Parallel Methods for Scaling Data Mining Algorithms to Large Data Sets

ROBERT GROSSMAN
robert.grossman@acm.org
Magnify, Inc. &
Laboratory for Advanced Computing/National Center for Data Mining,
University of Illinois at Chicago, USA

YIKE GUO
yg@doc.ic.ac.uk
Department of Computing
Imperial College, University of London, UK

This is a draft of the following publication: R. L. Grossman and Yike Guo, Parallel Methods for Scaling Data Mining Algorithms to Large Data Sets, Hanndbook on Data Mining and Knowledge Discovery, Jan M Zytkow, editor, Oxford University Press, 2002, pages 433 - 442.

In this chapter, we describe some approaches and specific techniques for scaling data mining algorithms to large data sets through parallel processing. We then analyse in more detail three core algorithms that can be scaled to large data sets: building decision trees, discovering association rules, and creating clusters.

C5.10.1 Introduction

A fundamental challenge is to extend data mining to large data sets. In this chapter, we introduce some of the basic approaches and techniques that have proved successful and describe in some detail work on scaling three fundamental data mining algorithms: trees, clustering algorithms and association rules.

Section C5.10.2 introduces computational models for working with large data sets. By large we mean data that does not fit into the memory of a single processor. Parallel RAM computational models describe algorithms which are distributed between several processors. Hierarchical memory computational models describe algorithms that require working with data both in memory and on disk. Parallel disk computational models describe algorithms in which data is distributed over several processors and disks.

Section C5.10.3 surveys some of the basic approaches to scaling data mining algorithms. The most basic approach is to manipulate the data until it fits into memory. Another fundamental technique is to use specialized data structures to work with data which is disk resident. We also describe techniques for distributing algorithms between several processors, precomputing various quantities, and intelligently reducing the amount of data.

Sections C5.10.4, C5.10.5 and C5.10.6 describe work on scaling tree-based algorithms, association rules, and clustering algorithms to large data sets.

C5.10.2 Computational and Programming Models

In this section, we briefly describe some of the different computational and programming models that are used in high performance and parallel computing. We begin by discussing the cost of computation. We describe 4 models: a RAM model, a parallel RAM model, a disk model and a parallel disk model. Next, we describe two basic distinctions between the various programming models used in high performance computing. The first distinction is whether the data itself is used to determine the parallelism (data parallelism) or whether the parallelism is determined explicitly by the programmer (task parallelism). The second distinction is how different processors communicate: this can be done with shared memory, with message passing, or with remote memory operations.

Computational Models. The standard model for measuring the complexity of an algorithm is the random access machine model (RAM) (Aho 1974). A RAM model has a single processor with unlimited memory, which can store and access data with unit cost. With the RAM model, sorting N records has cost O(N log N).

Parallel computers exploit multiple processors. Shared memory parallel computers allow more than one processor to share the same memory space. With the P-RAM model, different processors may simultaneously read the same memory location, but may not simultaneously write to the same memory. In distributed memory parallel computers, each processor has its own memory and processors communicate by explicitly sending messages to each other over an interconnection network. See (Kumar 1994) for more details about shared memory and distributed memory parallel computers.

In practice, accessing data from disk can affect the running time of an algorithm by one or two orders of magnitude so that an order $O(N^3)$ algorithm effectively becomes an order $O(N^5)$ algorithm. To model this, the most basic i/o model assumes that data is either in memory or on disk, and that data in memory can be accessed uniformly with unit cost, while data on disk can be accessed uniformly, but at a higher cost. On a parallel computer, there will usually be several disks which can read and write blocks in parallel. With the Parallel Disk Model (Vitter 1994), B data records (a block of data) can be read from disk into memory at unit cost and that D blocks of data can be read or written at once. A typical algorithm will read M=DB records into memory, compute with them, and write out necessary information to disk. An external memory algorithm is designed to work with N > M records so that several memory loads of M records must be used to examine all of the data. With the parallel disk model, sorting has cost [Vitter 1994]

$$O((N/DB) \log(N/B)/\log(M/B)).$$

N	number of input records
P	number of processors
M	number of records that fit into the aggregate internal memories of the P processors
B	number of records that can be transferred in a single block
D	number of disks, and more generally the number of disk blocks that can be transferred with one parallel read or parallel write operation

Communication Primitives. Certain communication patterns in parallel algorithms are very common and typically special hardware and software is provided to support them. We describe three of these here. *Scatter* takes a value at one processor and sends it to all the other processors. *Gather* takes values at all the processors and brings them to a common processor. *Reduction* takes values at all of the processors, computes the sum, and places the sum in each of the processors. Reduction can also be used for computing the max, min, and similar operations.

Data Parallelism. With data parallelism, data is divided into different partitions, the same program is run on each partition, and the results combined. Finding the maximum value in a list of N elements has an easy data parallel solution. If the list is divided into P sublists and one is stored in the local memory of each processor, then each processor can determine its local maximum and send the maximum to a central processor or place it in common shared memory. The global maximum is then the largest of the P values. As another example, a data parallel approach to growing a tree, splits the data into P partitions, grows a single tree on each partition, and then produces an ensemble of P trees

(Grossman, Bodek et. al. 1996).

Task Parallelism. Task parallelism is specified explicitly by the programmer. For example, a task parallel approach to growing a tree uses P processors to speed up the computation of locating the best split for a single node in the tree. In the simplest task parallel approach, the data is distributed evenly between the P processors and, for each attribute, each processor computes the class distribution information for that attribute using its local data. See Section C5.10.3 for an example of class distribution information. Reduction is used to exchange local class distribution information with each of the other P-1 processors to compute global class distribution information. A single split value is computed and scattered to each of the other P-1 processors. Using this split value, the data is distributed between the two nodes produced by the split and the process repeats. Notice, that unlike the ensemble based approach described above, the different processors need to communicate class distribution information before a split can be determined. On the other hand, a single tree is computed, where as the ensemble based approach yields a collection of trees.

Shared Memory. The simplest way for different processors to communicate is for each to share some global memory. Locking is used to control conflicts when different processors write to the same memory location. As the number of processors grows, it becomes more difficult to design machines in which all the global memory can be accessed uniformly. Some architectures allow each processor access to global memory, but different processors may require different amounts of time to read and write the common shared memory. A variant is for each processor to have some local memory and some global memory.

Message Passing. With message passing, each processor has its own memory, and different processors communicate by explicitly sending and receiving messages between them with a send and receive command. Messages are simply buffers of data of specified length.

Remote Memory Operations. With remove memory operations, a processor can explicitly access memory of other processors, but different operations are used for accessing local and remote memory. For example, local memory access is implicit, while remote memory access requires explicit get or put commands. Unlike message passing, in which the remote processor must explicitly receive the message, with remote memory operations, all the work is done by the local processor.

C5.10.3 Five Basic Approaches for Scaling Data Intensive Computing

Approach 1. *Manipulate the data so that it fits into memory*. We begin with the most common approach. There are four basic variants. The first is to *sample* the data until the number of records N is smaller than the memory of a single processor. The second is to *select features* until the amount of data is smaller than the memory a single processor. The third is to *partition* the data so that although it doesn't fit into the memory of a single processor, it does fit into the aggregate memory M of the processors. The fourth technique is to *summarize* the data in some fashion so that the summarized or partly summarized data can fit into memory. These four techniques can be used in any combination with each other. Broadly speaking, these techniques arose from the statistical community.

Approach 2. *Reduce the time to access out of memory data*. Special care is required when accessing data from disk. No more time is required to access all B records in a block on disk than is required to access any single one of them. Three basic techniques are common. The first uses *specialized data structures* to access data on disk. The most familiar is the B+-tree which uses a tree structure to determine which block contains a desired record and which has efficient operations for adding new blocks and merging existing blocks (Ramakrishnan 1997). The second technique is to lay out the data on disk to benefit from *block reads*. For example, some algorithms proceed faster if data is organized by record and others if data is organized by attribute. The third technique organizes data on disk to benefit from *parallel block reads*. A basic example is provided by matrix transpose. Instead of organizing the disk by row or column, a slightly more complicated organization can cut the number of reads by a factor of 2 (Shriver 1996). Broadly speaking, these techniques arose from the database community.

Approach 3. *Use several processors*. One of the easiest ways to speed up algorithms on large data sets is to use more than one processor. The success depends upon how easy it is to break up the problem into sub-problems which can be assigned to the different processors. As described above, there are two basic techniques. The first technique is data parallelism. With this technique, essentially the same program is applied to different partitions of the data. The second technique is task parallelism. With this technique, the program itself is broken into sub-tasks, which are distributed among the available processors. We will examine several examples of both techniques later in this chapter. Broadly speaking these techniques come from the high performance computing community.

Approach 4. *Precompute*. The most expensive part of building tree based classifiers on continuous attributes is sorting. For example, the table below contains the class distribution information for the continuous attribute 8. Computing the best split value for the tree requires sorting the data by the attributes values as indicated. Precomputing these sorts reduces the cost of the algorithm (Mehta 1996). For efficiency, specialized data structures are usually employed (Shafer, Agrawal and Mehta 1996).

Intermediate computations can sometimes be shared across algorithms. For example cross-tab tables, such as the one below for ordinal attribute 5, are of intrinsic interest and are also used by different algorithms including trees. Precomputing such tables can often save significant time.

A closely related approach is to provide very efficient implementations for certain basic operations, such as computing statistics on columns, which can be shared across algorithms. Sometimes these are known as data mining primitives. Broadly speaking, these techniques were developed by the data mining system implementation community.

Class Distribution Information for Attribute 8		
Attribute Value	Fraud	
	Fraud	No Fraud
0	2	284
.04	3	296
.15	1	672
.18	1	485
.26	1	794
etc.	etc.	etc.

Class Distribution Information for Attribute 5		
Attribute Value	Fraud	
	Fraud	No Fraud
0 (codes 0-1)	129	48484
1 (codes 2-5)	494	58492
2 (codes 5-9)	696	54040
3 (codes >9)	789	40949

Approach 5. *Reduce the amount of data.* This approach is very similar to Approach 1, except that there is no expectation the data will be able to fit into memory. Three of the techniques mentioned in Approach 1 apply here without change: sampling, selecting features, and summarizing data. We also mention three more specialized techniques, which can also sometimes be used in Approach 1. Discrete data points can be *smoothed* and replaced by a continuous approximation specified by one or more parameters. For example, a set of points can be replaced by its center, a measure of dispersion, the number of points, the sum of the errors, and the sum of the squared errors. Data can also be *compressed* and computations done directly on the compressed data. Finally, data can be *transformed* with more complicated transformations which reduce the size of the data and variants of

algorithms can be applied directly to the transformed data. For example, data can be reduced with a principal components analysis.

C.5.10.4 Parallel Tree Induction

In this section we describe some of the ways that have been used to scale tree algorithms. For simplicity we describe these approaches in the context of the C4.5 system (Quinlan 1993). C4.5 attempts to find the simplest classification tree that describes the structure of the data by applying search heuristics based on information theory. At any given node in the tree, the algorithm chooses the most suitable attribute to further expand it based on the concept of information gain, a measure of the ability of an attribute to minimise the information needed to classify the cases in the resulting sub-trees. The algorithm constructs a tree recursively using a depth first divide-and-conquer approach. Other tree induction algorithms share a similar computation structure. See Section C5.1.3.

There are three main approaches for building trees in parallel.

Move Class Distribution Information. This approach is based on dividing the initial data set evenly among the P processors. The processors leave the data in place but move the class distribution information (C5.10.3) using reduction in order to computer the splitting values, as described in the section above on task parallelism. In more detail, consider the expansion of a single node into its children using splitting values. Each processor computes the class distribution information for each attribute using its local data. Each processor then uses reduction with the other P-1 processors to compute the global class distribution information. The processors then simultaneously compute the splitting criteria and scatter the value of the attribute with the best split value. Using these splitting values, the data is then assigned to the children and the process continues.

The main advantage of this approach is that no data needs to be moved. On the other hand, moving the class distribution information can have a high communication cost and can result in a load imbalance. In particular, the deeper the tree, the less the data, and the greater the overhead of the communication.

Move Data. The advantage of this approach is that, when possible, different processors can work on different nodes at the same time. The basic idea is simple. Assume a group of processors are assigned to a node. The processors work together to compute the split value as described with the Move Class Distribution Information approach above. Assume that the number of children computed by the split is less than the number of processors available. Split the processors between the children and then distribute the data to the processors that are assigned to it. This partitions the underlying data between several processors so that they can work simultaneously. The processors are next used to compute the split value. Processors assigned to different nodes can proceed independently. The case in which the number of children is greater than the number of processors is handled similarly. Kumar (1998) gives a performance formula for this approach.

This method is also referred to as search parallelisation (Provost and Kolluri 1999) since the search space is divided among the processors so that different processors search different portions of the space in parallel.

A disadvantage of this approach is that moving data can result in a high communication overhead. Another disadvantage is that the work load can become unbalanced. On the other hand, an advantage of this approach is that once a single processor is assigned to a node it can compute the subtree without any communication overhead. Zaki (1998) applies this approach to parallel tree induction by taking advantage of shared memory multiprocessor architecture.

Ensemble-based Methods. With this approach, the data is divided into partitions, perhaps overlapping, and one or more processors are used to build a separate tree for each partition (Grossman et. al. 1996, Grossman and Poor 1996). This produces an ensemble of trees, which can be combined using a variety of methods. An ensemble is a collection of statistical models, together with a rule for a combining the models into a single model. For example, the models may be combined with a voting function or a function which averages the various values produced by the separate models. See (Dietterich 1997) for additional information about ensembles in data mining.

Two or more of these approaches may be combined to produce hybrid algorithms. For example, Kumar et al. (1998) describe an algorithm which starts by exploiting the approach of moving class distribution information. When the inter-processor communication and synchronisation requirements increase pass a certain threshold, the implementation switches to exploiting a mixture of approaches, involving moving both data and class distribution information. This method is also referred to as parallel matching (Provost and Kolluri 1999). This approach has been adopted in the work of Provost and Aronis (1996) and in the parallelisation of the SPRINT algorithm (Shafer, Agrawal and Mehta 1996) as well as in the recently proposed ScalParC parallel tree induction system (Joshi 1998). Other examples are given by Pearson (1994) who uses a vertical partitioning strategy, and Han (1999) who uses a horizontal partitioning strategy.

Ensemble based methods have also been combined with approaches that move both data and class distribution information (Grossman et. al. 1996).

It should be noted that after generating a classification tree by an algorithm such as C4.5, several post-processing steps might still be required. These are applied in order to simplify the tree and to translate it into a set of production rules (Quinlan 1993). Kufrin (1997) has noted that these post-processing steps may require more computation time than the actual tree generation phases and has describes how such steps can be parallelised.

C.5.10.5 Parallel Association Rule Discovery

Algorithms for uncovering associations were introduced in Agrawal et. al. (1993). The well known Apriori algorithm (Agrawal 1993) generates association rules by computing frequent item sets (C5.2.3). Frequent item sets of length 1 are simply singleton sets. Given a frequent item set of length n, there are efficient algorithms for computing frequent item sets of length n+1 (Agrawal 1994). The Apriori algorithm uses a hash tree to maintain items of length n while it computes frequent item sets of length n+1. The hash tree is required to remain in main memory, although the transaction data set is not. Association rules can be read easily off from frequent item sets.

There are two steps to construct frequent item sets of length n+1. In the first step, a set of candidate frequent item sets is created. In the second step, the entire database is scanned to count the number of transactions that the candidate sets contain. Concurrency can be used in both steps – parallel processing can be used to speed the creation of candidate frequent item sets and to speed up the counting of transactions.

Data parallel approaches which distribute the transaction data among several processors and count the transactions in parallel have been proposed by Park et. al. (Park 1995) and Agrawal et. al. (Agrawal 1996). The Count Distribution (CD) algorithm of Agrawal et. al. is an adaptation of the Apriori algorithm. At each iteration, the algorithm generates the candidate sets at each local processor by applying the same generation function as that used in the Apriori algorithm. Each processor then computes the local support counts of all the candidate sets and uses a reduction for computing the global frequent item sets for that iteration. In this way, the CD algorithm scales linearly with the number of transactions. On the other hand, since the CD algorithm, like Apriori, requires the hash tree to fit into the memory of a single processor, it does not scale as the number of candidates in the frequent item sets increases.

To scale as the number of candidates in the frequent item sets increases, the frequent item sets themselves can be distributed among the processors. In this case, a simple hash tree fitting into the memory of a single processor can no longer be used. Simple implementations of this idea require moving all the data to each of the processors in order to compute the counts. Sometimes this is called the Data Distribution (DD) method. Simple DD algorithms do not perform well due to communications overhead, but more complex implementations have been developed with better performance.

CD style algorithms scale to large transaction data sets since the transactions are partitioned. DD style algorithms scale to problems with large candidate sets since the candidates are partitioned. Some algorithms combine these two approaches to achieve scalability along both dimensions (Han 1997).

Chung (1996) observed that a global frequent item set must be a local frequent item for some processor. With this property, much smaller candidate sets can be generated in parallel at each processor. Moreover, local pruning can be applied by removing those sets that are not locally large. The communication required for exchanging support counts is therefore reduced from the $O(P^2)$ of directly parallelising Apriori, to $O(P)$, where P is the number of distributed processors or computers. This type of algorithm can be extended easily to parallelize any pattern discovery algorithms which employs a level by level monotonic search component like that of Apriori.

Speeding up association rules through sampling is discussed in Lee et. al. (1998).

In their recent paper (Pei 2000), Pei and Han et.al. proposed the CLOSET algorithm for computing association rules. Instead of generating frequent item sets, the algorithm computes a much smaller set of candidates. Their algorithm also employs a compact representation of association rules. This algorithm is based on a memorisation mechanism which avoids redundant computations. A partition-based approach can be used to scale this algorithm to large data sets. At this time, parallel versions of this algorithm have not been studied in detail.

C.5.10.6 Parallel Clustering Algorithms

Clustering algorithms can be broadly divided into three types: distance-based clustering, hierarchical clustering and density based clustering (C5.5). In general, clustering algorithms employ a two-stage search: an outer loop over possible cluster members and an inner loop to fit the best possible clustering for a given number of clusters.

With *distance-based clustering*, *n* clusters are constructed by computing a locally optimal solution to minimise the sum of the distances within the data clusters. This is either done by starting from scratch and constructing a new solution or by using a valid cluster solution as a starting point for improvements. A common distance-based algorithm is the K-Means algorithm, which minimises the sum of the distance between each data point and its nearest cluster centre (Selim and Ismail 1984). Parallelism in the distance-based clustering methods can be exploited in both the outer level, by trying different cluster numbers concurrently, and in the inner level by computing the distance metrics in parallel.

Hierarchical clustering groups data with a given similarity measurement into a sequence of nested partitions. Two different approaches can be employed. One is to start with each data point as a single cluster and then in each step, merge pairs of points together. This is known as the *agglomerative* approach. The alternative is to start with all data points in one cluster and then in each divide one cluster in two clusters in each step. This is the *divisive* approach. For both methods, $O(N^2)$ algorithms are known. Recent attempts have been made to develop parallel algorithms for hierarchical clustering using several distance metrics in parallel (Olson 1995).

With the *density-based clustering* approach, clustering is done by postulating a hidden density model indicating the cluster membership. The data is assumed to be generated from a mixture model with hidden cluster identifiers. The clustering problem is then one of finding parameters for each individual cluster which maximise the likelihood of the data set given the mixture model. A typical density-based clustering method is the EM algorithm which employs an iterative search procedure to find the best parameters of a mixture model to fit the data. The iteration procedure comprises the following steps:

1. Initialise the model parameters, thereby producing a current model
2. Decide memberships of the data items to clusters, assuming that the current model is correct
3. Re-estimate the parameters of the current model assuming that the data memberships obtained in 2 are correct, producing new model
4. If current model and new model are sufficiently close to each other then terminate, else goto 2.

This procedure has the same structure as the K-Means method where the only model parameter is the distance between assumed cluster centres and data points. The search hierarchy in EM algorithms includes the outermost-level search on cluster numbers, the middle-level search for functional forms and the inner-level search for parameter values. The rich inherent parallelism of the algorithm may be exploited by combining the decomposition of loops (task parallelism) and partition of data (data parallelism). Subramonian (1998) presents a parallelisation of the EM algorithm. The model employs three different methods for parallelising each of the three levels of search loops:

- vectorise the computation of parameters (inner level search);
- exploit data parallelism in computing the cluster model given the cluster number (middle level search);
- concurrently search cluster numbers using parallel machine clusters.

This method provides a general framework for parallelising iterative clustering procedures.

C5.10.7 Discussion and Summary

Broadly speaking, techniques for scaling data mining algorithms can be divided into five basic categories : 1) manipulating the data so that it fits into memory, 2) using specialized data structures to manage out of memory data, 3) distributing the computation so that it exploits several processors, 4) precomputing intermediate quantities of interest, and 5) reducing the amount of data mined.

During the past several years, there have been successes scaling tree based predictors and association rules to large data sets which do not fit into memory, but rather fill the memories of several processors, spill onto disks, or both. More recently, techniques have been introduced which scale clustering algorithms. These successes have typically involved combining two or more of the approaches described above.

References

Agrawal, R., Faloutsos. C, and Swami, A (1993). " Efficient Similarity search in Sequence Databases." In *Proceedings of 4th International Conference on Foundations of Data Organization and Algorithms.*

Agrawal, R. Imielinski, T., and Swami, A. (1993). "Mining Association Rules Between Sets of Items in Large Databases." In *Proceedings of the ACM SIGMOD International Conference on Management of Data,* page 207. ACM.

Agrawal, R. and Srikant, R. (1994). "Fast Algorithms for Mining Association Rules in Large Databases." In *Proceedings of the Twentieth International Conference on Very Large Databases*, pages 487-499.

Agrawal, R. and Shafer, J. (1996) "Parallel Mining of Association Rules" In *IEEE Transactions on Knowledge and Data Engineering, Vol. 8, No.6*

Aho, A., Hopcroft, J. and Ullman, J, (1974). *The Design and Analysis of Computer Algorithms.* Addison-Wesley, Reading, Massachusetts.

Allmuallim, H., Akiba, Y., and Kaneda, S. (1995) "On Handing Tree-Structure Attributes in Decision Tree Learning." In *Proceedings of the Twelfth International Conference on Machine Learning.*

Bradley, P., Fayyad, U. and Reina, C. (1998). "Scaling Clustering Algorithms to Large Databases." In *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining.*

Brieman, L., Friedman, J. H., Olshen, R. A., and Stone, C. J. (1984). *Classification and Regression Trees*. Belmont: Wadsworth.

Brieman, L. (1996). “Bagging Predictors.” *Machine Learning*, 24-2: pages123-140.
Catlett, J. (1991). *Megainduction: Machine learning on very large databases.* PhD Thesis, University of Sydney.

Chan, P. and Stolfo (1997). "On the Accuracy of Meta-Learning for Scalable Data Mining" In *Journal of Intelligent Information Systems, 8.*

Chattratichat, J., Darlington, J., Ghanem, M., Guo, Y., Huning, H., Kohler, M., Sutwaraphun, J., To, H. and Yang, D. (1997). "Large Scale Data Mining: Challenges and Responses." In *Proceedings of the Third International Conference on Knowledge Discovery and Data Mining.*

Chattratichat, J., Darlington, J., Guo, Y., Hedvall, S., Kohler, M. and Syed, J. (1998). "An Architecture for Distributed Enterprise Data Mining." In *Proceedings of the Seventh International Conference on High-Performance Networking and Computing (HPCN Europe).*

Cheung, D.W., Han, J. Ng, V.T. and Wong, C.Y. (1996) "Maintenance of Discovered Association Rules in Large Databases : An incremental Updating Techniques." In *Proceedings of International Conference on Data Engineering.*

Craven, M.W. (1996) "Extracting Comprehensible Models from Trained Neural Networks" Ph.D Thesis, University of Wisconson, *Technical Report No.1326*

Dietterich, T. G. (1997). “Machine Learning Research.” AI Magazine, 18: 97-136.

Domingos, P. (1997). "Knowledge Acquisition from Examples via Multiple Models." In *Proceedings of the Fourteenth International Conference on Machine Learning (ICML-97),* 98-106.

Freitas, A. and Lavington, S. (1998). "Mining Very Large Databases with Parallel Processing." Kluwer Academic Publishers.

Bennet, K., Fayyad, U and Geiger, D. (1999). "Density-Based Indexing for Approximate Nearest-Neighbor Queries." In *Proceedings of the Fifth International Conference on Knowledge Discovery and Data Mining.*

Freund, Y. and Schapire, R. E. (1995). “A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting. *Proceedings of the Second European Conference on Computational Learning Theory*, pages 23-37. Berlin: Springer-Verlag.

Graefe, G., Fayyad, U. and Chaudhuri, S. (1998). "On the Efficient Gathering of Sufficient Statistics for Classification from Large SQL Databases." In *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining.*

Grossman, R. L., Bodek, H., Northcutt, D., and Poor, H. V. (1996). “Data Mining and Tree-based Optimization.” In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, E. Simoudis, J. Han and U. Fayyad, editors, pages 323-326. Menlo Park, California: AAAI Press.

Grossman, R. L. and Poor, H. V. (1996). “Optimization Driven Data Mining and Credit Scoring.” In *Proceedings of the IEEE/IAFE 1996 Conference on Computational Intelligence for Financial Engineering (CIFEr).* Piscataway: IEEE, pages 104-110.

Guo, Y. and Sutiwaraphun, J. (1998). "Knowledge Probing in Distributed Data Mining." In *Working*

Notes of the KDD-97 Workshop on Distributed Data Mining, pp. 61-69.

Han, E., Karypis, G., and Kuma, V. (1997). “Scalable parallel data mining for association rules.” In Proceedings of the 1997 ACM-SIGMOD International Conference on the Management of Data.

Han, J., Fu, Y., Wang, W., Chiang, J., Gong, W., Koperski, K., Li, D., Lu, Y., Rajan, A., Stefanovic, N., Xia, B. and Zaiane, O.(1996) "DBMiner : A System for Mining Knowledge in Large Relational Databases" In *Proceedings of the Second International Conference on Data Mining and Knowledge Discovery.*

Han, S. et al. (1998) "Parallel Formulations of Decision-Tree Classification Algorithms." In *Proc. of the 1998 International Conference on Parallel Processing.*

John, G. (1997). *Enhancements to the Data Mining Process.* PhD Thesis, Stanford University.

John, G. and Langley, P. (1996). "Static Versus Dynamic Sampling for Data Mining." In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining.*

Joshi, M.V., Karypis, G. and Kumar, V (1998) " ScalParc : A New Scalable and Efficient Parallel Classification Algorithm for Mining Large Datasets." In *Proceedings of the International Parallel Processing Symposium..*

Kohavi, R. and John, G. (1997) "Wrappers for Feature Subset Selection." In *Artificial Intelligence 97(1-2)* pp 273-324.

Kononenko, I. (1994). "Estimating Attributes: Analysis and Extensions of RELIEF." In *Proceedings of the European Conference on Machine Learning, 1994.*

Kufrin, R. (1997) "Generating C4.5 Production Rules in Parallel." In *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI-97), AAAI-Press.*

Kumar, V., Grama, A., Karypis, G. 1994. *Introduction to Parallel Computing: Design and Analysis of Algorithms*. Benjamin Cummings Publishing Company.

Lee, S. D., Cheung, D. W., Kao, B. (1998). “Is Sampling Useful in Data Mining? A Case Study in the Maintenance of Discovered Association Rules.” *Data Mining and Knowledge Discovery*: **2**:233-262.

Mehta, M., Agrawal, R., and Rissanen, J (1996). “SLIQ: A Fast Scalable Classifier for Data Mining.” In *Proceedings of the Fifth International Conference on Extending Database Technology*. Avignon France.

Olson, C.F. (1995) "Parallel Algorithms for Hierarchical Clustering." In *Parallel Computing*, 21(8):1313-1325

Pei, J , Han, J, and Mao, R (2000) `` CLOSET: An Efficient Algorithm for Mining Frequent Closed Itemsets (PDF) ", *In Proc. 2000 ACM-SIGMOD Int. Workshop on Data Mining and Knowledge Discovery (DMKD'00)}, Dallas, TX*

Park, J.S. , Chen, M and Yu. P.S. (1995). "An effective Hash-based Algorithm for Mining Association Rules" In *Proceedings of the ACM SIGMOD International Conference on Management of Data.*

Pearson, R. A. (1994). "A Coase Grained Parallel Induction Heuristic." In *H. Kitano, V. Kumar and C.B. Sutter, editors, Parallel Processing for Artificial Intelligence 2, Pages 207--226, Elsevier Science.*

Provost, F. and Aronis, J. (1996) "Scaling up Inductive Learning with Massive Parallelism." In

Machine Learning 23, 33-46

Provost, F., Jensen, D. and Oates, T. (1999). "Efficient Progressive Sampling." In Proceedings of *the Fifth International Conference on Knowledge Discovery and Data Mining.*

Provost, F. and Kolluri, V. (1999). "A Survey of Methods for Scaling Up Inductive Algorithms." To appear in *Data Mining and Knowledge Discovery Journal.*

Quinlan, J. (1993). *C4.5 Programs for Machine Learning.* Morgan Kaufmann.

Ramakrishnan, Ramu (1997). *Database Management Systems.* McGraw-Hill, New York.

Sarawagi, S., Thomas, S. and Agrawal, R. (1998). "Integrating Association Rule Mining with Relational Database Systems: Alternatives and Implications." In *Proceedings of the ACM SIGMOD International Conference on Management of Data*

Selim, S.Z. and Ismail, M.A. (1984). "K-Means-Type Algorithms : A Generalized Convergence Theorem and Characterization of Local Optimality" In *IEEE Transactions on Pattern Analysis and Machine Intelligence, PAMI-6:81-87*

Shafer, J., Agrawal, R. and Mehta, M. (1996). "SPRINT: A Scalable Parallel Classifier for Data Mining." In *Proceedings of the 22nd International Conference on Very Large Databases (VLDB 1996).*

Subramonian, R. and Parthasarathy, S. (1998) " A Framework for Distributed Data Mining" In *Proceedings of KDD98 Workshop on Distributed Data Mining.*

Thomas, S. and Sarawagi, S. (1998). "Mining Generalized Association Rules and Sequential Patterns Using SQL Queries." In *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining.*

Vitter, J. S. and Shriver, E. A. M. (1994). “Algorithms for Parallel Memory I: Two Level Memories.” *Algorithmica*. Volume 12, pages 110-147.

Wolpert, D.H. (1992) "Stacked Generalization" In *Journal of Neural Networks, Vol 5.*

Zaki, M.J., C. Ho, and R. Agrawal (1999). "Scalable Parallel Classification for Data Mining on Shared Memory Multiprocessors" In *Proceedings of IEEE International Conference on Data Engineering.*