Scalable Parallel Algorithms for Surface Fitting and Data Mining

Peter Christen ^{a,1}, Markus Hegland ^a, Ole M. Nielsen ^a,
Stephen Roberts ^b, Peter E. Strazdins ^c and Irfan Altas ^d

^a*Computer Sciences Laboratory, RSISE, Australian National University,
Canberra, ACT 0200, Australia*

^b*School of Mathematical Sciences, Australian National University*

^c*Department of Computer Science, Australian National University*

^d*School of Information Studies, Charles Sturt University,
Wagga Wagga, NSW 2678, Australia*

Abstract

This paper presents scalable parallel algorithms for high dimensional surface fitting and predictive modelling which are used in data mining applications. These algorithms are based on techniques like finite elements, thin plate splines, wavelets and additive models. They all consist of two steps: First, data is read from secondary storage and a linear system is assembled. Secondly, the linear system is solved. The assembly can be done with almost no communication and the size of the linear system is independent of the data size. Thus the presented algorithms are both scalable with the data size and the number of processors.

Key words: Data Mining, Thin Plate Splines, Additive Models, Wavelets, Parallel Linear System Solver

1 Introduction

In the last few years there has been an explosive growth in the amount of data being collected. The computerisation of business transactions and the use of bar codes in commercial outlets has provided businesses with enormous amounts of data. In science, projects like the Human Genome Project deal with Terabytes of data. Revealing patterns and relationships in a data set

¹ Corresponding author, E-Mail: Peter.Christen@anu.edu.au

can improve the missions and objectives of many organisations and research projects. For example, sales records can reveal highly profitable retail sales patterns. As such it is important to develop automatic techniques to process and to detect patterns in very large data sets. This process is called *Data Mining* [6].

Algorithms applied in data mining have to deal with two major challenges: Large data sets and high dimensions (many attributes). In recent years, data sets had the size of Gigabytes, but Terabyte data collections are now being used in business and the first Petabyte collections are appearing in science [11]. It has also been suggested that the size of databases in an average company doubles every 18 months [5] which is similar to the growth of hardware performance according to Moore's law. Consequently, data mining algorithms have to be able to scale from smaller to larger data sizes when more data becomes available. The complexity of data is also growing as more attributes tend to be logged in each record. Data mining algorithms must, therefore, be able to handle high dimensions in order to process such data sets.

This combination of large data size with high data complexity poses a tough challenge for all data mining algorithms. Moreover, algorithms which do not scale linearly with data size are not feasible. Parallel processing can help to tackle larger problems and to get reasonable response times. In this paper, we present scalable parallel algorithms for predictive modelling and high dimensional surface fitting that successfully deal with these issues and are being applied to real world data mining problems where data sets consist of up to 36 million records with a dimensionality as large as 44 attributes.

An important technique applied in data mining is multivariate regression, which is used to determine functional relationships in high dimensional data sets. A major difficulty which one faces when applying nonparametric methods is that the complexity grows exponentially with the dimension of the data set. This has been called the *curse of dimensionality* [15]. In Section 2 we introduce our basic algorithms and explain how they deal with this curse. More detailed descriptions of three methods are then given in Sections 3, 4 and 5. A discussion of the complexity and scalability aspects of our algorithms is given in Section 6. Before we present implementation details in Sections 8 and 9 we give a short overview of the data mining process in Section 7. Section 10 finalises the paper with our conclusions and presents some ideas for future work.

2 A Finite Element Approach to Data Mining

An important task in data mining is predictive modelling. As a predictive model in some way describes the average behaviour of a data set, one can use it to find data records that lie outside of the expected behaviour. These *outliers* often have simple natural explanations but, in some cases, may be linked to fraudulent behaviour.

A predictive model is described by a function $y = f(x_1, \dots, x_d)$ from the set, T , of attribute vectors of dimension d into the response set, S . If S is a finite set (often $S = \{0, 1\}$), the determination of f is a *classification problem*, and if S is the set of real numbers, one speaks of *regression*. In the following it will mainly be assumed that all the attributes x_i as well as y are real values and we set $\mathbf{x} = (x_1, \dots, x_d)^T$.

In many applications, the response variable y is known to depend in a smooth way on the values of the attributes, so it is natural to compute f as a least squares approximation to the data with an additional smoothness component imposed. In this paper, we state the problem formally as follows. Given n data records $(\mathbf{x}^{(i)}, y^{(i)})$, $i = 1, \dots, n$ where $\mathbf{x}^{(i)} \in \Omega$ with $\Omega = [0, 1]^d$ (the d -dimensional unit cube), we wish to minimise the following functional subject to some constraints:

$$J_\alpha(f) = \sum_{i=1}^n (f(\mathbf{x}^{(i)}) - y^{(i)})^2 + \alpha \int_\Omega |\mathcal{L}f(\mathbf{x})|^2 d\mathbf{x} \quad (1)$$

where α is the smoothing parameter and $\mathcal{L}$ is a differential operator whose different choices may lead to different approximation techniques. For example, for $d = 2$, the standard thin plate splines method can be obtained by selecting $\mathcal{L}$ as the vector of the second partial derivatives of $f(x)$. The smoothing parameter α controls the trade-off between smoothness and fit: In the limit $\alpha \rightarrow 0$ the function f becomes an interpolant. If α is large, f becomes very smooth but may not reflect the data very well. The choice of α is therefore data dependent. Some techniques such as generalised cross validation can be employed to pick an appropriate value of α for a given data set. A more detailed discussion on the choice of α can be found in [27].

One can choose different function spaces to approximate the minimiser f of equation (1). For example, picking the approximating functions in the space of piecewise multi-linear functions requires the use of a non-conforming finite element approach for the Thin Plate Splines Finite Element Method as discussed in Section 3, and requires the addition of some constraints for Additive Models as discussed in Section 5. As such we generally need to consider the addition of constraints when we discretise the minimisation problem (1).

In this paper, we describe and compare three different methods to approximate the minimiser of Equation (1). These methods are:

- **TPSFEM**: Piecewise multilinear finite elements (Section 3)
- **HISURF**: Interpolatory wavelets (Section 4)
- **ADDFIT**: Additive models (Section 5)

These methods have been implemented as data mining tools and are used for predictive modelling. All of these tools approximate the minimiser in Equation (1), but they differ in how well they approximate f and more importantly in their algorithmic complexities. Roughly speaking, TPSFEM gives the most accurate approximation at the highest computational cost whereas ADDFIT has the lowest cost but the coarsest approximation. HISURF sits somewhere in between these two extremes and provides good approximations at a reasonable cost. Details about these methods are given in Sections 3 to 5.

For the TPSFEM and ADDFIT methods we use piecewise multi-linear basis functions to approximate f , where the vector of basis functions will be denoted by $\mathbf{h}(\mathbf{x})$. That is, in vector form,

$$f(\mathbf{x}) = \mathbf{h}(\mathbf{x})^T \mathbf{c} \quad (2)$$

where typically the vector $\mathbf{h}(\mathbf{x})$ is sparse and the vector $\mathbf{c}$ represents the linear combination coefficients. In the HISURF method f is approximated with wavelet functions as explained in Section 4.

In the case of $\alpha = 0$ the matrix of the normal equations relating to (1) is

$$\mathbf{A} = \sum_{i=1}^n \mathbf{h}(\mathbf{x}^{(i)}) \mathbf{h}(\mathbf{x}^{(i)})^T \quad (3)$$

and the right-hand side vector of the normal equations is

$$\mathbf{b} = \sum_{i=1}^n \mathbf{h}(\mathbf{x}^{(i)}) y^{(i)}. \quad (4)$$

These sums are evaluated by reading the data records from disk and computing the sum which is then accumulated into global data structures. Thus, the formation of $\mathbf{A}$ and $\mathbf{b}$ is scalable with respect to the data size. Also, the operation parallelises well because different segments of the data can be processed independently.

The approximation of Equation (1) together with the additional constraints leads to a linear system of equations of the form

$$\begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{B}^T & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{c} \\ \mathbf{w} \end{bmatrix} = \begin{bmatrix} \mathbf{b} \\ \mathbf{0} \end{bmatrix}. \quad (5)$$

The size of this system is independent of the number of data records, n , but depends on the number of basis functions, γ , used for the discretisation. The explicit form of such a system is presented in the following Section.

Our three methods have two main steps to approximate f in Equation (1):

- 1 **Assembly:** The $m \times m$ matrix $\mathbf{A}$ and the $m \times 1$ vector $\mathbf{b}$ are assembled by using Equation (3) and Equation (4). This step requires access to all n data records and it can be organised such that the amount of computational work is linear in n . As usually $m \ll n$ this step can be interpreted as a reduction operation on the original data. Note that the construction of the matrices coming from the smoothing part of Equation (1) and constraints do not require accessing the data at all. These matrices have similar sizes to $\mathbf{A}$. Hence, the costs related to the assembly of these matrices are negligible.
- 2 **Solving:** This step assembles the $m \times m$ matrices coming from the smooth part of Equation (1) and solves the entire system. This step does not involve the n data records and the computational work depends only on m , typically as $O(m^3)$.

Note that for large n step 1 will dominate whereas for large m step 2 will dominate. As the number of data records, n , is usually very large for data mining applications, the overall complexity is mainly determined by n .

3 Surface Fitting and Thin Plate Splines

In this section we describe the Thin Plate Spline Finite Element Method (TPSFEM). The fitting of a function to a set of data records is a recurring problem across numerous disciplines such as analysing meteorological data, reconstructing three dimensional scanned images and regression type analyses of very large data sets in the data mining area. One of the most frequently used and efficient techniques is the thin plate splines approach. The standard thin plate spline is the function that minimises the functional

$$J_\alpha(f) = \sum_{i=1}^n (f(\mathbf{x}^{(i)}) - y^{(i)})^2 + \alpha \int_{\Omega} \sum_{|\boldsymbol{\nu}|=2} \binom{2}{\boldsymbol{\nu}} (D^{\boldsymbol{\nu}} f(\mathbf{x}))^2 d\mathbf{x} \quad (6)$$

where $\boldsymbol{\nu} = (\nu_1, \dots, \nu_d)$ is a d dimensional multi-index and $|\boldsymbol{\nu}| = \sum_{s=1}^d \nu_s$. The case $d = 1$ corresponds to the natural cubic spline smoother and the case

$d = 2$ to the thin plate spline smoother. For $d > 2$ the smoothing term in the functional can be extended to

$$\int_{\Omega} \sum_{|\boldsymbol{\nu}|=2} \binom{2}{\boldsymbol{\nu}} (D^{\boldsymbol{\nu}} f(\mathbf{x}))^2 d\mathbf{x} + \int_{\Omega} \sum_{|\boldsymbol{\nu}|_{\infty}=1} (D^{\boldsymbol{\nu}} f(\mathbf{x}))^2 d\mathbf{x} \quad (7)$$

which ensures that pointwise evaluation of functions is well defined on the Hilbert space defined by the boundedness of the functional. However, with this extension rotation invariance of the functional is lost.

An explicit representation of the thin plate spline as a sum of radial basis functions was obtained by Duchon [10]. This approach requires the solution of a symmetric indefinite dense linear system of equations that has a size proportional to the number of data records, n . Although this initial approach was improved later [4,24], these techniques require complex data structures and algorithms and $O(n)$ workspace. Thus, standard thin plate splines may not be practical for applications that have very large data sets which is the case for data mining.

In this section we present a discrete thin plate spline which combines the favourable properties of finite element surface fitting with the ones of thin plate splines. The mathematical foundations of the method are briefly presented here. The details of the method were given in [17].

The smoothing problem can be approximated by minimising the functional over a finite dimensional space of piecewise multi-linear functions, yielding a solution of the form given in (2).

The idea is to minimise J_{α} over all f of this form. Unfortunately, the smoothing term in Equation (6) is not defined for piecewise multi-linear functions. We can either work with a more complicated set of approximation functions or we can use a non-conforming finite element principle, and introduce piecewise multi-linear functions to represent the gradient of f . We have chosen the latter.

Let $\mathbf{u}(\mathbf{x}) = (\mathbf{h}(\mathbf{x})^T \mathbf{g}_1, \dots, \mathbf{h}(\mathbf{x})^T \mathbf{g}_d)$ denote the approximate gradient of f . The functions f and $\mathbf{u}$ satisfy the relationship

$$\int_{\Omega} \nabla f(\mathbf{x}) \cdot \nabla v(\mathbf{x}) d\mathbf{x} = \int_{\Omega} \mathbf{u}(\mathbf{x}) \cdot \nabla v(\mathbf{x}) d\mathbf{x} \quad (8)$$

for all piecewise multi-linear functions v . This is equivalent to the relationship

$$\mathbf{L}\mathbf{c} = \sum_{s=1}^d \mathbf{G}_s \mathbf{g}_s \quad (9)$$

where $\mathbf{L}$ is a discrete approximation to the minus Laplacian operator and

$$(\mathbf{G}_1, \dots, \mathbf{G}_d)$$

is a discrete approximation to the gradient operator. We now consider the minimiser of the functional

$$\begin{aligned}
J_\alpha(\mathbf{c}, \mathbf{g}_1, \dots, \mathbf{g}_d) &= \sum_{i=1}^n (\mathbf{h}(\mathbf{x}^{(i)})^T \mathbf{c} - y^{(i)})^2 + \\
&\quad \alpha \int_{\Omega} \sum_{s=1}^d \nabla(\mathbf{h}(\mathbf{x})^T \mathbf{g}_s) \cdot \nabla(\mathbf{h}(\mathbf{x})^T \mathbf{g}_s) d\mathbf{x} \\
&= \sum_{i=1}^n (\mathbf{h}(\mathbf{x}^{(i)})^T \mathbf{c} - y^{(i)})^2 + \alpha \sum_{s=1}^d \mathbf{g}_s^T \mathbf{L} \mathbf{g}_s.
\end{aligned} \tag{10}$$

Our smoothing problem now consists of minimising this functional over all vectors $\mathbf{c}, \mathbf{g}_1, \dots, \mathbf{g}_d$ subject to the constraint (9).

The functional has exactly one minimum if the observation points $\mathbf{x}^{(i)}$ are not collinear. The function defined by $f(\mathbf{x}) = \mathbf{h}(\mathbf{x})^T \mathbf{c}$ provides a smoother which has essentially the same smoothing properties as the original thin plate smoothing spline, provided the discretisation is small enough.

For the rest of this section we will consider the case $d = 2$, but other cases of d are completely analogous. The discrete minimisation problem (10) is equivalent to the linear system

$$\begin{bmatrix} \mathbf{A} & \mathbf{0} & \mathbf{0} & \mathbf{L}^T \\ \mathbf{0} & \alpha \mathbf{L} & \mathbf{0} & -\mathbf{G}_1^T \\ \mathbf{0} & \mathbf{0} & \alpha \mathbf{L} & -\mathbf{G}_2^T \\ \mathbf{L} & -\mathbf{G}_1 & -\mathbf{G}_2 & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{c} \\ \mathbf{g}_1 \\ \mathbf{g}_2 \\ \mathbf{w} \end{bmatrix} = \begin{bmatrix} \mathbf{b} \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{11}$$

where $\mathbf{w}$ is a Lagrange multiplier associated with the constraint (9). The matrices $\mathbf{L}$, $\mathbf{G}_1$ and $\mathbf{G}_2$ are independent of the data, and the matrix $\mathbf{A}$ and the vector $\mathbf{b}$ as defined in the previous section depend on the data. The assembly of these matrices will be discussed in Section 8. The matrix $\mathbf{A}$ is symmetric indefinite and in this case, sparse.

If γ_s is the size of the discretisation in dimension s , then the total number of basis functions for TPSFEM is $m = \prod_{s=1}^d \gamma_s$. If $\gamma_s = \gamma$ for all s then

$$m = \gamma^d.$$

There are many possible ways to solve equation (11). In particular, it is possible to eliminate all the variables except $\mathbf{g}_1$ and $\mathbf{g}_2$. The equations for $\mathbf{g}_1$ and

$\mathbf{g}_2$ then become

$$\begin{bmatrix} \alpha \mathbf{L} + \mathbf{G}_1^T \mathbf{Z} \mathbf{G}_1 & \mathbf{G}_1^T \mathbf{Z} \mathbf{G}_2 \\ \mathbf{G}_2^T \mathbf{Z} \mathbf{G}_1 & \alpha \mathbf{L} + \mathbf{G}_2^T \mathbf{Z} \mathbf{G}_2 \end{bmatrix} \begin{bmatrix} \mathbf{g}_1 \\ \mathbf{g}_2 \end{bmatrix} = \begin{bmatrix} \mathbf{G}_1^T \mathbf{z} \\ \mathbf{G}_2^T \mathbf{z} \end{bmatrix} \quad (12)$$

where $\mathbf{Z} = \mathbf{L}^{-1} \mathbf{A} \mathbf{L}^{-1}$ and $\mathbf{z} = \mathbf{L}^{-1} \mathbf{b}$. Strictly speaking, $\mathbf{L}^{-1}$ does not exist, but we can define $\mathbf{c} = \mathbf{L}^{-1} \mathbf{b}$ to be that vector $\mathbf{c}$ which satisfies the augmented system

$$\begin{bmatrix} \mathbf{L} & \mathbf{e} \\ \mathbf{e}^T & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{c} \\ g \end{bmatrix} = \begin{bmatrix} \mathbf{b} \\ 0 \end{bmatrix}.$$

In fact, in our code, all calculations involving $\mathbf{L}^{-1}$ are provided by a multigrid Poisson solver.

It can be seen that the system (12) is symmetric positive definite and so we use the conjugate gradient method to solve the equation. In fact we use the preconditioned conjugate gradient (PCG) method, with preconditioner matrix

$$\mathbf{Q} = \begin{bmatrix} \mathbf{L}^{-1} & \mathbf{0} \\ \mathbf{0} & \mathbf{L}^{-1} \end{bmatrix}.$$

Each iteration of our PCG solver involves four applications of $\mathbf{L}^{-1}$. For smooth problems we need on the order of twenty iterations of our PCG method to reduce the residual of the equation by a factor 10^{-10} .

The details of this approach, the solution techniques of the system and parallel implementation of it are presented in [8]. One of the advantages of this approach is that the size of the linear system depends on the grid size and not on the number of data records n .

The TPSFEM method can be used for surface fitting problems with dimensions 1 to 3. For problems with higher dimensions the curse of dimensionality becomes an obstacle due to exponential growth of the dimension of the finite element space and the need to use more complicated smoothing terms. However, TPSFEM can still be employed to handle the surface fitting problems with more than 3 variables by combining TPSFEM with the methods discussed in Sections 4 and 5.

4 High Dimensional Smoothing Using Wavelets

This section describes the High Dimensional Surface Smoothing (HISURF) method. It uses a hierarchical interpolatory wavelet basis and tensor prod-

ucts thereof to approximate f as given in (1). Wavelets provide a multi-level decomposition of f in each dimension which can lead to very compact approximations of reasonable well behaved functions. See [9,23,25] for introductions to wavelet theory.

Let $I_{j,l} = [\frac{l}{2^j}, \frac{l+1}{2^j}]$ be an interval. The one dimensional hierarchical basis functions are then

$$\sigma_{j,l} = \begin{cases} \varphi_{0,l}, & j = 0 \\ \psi_{j-1,l}, & j > 0 \end{cases} \quad (13)$$

where

$$\varphi_{j,l}(x) = \begin{cases} 1 + 2^j x - l, & x \in I_{j,l-1} \\ 1 - 2^j x + l, & x \in I_{j,l} \\ 0, & \text{otherwise} \end{cases}$$

and $\psi_{j-1,l} = \varphi_{j,2l+1}$, $l = 0, \dots, 2^j$. The high dimensional basis functions are formed as the tensorial multi dimensional wavelet functions $\sigma_{\mathbf{j},\mathbf{l}} = \bigotimes_{s=1}^d \sigma_{j_s,l_s}$ where $\mathbf{j} = (j_1, \dots, j_d)^T$ is a vector of scales in each dimension and $\mathbf{l} = (l_1, \dots, l_d)^T$ is a vector of positions in each dimension.

Let $j \in \mathbb{Z}$ be a fixed maximal resolution, let $\gamma = 2^j + 1$ be the number of grid points in each dimension, and let U_j^d be the space generated by piecewise linear d -dimensional functions interpolated from the γ^d total grid points. These functions are the generic finite element basis used in the previous section. However, it can be shown (see [19]) that the functions $\sigma_{\mathbf{j},\mathbf{l}}$ also form a basis for U_j^d . We call this basis the rectangular wavelet basis.

A function $f(x_1, \dots, x_d) \in U_j^d$ then has an expansion in terms of the rectangular basis as follows:

$$u = \sum_{j_1, \dots, j_d=0}^j \sum_{l_1=0}^{\theta(j_1)} \cdots \sum_{l_d=0}^{\theta(j_d)} d_{\mathbf{j},\mathbf{l}} \sigma_{\mathbf{j},\mathbf{l}} \quad (14)$$

where the function $\theta(i)$ denotes the last local index of coefficients at scale i , defined as $\theta(i) = 2^{i-1} - 1$, $i \geq 0$.

It can be shown [19] that terms of this expansion where $j_1 + \dots + j_d > j$ can be deactivated without sacrificing the essential approximation power.

The compression error is bounded by the expression $\text{const} \binom{j+d-1}{j} 2^{-2j}$ where the constant depends only on the smoothness of f . In return, the dimension m of the compressed system is bounded by $j^{d-1}(2^{j+1} - 1)$, which is a significant reduction compared to TPSFEM, especially for large j and d . The approximated smoothing surface is then computed in terms the active coefficients only.

Fig. 1. Comparison of four different coefficient matrices, $d = 4, j = 2$.

This is data *independent* or a-priori compression as opposed to the more common (in the wavelet literature) data dependent compression where wavelet coefficients are discarded based on their magnitude. The data dependent compression is efficient for a function with isolated singularities. On the other hand, for fitting a high dimensional smooth surface, singularities are unlikely to occur so good approximations can be achieved by using a-priori compression. Since this compression scheme is data-independent the algorithm can be very fast.

Figure 1 shows the number of nonzero entries, nz , of four different matrices for the smoothing problem with $d = 4, j = 2$. The upper left panel shows it in terms of the generic basis for U_j . It is a large banded matrix with bandwidth increasing with d . The upper right panel shows the coefficient matrix in terms of the rectangular wavelet basis for U_j . Because of scale interactions, the number of nonzeros has increased. The lower left panel shows the same matrix with inactive coefficients set to zero, using only active basis functions. The lower right panel shows the compressed matrix where all inactive rows and columns have been removed. This is the matrix used to compute the compressed surface in HISURF.

Fig. 2. Modelling 2D aeromagnetic data with 735,700 records.

5 Additive Models

In this section we describe our method for Additive Model Fitting called ADDFIT. Functions of d variables can be represented as sums of the form

$$f(x_1, \dots, x_d) = f_0 + \sum_{i=1}^d f_i(x_i) + \sum_{i < j} f_{i,j}(x_i, x_j) + \dots$$

Such decompositions originate from the Analysis of Variance and have thus been called *ANOVA-decompositions* [12]. They can be viewed as generalisations of Taylor and Fourier-series. However, the terms are only uniquely determined if additional constraints are imposed. If this is not done, the component $f_1(x_1)$, for example, is a special case of $f_{1,2}(x_1, x_2)$ and thus cannot be determined.

Including so-called *interaction terms* $f_{i_1, \dots, i_k}(x_{i_1}, \dots, x_{i_k})$ up to order $k = d$ allows the exact representation of f . This, however, is computationally infeasible in general due to the *curse of dimensionality* which also posed a major challenge to TPSFEM and HISURF. Luckily, for high-dimensional data only the inclusion of lower-order terms is required as they give approximations which converge with the dimension d for smooth functions [18]. Practical algorithms [3,13,15,27] typically give good approximations for $k = 1$ or $k = 2$ and it is common folklore that interactions with higher order than $k = 5$ are highly unusual. (Of course, one also requires enough data in order to identify such high-dimensional interactions.) The terms in the ANOVA decomposition are represented using the same basis functions which have been used for TPSFEM and HISURF. The more general cases will be discussed elsewhere, here we only discuss functions of the form

$$f(\mathbf{x}) = f_0 + \sum_{s=1}^d f_s(x_s).$$

These *additive models* are discussed extensively in [15] from a statistical view-

Fig. 3. Elementary matrix structure, $d = 3$

point. The predictor variables x_s can be real numbers, categories or even more complex objects like sets, graphs and vectors. (Vectors allow the inclusion of higher order interactions.) In the following, however, only simple data types (real and categorical) will be discussed. Additive models have many advantages. They are easy to interpret as the overall effect is given as a sum of effects of single variables. When interpreting additive models, however, one has to take into account that the variables x_s might be correlated. Our implementation of additive models uses a basis representation of the component functions f_s as

$$f_s(x_s) = \sum_{i=1}^{\gamma_s} c_{s,i} b_{s,i}(x_s)$$

with the basis functions $b_{s,i}$, the coefficients $c_{s,i}$ and where γ_s is the number of basis functions characterising f_s . The basis functions are such that for any x_s only a small number of basis functions have nonzero values. For the categorical variables the basis functions are just the category indicator functions and for real variables we use piecewise linear functions. A difficulty is that these functions may be linearly dependent over the data set and this has to be addressed with constraints.

While the components of the sum defining $\mathbf{A}$ in Equation (3) are typically sparse (see Figure 3), the final matrix $\mathbf{A}$ is more or less dense, as can be seen in Figure 4. In the parallel implementation each processor needs storage of the size of $\mathbf{A}$ to store the partial sum. This means that we will not be able to increase the accuracy of the model in a scalable way with the numbers of processors, but this was not essential for our project. The accuracy of the estimate is given by the accuracy of the model (bias) and the variance of the estimate. The number of data records controls the variance of the estimate.

The total number of basis functions used is $m = 1 + \sum_{s=1}^d \gamma_s$. Thus, for constant $\gamma_s = \gamma$ the number of unknowns scales linearly with the dimension d ,

$$m = 1 + d\gamma.$$

Fig. 4. Matrix structure and complexity for a 3D data set

This scalability with dimension d is an import advantage of the additive model.

Figure 4 shows clearly that additive models lead to small dense linear systems which makes high dimensional modelling feasible. The somewhat poor approximation can be improved by introducing derived variables. These new variables should be chosen to suit particular applications. For higher dimensional problems it can be shown that the approximation power gets better with increasing number of dimensions [18].

6 Complexity and Scalability

The process of assembling the linear systems has the same structure for all three presented methods. For each data record, the elementary vector $\mathbf{h}(\mathbf{x})$ is computed. The number of nonzero elements in this vector is $O(d)$. Depending on the method each variable contributes only a small number of nonzero elements to this vector. Forming the normal equations matrix $\mathbf{A}$ is thus of

order $O(d^2)$ for each data record, and the total complexity of assembling n data records sequentially becomes

$$T_{assem}(1) = O(d^2 n).$$

As each processor can assemble a local linear system reading a fraction n/p of the data set without communication, the parallel complexity of the assembly step on p processors becomes

$$T_{assem}(p) = O\left(\frac{d^2 n}{p}\right).$$

Before it can be solved, the local linear systems have to be summed into the final linear system. The algorithm used to solve this system not only depends on the chosen method but also on the size of the assembled linear system. If the size is below a given threshold (which depends on the parallel hardware) we collect and sum the local linear systems on one processor and solve it sequentially. A reduction operation is needed in this case, which has a complexity of $O(\log(p)d\,m)$ for TPSFEM and $O(\log(p)m^2)$ for HISURF and ADDFIT, respectively. If we aim to solve the linear system in parallel, we can use a block-cyclic distribution which has the same complexity as the reduction operation, but the scalability is better as none of the processors becomes a bottleneck. For a scalable algorithm for very large p the size of the matrix would actually have to decrease like $\sqrt{\log(p)}$, however, this decrease is not thought to be too dramatic with the current numbers of processors available. The time for reducing or redistributing the linear system is still much smaller than the time to assemble the local linear systems as we deal with a very large number of data records n .

From the storage point of view each processor has to store the whole matrix, as every data record can contribute nonzero elements anywhere in the matrix. As can be seen from Figure 4 the matrix data structure depends on the chosen method. For TPSFEM, only a small number of diagonals are filled with nonzero elements. For a d dimensional data set 3^d diagonals will be filled, but as the matrix is symmetric only

$$\frac{3^d + 1}{2}$$

need actually to be stored. Using HISURF we get a dense matrix because of scale interactions, and for ADDFIT a block matrix with d^2 blocks is assembled. As all off-diagonal blocks can get dense in this case, we treat the complete matrix as dense. The size of the linear system also depends on the used method. Although TPSFEM results in a sparse matrix (which gets more and more dense with increasing d) the method is not scalable with respect to the number of dimensions as can be seen from the lower right panel of Figure 4.

Fig. 5. The Data Mining Process

7 The Data Mining Process

Before we start describing the implementation of our algorithms we give a short introduction to the process of data mining [7] as presented in Figure 5. A data mining project usually starts with a customer who has a large database and who wants this data to be analysed in order to find some valuable information that is previously unknown and hidden in the data.

The first steps in data mining – after understanding what a customer wants – are the understanding and preprocessing of suitable data out of the given data collections. Once data is understood and available, it can be used by modelling algorithms like neural networks, decision trees, association rules or predictive models as presented in this paper. The results of this modelling process then have to be evaluated and deployed at the customer’s organisation. As Figure 5 shows data mining is not a linear process, some steps usually need to be done several times until desired results are achieved. Different modelling techniques, for example, require different data preparations.

Figure 6 shows the steps that are actually addressed by our data mining algorithms. The main purpose of a data collection is usually not data mining. Therefore, a suitable subset of the data often has to be extracted and converted so it can be used for further analysis by data mining tools. This extraction may include choosing a subset of attributes and/or a subset of records. Data can be noisy, so it has to be cleaned. Missing data has to be converted into some well defined values so a modelling technique can deal with them. This data preparation can also include a normalisation or conversion of data into numerical values (e.g. if an algorithm only works with numerical data).

For our data mining algorithms we need to convert categorical attributes into category numbers and normalise continuous attributes into the interval $[0, 1]$. This preprocessing has to be done once only for a given data set. A straightforward parallelisation distributes the attributes on processors. Unfortunately this has two drawbacks. First, every processor still has to read the whole data file (assumed all attributes are stored in a single file) and secondly load imbalance may occur if some processors have to process more attributes than others. Moreover, the time to process categorical attributes depends heavily

Fig. 6. Steps of our Data Mining Algorithms

on the number of categories, as for each data record a category has to be checked against all already stored categories.

To our disposal we had Australian Medicare data collections containing almost 56 million transactions and 10 attributes (two continuous and eight categorical). The aim of the project was to explore this data collection and link it with data about medication prescription. Due to time and privacy constraints we were not able to test our predictive modelling techniques on this data sets, so we had to create synthetic data with similar properties for testing purpose. All results presented in this and the following section are based on this synthetic data sets.

Table 1 presents timings we measured on a Sun Enterprise 4500 multiprocessor with twelve 400 MHz Ultra-Sparc processors, 6.75 Gigabytes of main memory and an attached 256 Gigabyte RAID disk array. As can be seen there was quite a difference between the first and last processor to finish, which is due to one categorical attribute that had 885,351 categories. Since the number of categories is often unknown prior to preprocessing, it is hard to achieve good load balancing. For continuous – as well as categorical attributes with only a small number of categories – good load balancing can be achieved, as the time in such a case is bounded by the time to read and parse the (text) file from secondary storage.

Table 1

Preprocessing times in seconds for 56 million records

	Number of processors					
	1	2	4	6	8	10
First processor	3,328	1,865	1,054	827	829	802
Last processor	–	2,027	1,551	1,367	1,342	1,067

Once the binary normalised files are available we can run our predictive mod-

elling algorithms with different parameters (grid resolution, smoothing parameter, choosing predictor and response variable out of the set of attributes, etc.) until a desired model is achieved.

The preprocessing and visualisation steps finally include a suitable presentation of the achieved results. This can either be a printed report describing the outcomes in a customer related language or it can be an interactive tool where a customer can interactively run the models and explore the data.

Until now we have used our predictive modelling techniques in various research projects. Within the already mentioned Medicare data mining project [16] we used additive models to predict the number of tests ordered by a pathology laboratory, based on a given laboratory code, a region code and a provided neural network classification of pathology laboratories. We also modelled the probability that a doctor-laboratory pair would order more tests than the average, and we found that this probability was increased by a factor of over three for certain laboratories. Within the MACHO (MAssive Compact Halo Objects), a collaboration between Australian and US astronomers, our additive modelling techniques are being used to classify observations of stars according to their varying intensity [20]. Finally, an ongoing project in collaboration with a group of computational chemists at the Australian National University applies our predictive modelling approach to protein secondary structures (no results are available yet).

8 Assembling the Linear System

Once a data set has been preprocessed it is available as binary files on a secondary storage. The first step in our algorithms, which requires access to the data, is the assembly of a linear system. For each method a different linear system is assembled, but the structure of the assembly remains the same. Basically, each data record requires the addition of some values into the linear system according to Equation 3 and Figure 3.

The assembly of data records into the linear system is additive and thus each data record can be assembled independently from all others. Therefore, parallelisation can be achieved in a way that each processor assembles a fraction n/p of all n data records into a local linear system. Load balancing in the assembly is easily achieved, as each processor can read and assemble the same number of data records. However, if the processors are loaded differently, the most loaded processor may become a bottleneck.

The assembly has been implemented for the ADDFIT method using C and MPI [22]. As the basic assembly structure is the same for all three methods,

Fig. 7. Assembly on Sun Enterprise 4500 multiprocessor

it is simple to include new routines for TPSFEM and HISURF. The matrix data structures and the assembly of a data record are the only parts that have to be changed. All results presented in this paper are based on the assembly of a small (dimension less than 100) linear system for ADDFIT.

We implemented both an SPMD and a master-worker version of the assembly. In the SPMD implementation each processor computes the partition of a file it has to assemble. It then opens the file, seeks to the start position, reads the appropriate number of data records and assembles them into a local linear system. In the master-worker implementation the master sends short messages to workers, containing the file name, the start position and the number of records a worker has to assemble. A worker then opens a file, seeks to the given position and reads and assembles these records into a local linear system. Once a worker is finished, it sends a ready message back to the master. The master sends tasks to workers as long as there are data records to assemble. With this method an automatic load balancing is achieved as more loaded processors (e.g. by tasks from other users) need longer to assemble records but automatically get less data. However, the additional costs for this load balancing are small messages that are communicated between the master and workers.

For both implementations the innermost loop operates in a blocking structure, i.e. a block (a given number of data records) is loaded from file and assembled into the local linear system, then the next block is loaded and assembled, etc. By using different block sizes it is possible to trade between disk reading, memory usage and communication (for the master-worker implementation only).

Once all data records are assembled the local linear systems have to be collected somehow to form the final linear system. Depending on the algorithm used to solve the linear system (sequential or parallel) we reduce and sum the linear system on one processor or we use a block-cyclic redistribution. The results presented in this section include the time to reduce and sum the linear system on one processor.

Fig. 8. Assembly on Beowulf Linux cluster

Based on real-world Medicare health data we created a synthetic test set (about 11 million records) containing three continuous and four categorical attributes. The times for assembling this data set on two different parallel architectures can be seen in Figures 7 and 8. The first machine used was the twelve processor Sun Enterprise described in the previous section, while the second machine was the 196 processor Beowulf Linux cluster *Bunyip* at the Australian National University [1]. This cluster is built with 98 dual 550 MHz Pentium III nodes, each equipped with 384 Megabytes of RAM (totally about 36 Gigabytes), 13 Gigabytes of disk space (totally 1.3 Terabytes) and 3×100 MBit/s fast Ethernet cards. Logically 96 nodes are connected in four groups of 24 nodes arranged as a tetrahedron with a group of nodes at each vertex. Two nodes are designated as servers. For our tests we only used two groups (i.e. up to 48 nodes), whereas all data files have been distributed onto local disks.

On both architectures an almost linear speedup can be observed. As there is one worker less than the number of processes, the master-worker implementation is generally slower than the SPMD implementation on the same number of processes. The master-worker test-run with two processes on the Beowulf is even slower than the sequential version, which is due to communication. We also did timing tests with smaller and larger block lengths. For blocks of length 1,000 the speedup for the master-worker implementation is quite smaller than linear (specially on the Beowulf) due to an increased communication volume.

Because the data files are replicated onto local disks on every node of the Beowulf, there is no communication involved with file access. The assembled linear system in the presented example is with $m < 100$ very small, but even when communicating a larger system the time for the reduction step is small compared to the time for the assembly process.

9 Solving the Linear System

The systems currently generated by HISURF (Section 4) and ADDFIT (Section 5) are dense and symmetric, positive definite in the former case, and semi-definite in the latter case. However, in future refinements of these models, the definiteness property may be lost, for example because of the addition of extra constraints or in the case of additive models, extending it to a second-order model.

The size of the assembled linear system depends on the total number of categories for categorical variables and on the number of grid points for continuous variables. Solving this system can be done on either a sequential or parallel solver depending on the size of the system and the availability of a parallel machine.

We thus require a solver that will be accurate for any symmetric dense system, and also have good parallel and sequential performance. The former requirement argues for a direct solver with good stability properties; the latter argues for one that exploits symmetry to require only $\frac{m^3}{3} + O(m^2)$ floating point operations, and that has been shown to have an efficient parallelisation. A direct solver for general symmetric (indefinite) systems based on the diagonal pivoting method [2,14] meets these requirements.

In the diagonal pivoting method, the decomposition $A = LDL^T$ is performed, where L is an $m \times m$ lower triangular matrix with a unit diagonal, and D is a block diagonal matrix with either 1×1 or 2×2 sub-blocks [14]. The factorisation of A proceeds column by column; in the elimination of column j , three cases arise:

1. Eliminate using a 1×1 pivot from $A_{j,j}$. This corresponds to the definite case, and will be used when $A_{j,j}$ is sufficiently large (compared with $\max(A_{j+1:m,j})$).
2. Eliminate using a 1×1 pivot from $A_{i,i}$, where $i > j$. This corresponds to the semi-definite case; a symmetric interchange with row/columns i and j must be performed.
3. Eliminate using a 2×2 pivot using columns i' and i ($i', i \geq j, i' \neq i$). This case produces a 2×2 sub-block at column j of D . This corresponds to the indefinite case; a symmetric interchange with rows/columns i', i and $j, j+1$ must be performed. However, columns j and $j+1$ are eliminated in this case.

The tests used to decide between these cases, and the searches used to select column i (and i'), yield several algorithms based on the method, the most well-known being the various variants of the Bunch-Kaufman algorithm (see [14] and the references cited within).

It has been recently shown for the Bunch-Kaufman algorithm that there is no guarantee that the growth of L is bounded [2]. Variants such as the *bounded Bunch-Kaufman* and *fast Bunch-Parlett* algorithms have been devised which overcome this problem [2]. The extra accuracy of these methods results from more extensive searching for stable pivot columns i (and i') for cases 2 and 3, with a correspondingly more frequent use of these cases.

For linear systems that are close to definite, such as are likely to be generated by our models, the diagonal pivoting methods permit most columns to be eliminated by case 1, requiring no symmetric interchanges. For parallel implementation, this is a highly useful property, as even for large matrices, the communication startup and volume overheads of symmetric interchange, when the rows and columns come from different processors, is considerable [26].

Instead of suppressing interchanges, which even if done judiciously may result in the loss of some accuracy [26], high parallel performance can also be achieved with a *block-search* algorithm that searches for suitable pivot columns i and i' from the current storage block [21]. If this search was successful, the symmetric interchanges would require no communication, resulting in no parallel overhead. Such a strategy could be based on the Duff-Reid algorithm used for sparse matrices [2,21], which also has strong guarantees of accuracy.

However, if the search was not successful, an equally stable means of eliminating column j must then be used. We chose the *bounded Bunch-Kaufman* algorithm over the *fast Parlett-Reid* algorithm, as the latter requires sorting of the columns by the size of the diagonal, which would give it higher parallel overheads.

Preliminary results on a 9 node Fujitsu AP3000 distributed memory multi-computer (having 170 MHz UltraSPARC processors) are given in Table 2. The matrices are random from the unit circle, with a bias $D \geq 0.0$ added to the diagonal elements. $D = 0$ corresponds to a strongly indefinite matrix, at $D = 10$ about 2/3 of the columns are eliminated by case 1, whereas $D = 100$ is almost positive definite (about 99% of the columns are eliminated by case 1). An algorithmic blocking factor of 64, equal to the block-cyclic distribution storage block size, was used.

In Table 2, the quantity f represents the fraction of columns that were eliminated requiring interchanges from outside the current storage block. The low values of f indicate that the block search was reasonably effective. The speedups increasing with D indicate the overhead (redundant computation and some off-block symmetric interchanges) of cases 2 and 3. Compared with the fastest known serial algorithm on the UltraSPARC, a highly optimised implementation of the less accurate Bunch-Kaufman algorithm [26], the *block search* is between 5%–10% slower. This is primarily due to overheads of extra

Table 2

Performance in GFLOPs (and parallel scaled speedup) of a solve of an $m \times m$ linear system using the *block-search* algorithm on a 3×3 AP3000

D	f	Matrix dimension m				
		1,500	3,000	4,500	6,000	7,500
0	≈ 0.3	0.35 (3.7)	0.71 (5.1)	0.94 (6.2)	1.10 (6.9)	1.21 (7.5)
10	0.02	0.50 (4.2)	0.94 (5.8)	1.12 (5.9)	1.29 (7.1)	1.34 (7.6)
100	0.00	0.56 (4.3)	1.03 (6.0)	1.24 (6.8)	1.39 (7.3)	1.45 (7.5)

column maximum finding required by the block search, and the fact that it also requires a right-looking level-2 factorisation (slower than the left-looking level-2 factorisations normally used in diagonal pivoting methods).

10 Conclusions

We have presented a three-fold approach to predictive modelling of very large data sets, which is both scalable with the data size and the number of processors. All three methods are based on approximate smoothing splines using finite element techniques. While all methods are equally scalable, they differ in their approximation power and range of dimensions for which they are applicable. A first method, called TPSFEM, is based on piecewise multilinear elements and is most suited to low-dimensional problems. For the intermediate range of dimensions we suggest HISURF which uses a sparse-grid type approach based on interpolatory wavelets. Finally, for very high-dimensions we have developed ADDFIT implementing additive models.

Timing results on different parallel architectures show the parallel scalability of the first step – the assembly of a linear system – of our methods. Work on a optimised parallel solver for the linear systems resulting from our methods is an ongoing project.

In future work we plan to add support for data types other than continuous and categorical variables. We hope to include in a first instance support for sets, time series and graphs. We plan also to explore how the reduction techniques could be used to address data mining problems other than prediction.

Acknowledgements

This research is partially supported by the Australian Advanced Computational Systems CRC. Peter Christen is funded by grants from the *Swiss National Science Foundation* (SNF) and the *Novartis Stiftung, vormalis Ciba-Geigy Jubiläums-Stiftung*, Switzerland.

References

- [1] D. Aberdeen, J. Baxter and R. Edwards, 98 c/MFlop, Ultra-Large-Scale Neural Network Training on a PIII Cluster, Gordon Bell award price/performance entry, and student paper award entry, submitted to the High-Performance Networking and Computing Conference, Dallas, November 2000.
- [2] C. Ashcraft, R.G. Grimes and J.G. Lewis, Accurate Symmetric Indefinite Linear Equation Solvers, *SIMAX*, 20(2):513–561, 1998.
- [3] S. Bakin, M. Hegland and M. Osborne, Can MARS be Improved with B-Splines?, *Computational Techniques and Applications: CTAC-97*, World Scientific Publishing Co., pages 75-82, 1998.
- [4] R.K. Beatson, G. Goodsell and M.J.D. Powell, On Multigrid Techniques for Thin Plate Spline Interpolation in Two Dimensions, in the *Mathematics of Numerical Analyses, Lectures in Appl. Math.*, pages 77-97, 1996.
- [5] G. Bell and J.N. Gray, The revolution yet to happen, *Beyond Calculation* (P.J. Denning and R.M. Metcalfe, eds.), Springer Verlag, 1997.
- [6] M.J.A. Berry and G. Linoff, *Data Mining, Techniques for Marketing, Sales and Customer Support*. John Wiley & Sons, 1997.
- [7] P. Chapman, R. Kerber, J. Clinton, T. Khabaza, T. Reinartz and R. Wirth, The CRISP-DM Process Model, Discussion paper, March 1999. www.crisp.org
- [8] P. Christen, M. Hegland, S. Roberts and I. Altas, A Scalable Parallel FEM Surface Fitting Algorithm for Data Mining, submitted to the *International Journal of Computer Research (IJCR)*, Special Issue on Industrial Applications of Parallel Computing, November 1999.
- [9] I. Daubechies, Orthonormal bases of compactly supported wavelets, *Comm. Pure and Appl. Math.*, pages 909-996, 1998.
- [10] J. Duchon, Splines Minimizing Rotation-Invariant Semi-norms in Sobolev Spaces, in *Lecture Notes in Math.*, vol. 571, pages 85-100, 1977.
- [11] D. Düllmann, Petabyte databases. *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD-99)*, ACM Press, July 1999.

- [12] B. Efron and C. Stein, The jackknife estimate of variance, *The Annals of Statistics*, vol. 9, num. 3, pages 586-596, 1981.
- [13] J.H. Friedman, Multivariate adaptive regression splines, *The Annals of Statistics*, vol. 19, pages 1-141, 1991.
- [14] G. Golub and C. Van Loan, *Matrix Computations*, John Hopkins University Press, Baltimore, second edition, 1989.
- [15] T.J. Hastie and R.J. Tibshirani, *Generalized additive models*, Chapman and Hall, *Monographs on statistics and applied probability* 43, 1990.
- [16] S. Hawkins, G. Williams, R. Baxter, P. Christen, M. Fett, M. Hegland, F. Huang, O.M. Nielsen, T. Semenova and A. Smith, *Data Mining of Administrative Claims Data of Pathology Services*, Submitted to the Thirty-Fourth Hawaii International Conference on System Sciences (HICSS-34), January 2001.
- [17] M. Hegland, S. Roberts and I. Altas, Finite Element Thin Plate Splines for Data Mining Applications, in *Mathematical Methods for Curves and Surfaces II*, M. Daehlen, T. Lyche and L.L. Schumaker, Eds., pages 245-253, Vanderbilt University Press, 1998.
- [18] M. Hegland and V. Pestov, *Additive Models in High Dimensions*, School of Mathematical and Computing Sciences, Victoria University of Wellington, 1999.
- [19] M. Hegland, O.M. Nielsen and Z. Shen, *High Dimensional Smoothing based on Multi-level Analysis*. In preparation.
- [20] M. Hegland, *Mining the MACHO Dataset*, invited presentation, to be presented at the Conference on Computational Physics 2000 (CCP-2000), Gold Coast, December 2000.
- [21] J.G. Lewis. Private communications, 1999.
- [22] W. Gropp, E. Lusk and A. Skjellum, *Using MPI – Portable Parallel Programming with the Message-Passing Interface*. The MIT Press, Cambridge, Massachusetts, 1994.
- [23] O.M. Nielsen, *Wavelets in Scientific Computing*, PhD thesis, Technical University of Denmark, 1998.
Available at www.bigfoot.com/~Ole.Nielsen/thesis.html
- [24] R. Sibson and G. Stone, Computation of Thin Plate Splines, *SIAM J. Sci. Statist. Comput.*, 12:1304-1313, 1991.
- [25] G. Strang and T. Nguyen, *Wavelets and Filter Banks*, Wellesley-Cambridge Press, 1996.
- [26] P.E. Strazdins, Accelerated Methods for Performing the LDLT Decomposition, *Proceedings of CTAC-99: The 9th Biennial Computational Techniques and Applications Conference and Workshops*, Canberra, September 1999.
- [27] G. Wahba, *Spline models for observational data*, CBMS-NSF Regional Conference Series in Applied Mathematics, vol. 59, SIAM, 1990.