Parallel Algorithms in Data Mining *

Mahesh V. Joshi, Eui-Hong (Sam) Han, George Karypis, and Vipin Kumar
Department of Computer Science,
University of Minnesota, Minneapolis, MN 55455, USA.

1 Introduction

Recent times have seen an explosive growth in the availability of various kinds of data. It has resulted in an unprecedented opportunity to develop automated data-driven techniques of extracting useful knowledge. Data mining, an important step in this process of knowledge discovery, consists of methods that discover interesting, non-trivial, and useful patterns hidden in the data [SAD$^+$93, CHY96]. The field of data mining builds upon the ideas from diverse fields such as machine learning, pattern recognition, statistics, database systems, and data visualization. But, techniques developed in these traditional disciplines are often unsuitable due to some unique characteristics of today's data-sets, such as their enormous sizes, high-dimensionality, and heterogeneity.

To date, the primary driving force behind the research in data mining has been the development of algorithms for data-sets arising in various business, information retrieval, and financial applications. Businesses can use data mining to gain significant advantages in today's competitive global marketplace. For example, retail industry is using data mining techniques to analyze buying patterns of customers, mail order business is using them for targeted marketing, telecommunication industry is using them for churn prediction and network alarm analysis, and credit card industry is using them for fraud detection. Also, recent growth of electronic commerce is generating wealths of online web data, which needs sophisticated data mining techniques.

Due to the latest technological advances, very large data-sets are becoming available in many scientific disciplines as well. The rate of production of such data-sets far outstrips the ability to analyze them manually. For example, a computational simulation running on the state-of-the-art high performance computers can generate tera-bytes of data within a few hours, whereas human analyst may take several weeks or longer to analyze and discover useful information from these data-sets. Data mining techniques hold great promises for developing new sets of tools that can be used to automatically analyze the massive data-sets resulting from such simulations, and thus help engineers and scientists unravel the causal relationships in the underlying mechanisms of the dynamic physical processes. Some other recently emerging applications of data mining can be found in the analysis and understanding of gene functions in the field of genomics, and categorization of stars and galaxies in the field of astrophysics.

The huge size of the available data-sets and their high-dimensionality make large-scale data mining applications computationally very demanding, to an extent that high-performance parallel computing is fast becoming an essential component of the solution. Moreover, the quality of the data mining results often depends directly on the amount of computing resources available. In fact, data mining applications are poised to become the dominant consumers of supercomputing in the near future. There is a necessity to develop effective parallel algorithms for various data mining techniques. However, designing such algorithms is challenging. In the rest of this chapter, we will describe the parallel formulations of two important data mining algorithms: discovery of association rules, and induction of decision trees for classification.

*This work was supported by NSF CCR-9972519, by Army Research Office contract DA/DAAG55-98-1-0441, by the DOE grant LLNL/DOE B347714, and by Army High Performance Computing Research Center contract number DAAH04-95-C-0008. Access to computing facilities was provided by AHPCRC, Minnesota Supercomputer Institute. Related papers are available via WWW at URL: http://www.cs.umn.edu/~kumar

Table 1: Transactions from supermarket.

TID	Items
1	Bread, Coke, Milk
2	Beer, Bread
3	Beer, Coke, Diaper, Milk
4	Beer, Bread, Diaper, Milk
5	Coke, Diaper, Milk

2 Parallel Algorithms for Discovering Associations

An important problem in data mining [CHY96] is discovery of associations present in the data. Such problems arise in the data collected from scientific experiments, or monitoring of physical systems such as telecommunications networks, or from transactions at a supermarket. The problem was formulated originally in the context of the transaction data at supermarket. This *market basket* data, as it is popularly known, consists of transactions made by each customer. Each transaction contains items bought by the customer (see Table 1). The goal is to see if occurrence of certain items in a transaction can be used to deduce occurrence of other items, or in other words, to find associative relationships between items. If indeed such interesting relationships are found, then they can be put to various profitable uses such as shelf management, inventory management, etc. Thus, *association rules* were born [AIS93b]. Simply put, given a set of items, association rules predict the occurrence of some other set of items with certain degree of confidence. The goal is to discover *all* such *interesting* rules. This problem is far from trivial because of the exponential number of ways in which items can be grouped together and different ways in which one can define interestingness of a rule. Hence, much research effort has been put into formulating efficient solutions to the problem.

Let T be the set of transactions where each transaction is a subset of the itemset I. Let C be a subset of I, then we define the *support count* of C with respect to T to be:

$$\sigma(C) = |\{t | t \in T, C \subseteq t\}|.$$

Thus $\sigma(C)$ is the number of transactions that contain C. An *association rule* is an expression of the form $X \overset{s,\alpha}{\Longrightarrow} Y$, where $X \subseteq I$ and $Y \subseteq I$. The *support* s of the rule $X \overset{s,\alpha}{\Longrightarrow} Y$ is defined as $\sigma(X \cup Y)/|T|$, and the confidence α is defined as $\sigma(X \cup Y)/\sigma(X)$. For example, for transactions in Table 1, the support of rule {Diaper, Milk} $\Longrightarrow$ {Beer} is $\sigma(Diaper, Milk, Beer)/5 = 2/5 = 40\%$, whereas its confidence is. $\sigma(Diaper, Milk, Beer)/\sigma(Diaper, Milk) = 2/3 = 66\%$.

The task of discovering an association rule is to find all rules $X \overset{s,\alpha}{\Longrightarrow} Y$, such that s is greater than or equal to a given minimum support threshold and α is greater than or equal to a given minimum confidence threshold. The association rule discovery is usually done in two phases. First phase finds all the *frequent* itemsets; i.e., sets satisfying the support threshold, and then they are post-processed in the second phase to find the high confidence rules. The former phase is computationally most expensive, and much research has been done in developing efficient algorithms for it. A comparative survey of all the existing techniques is given in [JHKKar]. A key feature of these algorithms lies in their method of controlling the exponential complexity of the total number of itemsets ($2^{|I|}$). Briefly, they all use the anti-monotone property of an itemset support, which states that an itemset is frequent only if all of its sub-itemsets are frequent. *Apriori* algorithm [AS94] pioneered the use of this property to systematically search the exponential space of itemsets. In an iteration k, it generates all the *candidate* k-itemsets (of length k) such that all their $(k-1)$-subsets are frequent. The number of occurrences of these candidates are then counted in the transaction database, to determine frequent k-itemsets. Efficient data structures are used to perform fast counting.

Overall, the serial algorithms such as *Apriori* have been successful on a wide variety of transaction databases. However, many practical applications of association rules involve huge transaction databases which contain a large number of distinct items. In such situations, these algorithms running on single-processor machines may take unacceptably large times. As an example, in the Apriori algorithm, if the number of candidate itemsets becomes too large, then they might not all fit in the main memory, and multiple database passes would be required within each iteration, incurring expensive I/O cost. This implies that, even with the highly effective pruning method of Apriori, the task of finding all association rules can require a lot of computational and memory resources. This is true of most of the other serial algorithms as well, and it motivates the development of parallel formulations.

Various parallel formulations have been developed so far. A comprehensive survey can be found in [JHKKar, Zak99]. These formulations are designed to effectively parallelize either or both of the computation phases: candidate generation and candidate counting. The candidate counting phase can be parallelized relatively easily by distributing the transaction database, and gathering local counts for the entire set of candidates stored on all the processors. The CD algorithm [AS96] is an example of this simple approach. It scales linearly with respect to the number of transactions; however, generation and storage of huge number of candidates on all the processors becomes a bottleneck, especially when high-dimensional problems are solved for low support thresholds using large number of processors . Other parallel formulations, such as IDD [HKK97], have been developed to solve these problems. Their key feature is to distribute the candidate itemsets to processors so as to extract the concurrency in candidate generation as well as counting phases. Various ways are employed in IDD to reduce the communication overhead, to exploit the total available memory, and to achieve reasonable load balance. IDD algorithm exhibits better scalability with respect to the number of candidates. Moreover, reduction of redundant work and ability to overlap counting computation with communication of transactions, improves its scalability with respect to number of transactions. However, it still faces problems when one desires to use large number of processors to solve the problem. As more processors are used, the number of candidates assigned to each processor decreases. This has two implications on IDD. First, with fewer number of candidates per processor, it is much more difficult to achieve load balance. Second, it results in less computation work per transaction at each processor. This reduces the overall efficiency. Further lack of asynchronous communication ability may worsen the situation.

Formulations that combine the approaches of replicating and distributing candidates so as to reduce the problems of each one, have been developed. An example is the HD algorithm of [HKK97]. Briefly, it works as follows. Consider a P-processor system in which the processors are split into G equal size groups, each containing P/G processors. In the *HD* algorithm, we execute the *CD* algorithm as if there were only P/G processors. That is, we partition the transactions of the database into P/G parts each of size $N/(P/G)$, and assign the task of computing the counts of the candidate set C_k for each subset of the transactions to each one of these groups of processors. Within each group, these counts are computed using the *IDD* algorithm. The *HD* algorithm inherits all the good features of the *IDD* algorithm. It also provides good load balance and enough computation work by maintaining minimum number of candidates per processor. At the same time, the amount of data movement in this algorithm is cut down to $1/G$ of that of *IDD*. A detailed parallel runtime analysis of HD is given in [HKK00]. It shows that HD is scalable with respect to both number of transactions and number of candidates. The analysis also proves the necessary conditions under which HD can outperform CD.

Sequential Associations The concept of association rules can be generalized and made more useful by observing another fact about transactions. All transactions have a timestamp associated with them; i.e. the time at which the transaction occurred. If this information can be put to use, one can find relationships such as if a customer bought [The C Programming Language] book today, then he/she is likely to buy a [Using Perl] book in a few days time. The usefulness of this kind of rules gave birth to the problem of discovering *sequential patterns* or *sequential associations*. In general, a sequential pattern is a sequence of item-sets with various timing constraints imposed on the occurrences of items appearing in the pattern. For example, (A) (C,B) (D) encodes a relationship that event D occurs after an event-set (C,B), which in turn occurs after event A. Prediction of events or identification of sequential rules that characterize different parts of the data, are some example applications of sequential patterns. Such patterns are not only important because they represent more powerful and predictive relationships, but they are also important from the algorithmic point of view. Bringing in the sequential relationships increases the combinatorial complexity of the problem enormously. The reason is that, the maximum number of sequences having k events is $O(m^k 2^{k-1})$, where m is the total number of distinct events in the input data. In contrast, there are only $\binom{m}{k}$ size-k item-sets possible while

Outlook	Temp(F)	Humidity(%)	Windy?	Class
sunny	75	70	true	Play
sunny	80	90	true	Don't Play
sunny	85	85	false	Don't Play
sunny	72	95	false	Don't Play
sunny	69	70	false	Play
overcast	72	90	true	Play
overcast	83	78	false	Play
overcast	64	65	true	Play
overcast	81	75	false	Play
rain	71	80	true	Don't Play
rain	65	70	true	Don't Play
rain	75	80	false	Play
rain	68	80	false	Play
rain	70	96	false	Play

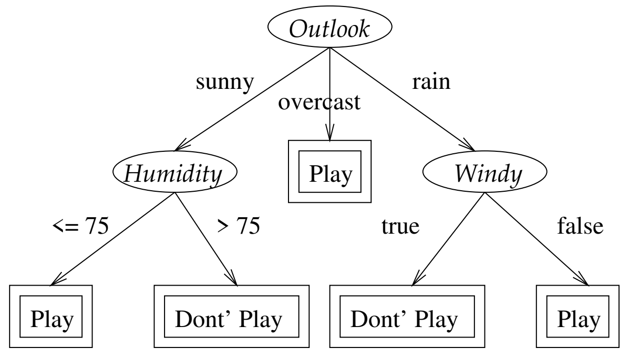

Figure 1: A small training data set [Qui93] and its final classification decision tree.

discovering non-sequential associations from m distinct items. Designing parallel algorithms for discovering sequential associations is equally important and challenging. In many situations, the techniques used in parallel algorithms for discovering standard non-sequential associations can be extended easily. However, different issues and challenges arise specifically due to the sequential nature and various ways in which interesting sequential associations can be defined. Details of various serial and parallel formulations and algorithms for finding such associations can be found in [JKK99, JHKKar].

3 Parallel Algorithms for Induction of Decision Tree Classifiers

Classification is an important data mining problem. The input to the problem is a data-set called the training set, which consists of a number of examples each having a number of attributes. The attributes are either *continuous*, when the attribute values are ordered, or *categorical*, when the attribute values are unordered. One of the categorical attributes is called the *class label* or the *classifying attribute*. The objective is to use the training set to build a model of the class label based on the other attributes such that the model can be used to classify new data not from the training data-set. Application domains include retail target marketing, fraud detection, and design of telecommunication service plans. Several classification models like neural networks [Lip87], genetic algorithms [Gol89], and decision trees [Qui93] have been proposed. Decision trees are probably the most popular since they obtain reasonable accuracy [DMT94] and they are relatively inexpensive to compute.

Most of the existing induction–based algorithms like *C4.5* [Qui93], *CDP* [AIS93a], *SLIQ* [MAR96], and *SPRINT* [SAM96] use Hunt's method [Qui93] as the basic algorithm. Here is its recursive description for constructing a decision tree from a set T of training cases with classes denoted $\{C_1, C_2, \ldots, C_k\}$.

Case 1 T contains cases all belonging to a single class C_j. The decision tree for T is a leaf identifying class C_j.

Case 2 T contains cases that belong to a mixture of classes. A test is chosen, based on a single attribute, that has one or more mutually exclusive outcomes $\{O_1, O_2, \ldots, O_n\}$. Note that in many implementations, n is chosen to be 2 and this leads to a binary decision tree. T is partitioned into subsets $T_1, T_2, \ldots, T_n$, where T_i contains all the cases in T that have outcome O_i of the chosen test. The decision tree for T consists of a decision node identifying the test, and one branch for each possible outcome. The same tree building machinery is applied recursively to each subset of training cases.

Case 3 T contains no cases. The decision tree for T is a leaf, but the class to be associated with the leaf must be determined from information other than T. For example, *C4.5* chooses this to be the most frequent class at the parent of this node.

Figure 1 shows a training data set with four data attributes and two classes and its classification decision tree constructed using the Hunt's method. In the case 2 of Hunt's method, a test based on a single attribute

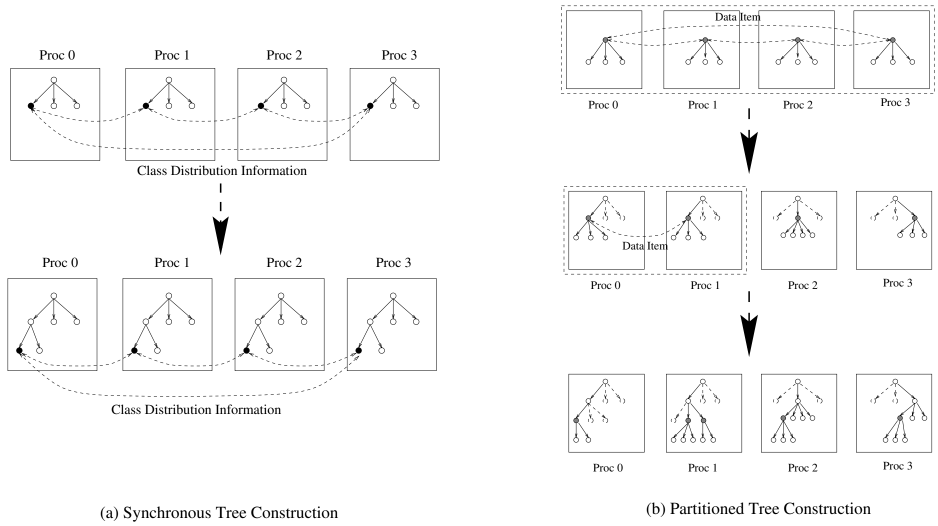

Figure 2: Synchronous Tree Construction Approach and Partitioned Tree Construction Approach

is chosen for expanding the current node. The choice of an attribute is normally based on the entropy gains [Qui93] of the attributes. The entropy of an attribute, calculated from class distribution information, depicts the classification power of the attribute by itself. The best attribute is selected as a test for the node expansion.

Highly parallel algorithms for constructing classification decision trees are desirable for dealing with large data sets in reasonable amount of time. Classification decision tree construction algorithms have natural concurrency, as once a node is generated, all of its children in the classification tree can be generated concurrently. Furthermore, the computation for generating successors of a classification tree node can also be decomposed by performing data decomposition on the training data. Nevertheless, parallelization of the algorithms for construction the classification tree is challenging for the following reasons. First, the shape of the tree is highly irregular and is determined only at runtime. Furthermore, the amount of work associated with each node also varies, and is data dependent. Hence any static allocation scheme is likely to suffer from major load imbalance. Second, even though the successors of a node can be processed concurrently, they all use the training data associated with the parent node. If this data is dynamically partitioned and allocated to different processors that perform computation for different nodes, then there is a high cost for data movements. If the data is not partitioned appropriately, then performance can be bad due to the loss of locality.

Several parallel formulations of classification decision tree have been proposed recently [Pea94, GAR96, SAM96, CDG^{+}97, Kuf97, JKK98, SHKS99]. In this section, we present two basic parallel formulations for the classification decision tree construction and a hybrid scheme that combines good features of both of these approaches described in [SHKS99]. Most of other parallel algorithms are similar in nature to these two basic algorithms, and their characteristics can be explained using these two basic algorithms. For these parallel formulations, we focus our presentation for discrete attributes only. The handling of continuous attributes is discussed separately. In all parallel formulations, we assume that N training cases are randomly distributed to P processors initially such that each processor has N/P cases.

Synchronous Tree Construction Approach In this approach, all processors construct a decision tree synchronously by sending and receiving class distribution information of local data. Figure 2 (a) shows the overall picture. The root node has already been expanded and the current node is the leftmost child of the root (as shown in the top part of the figure). All the four processors cooperate to expand this node to have two child nodes. Next, the leftmost node of these child nodes is selected as the current node (in the bottom of the figure) and all four processors again cooperate to expand the node.

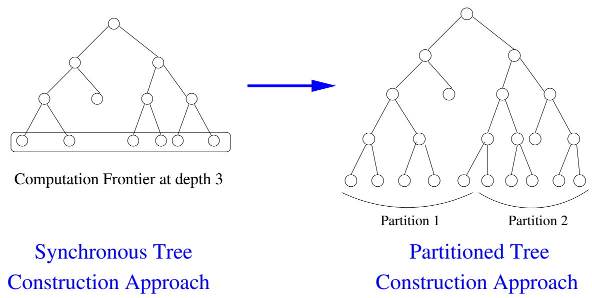

Figure 3: Hybrid Tree Construction Approach

Partitioned Tree Construction Approach In this approach, whenever feasible, different processors work on different parts of the classification tree. In particular, if more than one processors cooperate to expand a node, then these processors are partitioned to expand the successors of this node. Figure 2 (b) shows an example. First (at the top of the figure), all four processors cooperate to expand the root node just like they do in the synchronous tree construction approach. Next (in the middle of the figure), the set of four processors is partitioned in three parts. The leftmost child is assigned to processors 0 and 1, while the other nodes are assigned to processors 2 and 3, respectively. Now these sets of processors proceed independently to expand these assigned nodes. In particular, processors 2 and processor 3 proceed to expand their part of the tree using the serial algorithm. The group containing processors 0 and 1 splits the leftmost child node into three nodes. These three new nodes are partitioned in two parts (shown in the bottom of the figure); the leftmost node is assigned to processor 0, while the other two are assigned to processor 1. From now on, processors 0 and 1 also independently work on their respective subtrees.

Hybrid Parallel Formulation The hybrid parallel formulation has elements of both schemes. The *Synchronous Tree Construction Approach* incurs high communication overhead as the frontier gets larger. The *Partitioned Tree Construction Approach* incurs cost of load balancing after each step. The hybrid scheme keeps continuing with the first approach as long as the communication cost incurred by the first formulation is not too high. Once this cost becomes high, the processors as well as the current frontier of the classification tree are partitioned into two parts. Figure 3 shows one example of this parallel formulation. At the classification tree frontier at depth 3, no partitioning has been done and all processors are working cooperatively on each node of the frontier. At the next frontier at depth 4, partitioning is triggered, and the nodes and processors are partitioned into two partitions.

A key element of the algorithm is the criterion that triggers the partitioning of the current set of processors (and the corresponding frontier of the classification tree). If partitioning is done too frequently, then the hybrid scheme will approximate the partitioned tree construction approach, and thus will incur too much data movement cost. If the partitioning is done too late, then it will suffer from high cost for communicating statistics generated for each node of the frontier, like the synchronized tree construction approach. In the hybrid algorithm, the splitting is performed when the accumulated cost of communication becomes equal to the cost of moving records and load balancing in the splitting phase.

The size and shape of the classification tree varies a lot depending on the application domain and training data set. Some classification trees might be shallow and the others might be deep. Some classification trees could be skinny others could be bushy. Some classification trees might be uniform in depth while other trees might be skewed in one part of the tree. The hybrid approach adapts well to all types of classification trees. If the decision tree is skinny, the hybrid approach will just stay with the *Synchronous Tree Construction Approach*. On the other hand, it will shift to the *Partitioned Tree Construction Approach* as soon as the tree becomes bushy. If the tree has a big variance in depth, the hybrid approach will perform dynamic load balancing with processor groups to reduce processor idling.

Handling Continuous Attributes The approaches described above concentrated primarily on how the tree is constructed in parallel with respect to the issues of load balancing and reducing communication overhead. The discussion was simplified by the assumption of absence of continuous-valued attributes. Presence of continuous attributes can be handled in two ways. One is to perform intelligent discretization, either once in the beginning or at each node as the tree is being induced, and treat them as categorical attributes. Another, more popular approach is to use decisions of the form $A < x$ and $A \geq x$, directly on the values x of continuous attribute A. The decision value of x needs to be determined at each node. For efficient search of x, most algorithms require the attributes to be sorted on values, such that one linear scan can be done over all the values to evaluate the best decision. Among various different algorithms, the approach taken by SPRINT algorithm[SAM96], which sorts each continuous attribute only once in the beginning, is proven to be efficient for large datasets. The sorted order is maintained throughout the induction process, thus avoiding the possibly excessive costs of re-sorting at each node. A separate list is kept for each of the attributes, in which the record identifier is associated with each sorted value. The key step in handling continuous attributes is the proper assignment of records to the children node after a splitting decision is made. Implementation of this offers the design challenge. SPRINT builds a mapping between a record identifier and the node to which it goes to based on the splitting decision. The mapping is implemented as a hash table and is probed to split the attribute lists in a consistent manner.

Parallel formulation of the SPRINT algorithm falls under the category of synchronous tree construction design. The multiple sorted lists of continuous attributes are split in parallel by building the entire hash table on all the processors. However, with this simple-minded way of achieving a consistent split, the algorithm incurs a communication overhead of $O(N)$ per processor. Since, the serial runtime of the induction process is $O(N)$, SPRINT becomes unscalable with respect to runtime. It is unscalable in memory requirements also, because the total memory requirement per processor is $O(N)$, as the size of the hash table is of the same order as the size of the training dataset for the upper levels of the decision tree, and it resides on every processor. Another parallel algorithm, ScalParC [JKK98], solves this scalability problem. It employs a distributed hash table to achieve a consistent split. The communication structure, used to construct and access this hash table, is motivated by the parallel sparse matrix-vector multiplication algorithms. It is shown in [JKK98] that with the proper implementation of the parallel hashing, the overall communication overhead does not exceed $O(N)$, and the memory required does not exceed $O(N/p)$ per processor. Thus, ScalParC is scalable in runtime as well as memory requirements.

4 Conclusion

This chapter presented an overview of parallel algorithms for two of the commonly used data mining techniques. Key issues such as load balancing, attention to locality, extracting maximal concurrency, avoiding hot spots in contention, and minimizing parallelization overhead are just as central to these parallel formulations as they are to the traditional scientific parallel algorithms. In fact, in many cases, the underlying kernels are identical to well known algorithms, such as sparse matrix-vector product.

To date, the parallel formulations of many decision-tree induction and association rule discovery algorithms are reasonably well-understood. Relatively less work has been done on the parallel algorithms for other data mining techniques such as clustering, rule-based classification algorithms, deviation detection, and regression. Some possible areas of further research include parallelization of many emerging new and improved serial data mining algorithms, further analysis and refinements of existing algorithms for scalability and efficiency, designs targetted for shared memory and distributed shared memory machines equipped with symmetric multiprocessors, and efficient integration of parallel algorithms with parallel database systems.

References

[AIS93a] R. Agrawal, T. Imielinski, and A. Swami. Database mining: A performance perspective. *IEEE Transactions on Knowledge and Data Eng.*, 5(6):914–925, December 1993.

[AIS93b] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In *Proc. of 1993 ACM-SIGMOD Int. Conf. on Management of Data*, Washington, D.C., 1993.

[AS94] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proc. of the 20th VLDB Conference*, pages 487–499, Santiago, Chile, 1994.

[AS96] R. Agrawal and J.C. Shafer. Parallel mining of association rules. *IEEE Transactions on Knowledge and Data Eng.*, 8(6):962–969, December 1996.

[CDG+97] J. Chattratichat, J. Darlington, M. Ghanem, Y. Guo, H. Huning, M. Kohler, J. Sutiwaraphun, H.W. To, and D. Yang. Large scale data mining: Challenges and responses. In *Proc. of the Third Int'l Conference on Knowledge Discovery and Data Mining*, 1997.

[CHY96] M.S. Chen, J. Han, and P.S. Yu. Data mining: An overview from database perspective. *IEEE Transactions on Knowledge and Data Eng.*, 8(6):866–883, December 1996.

[DMT94] D.J. Spiegelhalter D. Michie and C.C. Taylor. *Machine Learning, Neural and Statistical Classification*. Ellis Horwood, 1994.

[GAR96] S. Goil, S. Aluru, and S. Ranka. Concatenated parallelism: A technique for efficient parallel divide and conquer. In *Proc. of the Symposium of Parallel and Distributed Computing (SPDP'96)*, 1996.

[Gol89] D. E. Goldberg. *Genetic Algorithms in Search, Optimizations and Machine Learning*. Morgan-Kaufman, 1989.

[HKK97] E.-H. Han, G. Karypis, and V. Kumar. Scalable parallel data mining for association rules. In *Proc. of 1997 ACM-SIGMOD Int. Conf. on Management of Data*, Tucson, Arizona, 1997.

[HKK00] E.H. Han, G. Karypis, and V. Kumar. Scalable parallel data mining for association rules. *IEEE Transactions on Knowledge and Data Eng.*, 12(3), May/June 2000.

[JHKKar] M. V. Joshi, E.-H. Han, G. Karypis, and V. Kumar. Efficient parallel algorithms for mining associations. In M. J. Zaki and C.-T. Ho, editors, *Lecture Notes in Computer Science: Lecture Notes in Artificial Intelligence (LNCS/LNAI)*, volume 1759. Springer-Verlag, To Appear.

[JKK98] M. V. Joshi, G. Karypis, and V. Kumar. ScalParC: A new scalable and efficient parallel classification algorithm for mining large datasets. In *Proc. of the International Parallel Processing Symposium*, 1998.

[JKK99] M. V. Joshi, G. Karypis, and V. Kumar. Universal formulation of sequential patterns. Technical Report TR 99-021, Department of Computer Science, University of Minnesota, Minneapolis, 1999.

[Kuf97] R. Kufrin. Decision trees on parallel processors. In J. Geller, H. Kitano, and C.B. Suttner, editors, *Parallel Processing for Artificial Intelligence 3*. Elsevier Science, 1997.

[Lip87] R. Lippmann. An introduction to computing with neural nets. *IEEE ASSP Magazine*, 4(22), April 1987.

[MAR96] M. Mehta, R. Agrawal, and J. Rissanen. SLIQ: A fast scalable classifier for data mining. In *Proc. of the Fifth Int'l Conference on Extending Database Technology*, Avignon, France, 1996.

[Pea94] R.A. Pearson. A coarse grained parallel induction heuristic. In H. Kitano, V. Kumar, and C.B. Suttner, editors, *Parallel Processing for Artificial Intelligence 2*, pages 207–226. Elsevier Science, 1994.

[Qui93] J. Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Mateo, CA, 1993.

[SAD+93] M. Stonebraker, R. Agrawal, U. Dayal, E. J. Neuhold, and A. Reuter. DBMS research at a crossroads: The vienna update. In *Proc. of the 19th VLDB Conference*, pages 688–692, Dublin, Ireland, 1993.

[SAM96] J. Shafer, R. Agrawal, and M. Mehta. SPRINT: A scalable parallel classifier for data mining. In *Proc. of the 22nd VLDB Conference*, 1996.

[SHKS99] A. Srivastava, E.-H. Han, V. Kumar, and V. Singh. Parallel formulations of decision-tree classification algorithms. *Data Mining and Knowledge Discovery: An International Journal*, 3(3):237–261, September 1999.

[Zak99] M. J. Zaki. Parallel and distributed association mining: A survey. *IEEE Concurrency (Special Issue on Data Mining)*, December 1999.