+1}, \hat{\mathbf{T}}_j)\hat{T}_j(\mathbf{x})$$

and, then normalised

$$\hat{T}_{J+1}(\mathbf{x}) \leftarrow \frac{\hat{T}_{J+1}(\mathbf{x})}{[\sum_{n=1}^{N} \hat{T}_{J+1}^2(\mathbf{x}_n)]^{\frac{1}{2}}}$$

As the most intensive step of this procedure is the computation of the scalar products in (9)

$$\begin{aligned}(\mathbf{T}_{J+1}, \hat{\mathbf{T}}_j) &= \sum_{n=1}^{N} T_{J+1}(\mathbf{x}_n)\hat{T}_j(\mathbf{x}_n), \quad j = 0, \ldots, J \\ (\mathbf{T}_{J+1}, \mathbf{y}) &= \sum_{n=1}^{N} T_{J+1}(\mathbf{x}_n) y_n, \end{aligned} \quad (10)$$

the approach based on data partitioning appears to be appropriate. Assuming that our system is comprised of P processors[1], the data partitioning involves the allocation of N/P records of the data set to each processor[2] where the corresponding partial

[1]In this paper, the term processor refers either a processor of a multiprocessor system or a member of a cluster of workstations.

[2]Here we assume that all processors have the same computing power. We will discuss an unequal distribution of data in a later section.

scalar products

$$\begin{aligned}(\mathbf{T}_{J+1}, \hat{\mathbf{T}}_j)_p &= \sum_{n\in I_p} T_{J+1}(\mathbf{x}_n)\hat{T}_j(\mathbf{x}_n), \quad j = 0, \ldots, J \\ (\mathbf{T}_{J+1}, \mathbf{y})_p &= \sum_{n\in I_p} T_{J+1}(\mathbf{x}_n) y_n, \end{aligned} \tag{11}$$

are computed [13]. In the above formula I_p is an index set pointing to the portion of data stored by the pth, $p = 1, \ldots, P$ processor. It should be noted though that, in order to ensure the uniform distribution of the computational load, data records have to be assigned to processors randomly rather that in accordance with any deterministic rule. This requirement arises due to the fact that the basis functions produced by BMARS have compact support, i.e. they are non-zero only over certain subdomains of the predictor domain. Therefore, if the data is distributed among the processors in a certain deterministic way, it is easy to see that there always exists an ordering of data records (due to, for instance the way how the data was collected) which leads to all or maybe several tensor product basis functions evaluating to zero for all data points assigned to a particular processor. This, in turn means that such a processor would have very little or even no job to do because most (or even all) function values $T_{J+1}(\mathbf{x}_n)$ and $T_j(\mathbf{x}_n)$ in (11) would be equal to zero whereas other processes may have to work at full steam.

The parallel BMARS algorithm has been implemented using the Parallel Virtual Machine (PVM) software [8], [12]. Any program based on PVM runs on the so-called parallel virtual machine which is comprised of a number of networked workstations as well as multiprocessor systems. The architectures of the machines involved may differ significantly. However, this is of no concern to the user as the actual hardware is handled by PVM which emulates a computing environment similar to that of a multiprocessor system with distributed memory. Performing parts of an application run on the processors of the virtual machine and communicate via sending and receiving messages to one another. The software based on PVM is highly portable. In fact, it can run on any platform for which PVM was compiled. The parallel BMARS algorithm is based on the master-slave paradigm:

- The master component of BMARS computes partial scalar products as well as carries out various control tasks such as accumulation of the information produced by the slave components (see below), addition of new basis functions to a model, issuing instructions for the slaves etc. There is only one master process running on the parallel virtual machine at any time.

- The master process spawns a number (specified by a user) of the slave processes which compute the corresponding partial scalar products. Depending on the type of the hardware comprising the virtual machine, the slave processes may run either on the processors of a multiprocessor system or on the remote computers.

The pseudo-code 2 presents the parallel algorithm that solves the minimisation problem (6) and, thereby determines the new $(J+1)$st basis function given the current model comprised of $T_j(\mathbf{x}),\ j = 0, \dots, J$. In the next section we investigate the scalability properties of the parallel BMARS procedure.

Algorithm 2 Generation of the $(J+1)$st basis basis function by the parallel BMARS

```
Best candidate $T^{best}_{J+1}(\mathbf{x}) \leftarrow 0$
Best_RSS $\leftarrow$ RSS[model]
for For all candidates $T_{J+1}(\mathbf{x}) = T_j(\mathbf{x}) B_s[x_v]$ do
  Compute partial scalar products (PSP) (11) for $(p = 1)$
  for $p = 2$ to $P$ do
    Receive PSP values from $p$th slave processor
  end for
  Add up PSP values to obtain global scalar products (10)
  Determine RSS[model $\cup\, T_{J+1}(\mathbf{x})$] according to (9)
  if RSS[model $\cup\, T_{J+1}(\mathbf{x})$] $<$ Best_RSS then
    $T^{best}_{J+1}(\mathbf{x}) \leftarrow T_{J+1}(\mathbf{x})$
    Best_RSS $\leftarrow$ RSS[model $\cup\, T_{J+1}(\mathbf{x})$]
  end if
end for
```

3 Performance of the Parallel BMARS

To test the performance of the parallel BMARS we carried out an experiment on a 10 Processor Sun Enterprise 4000 with 167MHz UltraSparc chips and 1MB onboard caches. It had 4.75GB main memory with 2 Sun Storage Arrays giving 0.5TB disk storage connected over a fibre channel link. It was running Solaris 2.6 and the PVM 3.3 system. We applied BMARS to the analysis of a large motor vehicle insurance data set comprised of $1,601,277$ records. The graph in the Figure 2 displays the dependence of an efficiency of the algorithm on the number of processors involved. The parallel efficiency is defined as

$$\text{efficiency} = \frac{T_1}{T_P \cdot P},$$

where P is the number of processors engaged, and T_1 and T_P are execution times (for the same code) on one and P processors respectively. As can be seen, the efficiency of BMARS is quite close to 1 which is the efficiency of an ideal parallel algorithm. The overheads due to necessity to run a PVM daemon on a system as well as to the cost of the exchange of information between master and slave modules are responsible for the deviation of the actual efficiency from the ideal level. As was pointed out by one of the referees, the apparent increase in the efficiency for 4 through 8 processors could be due to the cache effects compensating for the communication overheads. It is worth mentioning that it took BMARS $\sim$ 3.5 hours to produce a model on one processor while the solution of the same problem on 9 processors took only $\sim$ 0.4 hours.

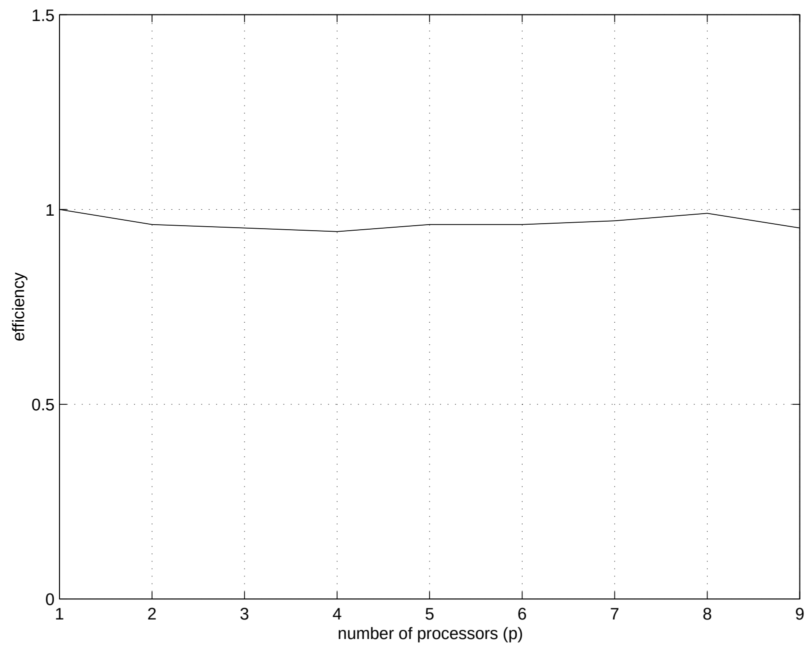

Figure 2: Efficiency of the parallel BMARS against the number of processors.

The load balancing does not seem to be the issue as long as a multiprocessor system is used because each processor of the system has the same computing power. This, however, is not the case when PVM is used to combine the computational resources of a number of networked (possibly different) computers. The next experiment is intended to illustrate this point. Here we used a cluster of four SUN workstations connected via a local area network. Each workstation had one processor[1] and ran SunOS 5.6 as well as the PVM 3.4 system. We applied the parallel BMARS to the regression analysis of a synthetic data set comprised of $300,000$ data records, each record containing values of 10 numeric predictor values as well as a response value. The parallel virtual machine used to run BMARS was configured to include all four workstations. To monitor performance of the algorithm we used the XPVM software, version 1.2.5 [9] which is a graphical console and monitor for PVM. One of the monitor functions it provides is the Utilisation View which shows the state of the virtual machine during the execution of a parallel application. It displays the

[1]The speeds of the processors of the workstations involved in the experiment were unknown.

number of processes (tasks) running on the parallel virtual machine "...in each of the three possible states: busy 'Computing', in 'PVM Overhead', or idle 'Waiting' for a message. This information is represented by vertical stacks of up to three coloured rectangles. One stack is used to represent each time instant and one rectangle is shown for each of the possible task states, with Computing on the bottom, Overhead in the middle, and Waiting on top. The height of each rectangle in the stack is proportional to the number of tasks in that state versus the total number of tasks executing at that time instant....The overall utilisation for tasks is seen by comparing the relative height of the rectangles over time, such that in cases of good utilisation the ''Computing" areas are prominent, and for poor utilisation the 'Overhead' and 'Waiting' areas dominate..." [9].

The first run of BMARS was performed with the equal number of data point being assigned to each processor (in fact, workstation) of the parallel virtual machine. The fragment of the corresponding Utilisation View is shown in the Figure 3. As can be seen, the Overhead areas were quite prominent. It turns out that the PVM routine called `pvm_barrier` [8] caused the overhead (XPVM allows one to establish that). This routine blocks the calling process until all processes have called it. This routine is used in BMARS for synchronisation purposes and is called by each process after it finishes computation of the corresponding partial scalar products. Thus, one can conclude that some of the workstations were slower than the others.

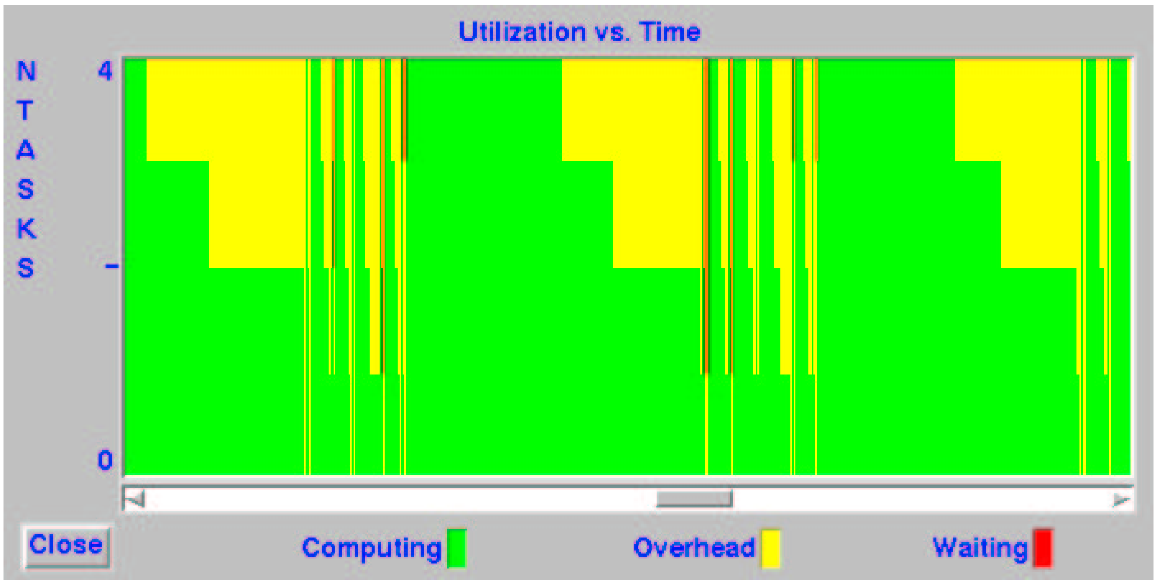

Figure 3: Utilisation View corresponding to the equal distribution of data.

XPVM offers another visualisation tool called Space-Time View which allows one to estimate the relative speeds of the workstations involved. These turned out to be 1.5 : 1.9 : 1.0 : 1.0. In the second run the number of data points assigned to a particular workstation was set to be proportional to the speed of that workstation.

The corresponding Utilisation View is shown in the Figure 4. This time the Overhead areas were considerably less prominent. The overall effect of such an unequal data distribution was also considerable: the execution times obtained via averaging over 6 independent runs were 490 and 400 seconds in the first and the second cases respectively.

The important conclusion that can be drawn from these results is that the way in which data is assigned to the processors of a virtual machine can have a significant impact on the performance of the parallel BMARS especially in the situations where the virtual machine is comprised of a number of heterogeneous systems. As was already pointed out, this is of no concern as long as a genuine multiprocessor system is used. For instance, the Figure 5 shows the Utilisation View corresponding to BMARS being applied to the analysis of the same synthetic data set and running on 3 processors of the multiprocessor system mentioned earlier. The data was distributed equally among the processors. As can be seen, the Overheads and the Waiting areas were very small which is consistent with the efficiency of BMARS being close to 1 as shown in Figure 2.

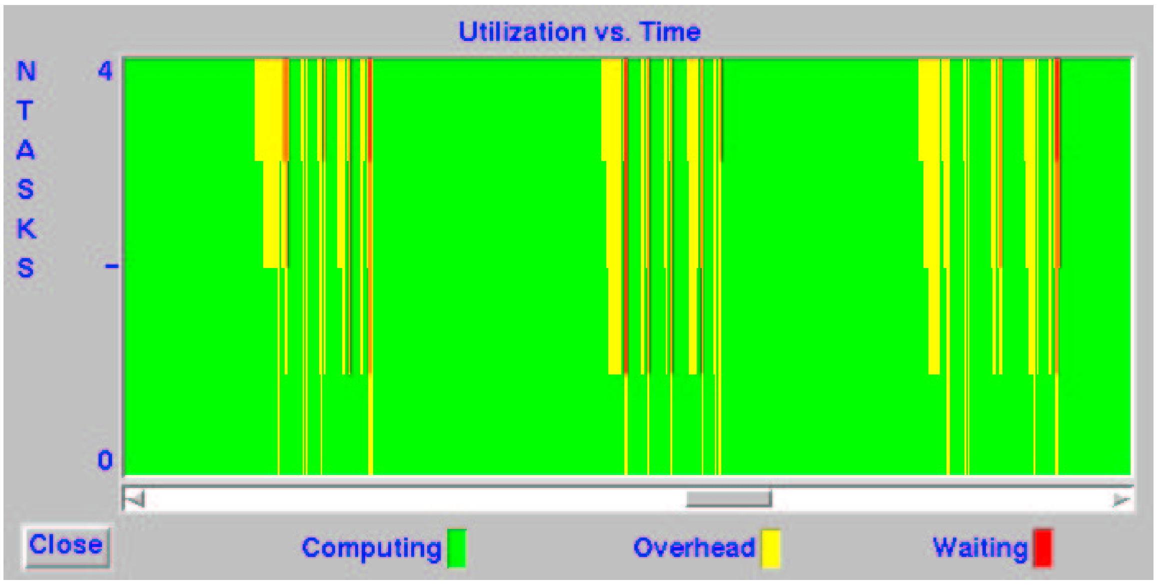

Figure 4: Utilisation View corresponding to the unequal distribution of data.

4 Data Mining for Risk Rating in Taxation

The Australian Taxation Office (ATO) is responsible for the collection of taxes in Australia. Each year some seven million individual tax returns are processed. Based on a self-assessment system there is the potential for inaccuracies to appear in the returns. Automated systems are in place to identify many types of inaccuracies. The ATO has an ongoing program to more efficiently and accurately identify tax returns

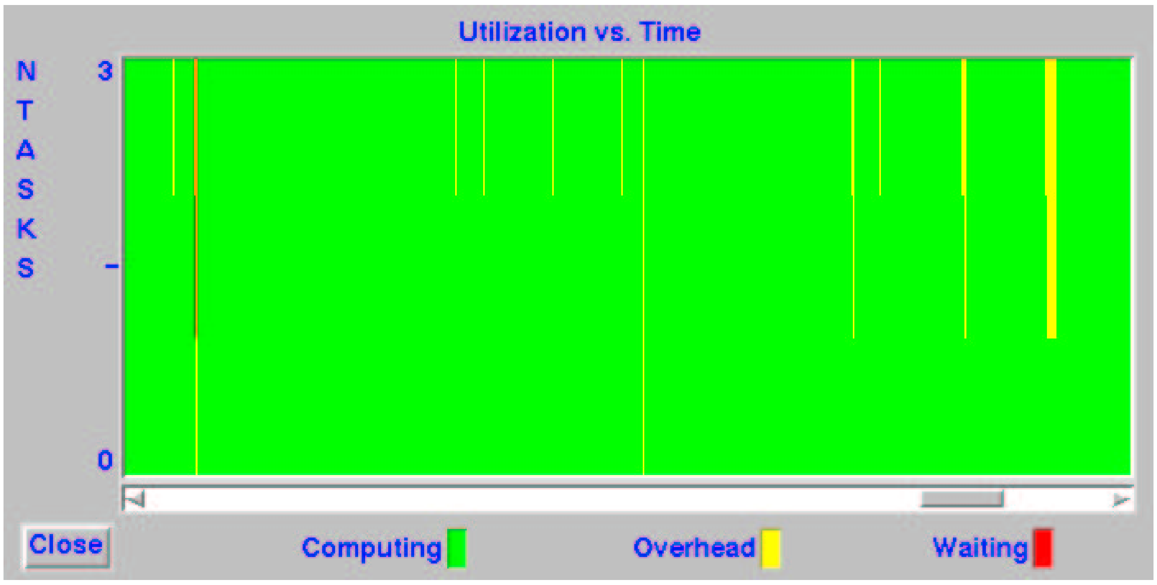

Figure 5: Utilisation View corresponding to the equal distribution of data on a multiprocessor system.

containing discrepancies. A study has explored the use of data mining [5] for this task.

As part of the data mining exercise for the Australian Taxation Office BMARS was employed to explore for relationships in the data that may provide insights into compliance. Privacy and commercial in confidence issues restrict the level of details here, but the results presented give an indication of the approach employed.

The data used for this study consisted of several hundred thousand tax returns that had been audited, and hence had available with it the outcome of the audit. The more than twenty variables selected for the study (from a pool of over 100 variables) consisted of both numeric data (e.g., income) and categorical data (e.g., occupation).

The exploratory study used BMARS to build predictive models. The application of BMARS across all variables and with all of the data leads to the identification of the most important variables and the development of a model for predicting risk. However, an important outcome of most data mining exercises is to provide insights into the models and to identify patterns of behaviour that exhibit interesting properties.

The results are in a preliminary stage, requiring validation and refinement, but nonetheless illustrate the potential application of MARS-type model building to large collections of real world data. This is made possible because of BMARS' parallel architecture and use of B-splines for improved numerical stability. After developing initial models of the whole data, and consequently identifying a number of important variables, a series of interaction models were built. The aim was to explore interactions between pairs of specific variables thought to play an important role in determining the level of risk associate with a tax return.

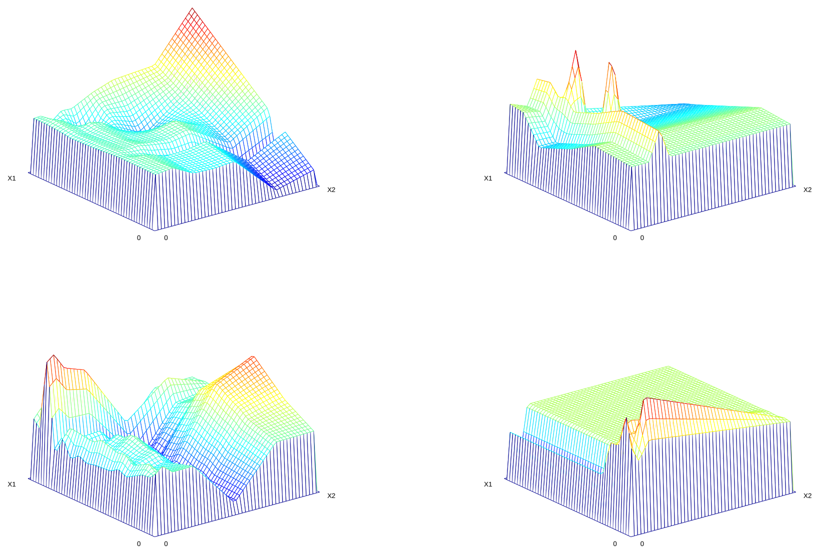

Figure 6: Sample pairwise interactions from models developed by BMARS.

Figure 6 illustrates a number of the interesting interactions identified. Again, the actual variables in the data corresponding to the X_1 and X_2 axes are not identified for reasons of privacy. The surface represents the degree of risk as modelled by BMARS, with peaks representing particularly high levels of risk and troughs representing low levels of risk.

The models built by BMARS, allow near interactive development of ideas and drive the further exploration of the data. These explorations have lead to useful insights into the characteristics of tax returns that may be indicative of fraudulent practises, or at least common errors.

5 Summary

In this paper we have presented the BMARS algorithm. This parallel algorithm has been shown to significantly shorten the task of building Multivariate Adaptive Regression Spline based models. This allows refinements to the model building process to be made interactively, and provides a useful tool for the exploration of relationships in extremely large datasets. Such activity is common in data mining and is beginning to be used in regular data mining activities involving very large, real world, datasets.

References

[1] Bakin, S., Adaptive Regression and Model Selection in Data Mining Problems, *PhD Thesis*, ANU, 1999.

[2] Breiman, L., Friedman, J.H., Olshen, R.A. and Stone, C.J., Classification and Regression Trees, Wadsworth, Belmont, California, 1984.

[3] Chen, Z., Beyond additive models: interactions by smoothing spline methods, *Technical Report*, SMS-009-90, The Australian National University, 1990.

[4] Cox, M.G., Practical spline approximation, *Topics in Numerical Analysis*, Lancaster, 1981, 79- 112.

[5] Fayyad, U. and Piatetsky-Shapiro, G. and Smyth, P., From Data Mining to Knowledge Discovery: An Overview, *Advances in Knowledge Discovery and Data Mining*, MIT Press, Cambridge, MA, 1996, 1–36.

[6] Friedman, J.H., Multivariate Adaptive Regression Splines, *The Annals of Statistics*, **19**, 1, 1991, 1–141.

[7] Friedman, J.H., Estimating functions of mixed ordinal and categorical variables, Stanford University, *Technical Report* NO.108, June 1991.

[8] Geist, A., Beguelin, A., Dongarra, J., Jiang, W., Manchek, R., PVM: Parallel Virtual Machine, MIT Press, 1994.

[9] Kohl, J.A. and Geist, G.A., XPVM 1.0 User's Guide, *Technical Report* ORNL/TM-12981, Oak Ridge National Laboratory, 1996.

[10] McCullagh, P. and Nelder, J.A., Generalized Linear Models, Chapman and Hall, 1983.

[11] Miller, A.J., Subset Selection in Regression, Chapman and Hall, 1990.

[12] PVM Home Page: `http://www.epm.ornl.gov/pvm/pvm_home.html`

[13] Osborne, M.R., Solving Least Squares Problems on Parallel Vector Computers, *Mathematics Research Report* No. MRR 043-94.

[14] Schumaker, L., Discussion: Multivariate Adaptive Regression Splines, *The Annals of Statistics*, **19**, 1, 1991, 112–113,

[15] Stone, G., Analysis of Motor Vehicle Claims Data using Statistical Data Mining, *CSIRO Technical Report*, CMIS-97/73, 1997.

[16] Wahba, G., Spline Models for Observational Data, SIAM, Philadelphia, 1990.