

ATHABASCA UNIVERSITY

SURVEY OF EXISTING DATA MINING TECHNIQUES, METHODS AND  
GUIDELINES WITHIN THE CONTEXT OF ENTERPRISE DATA WAREHOUSE

BY

ARMAN KANOONI

A thesis project submitted in partial fulfillment

Of the requirements for the degree of

MASTER OF SCIENCE in INFORMATION SYSTEMS

Athabasca, Alberta

August 2004

© Arman Kanooni, 2004

ATHABASCA UNIVERSITY

The undersigned certify that they have read and recommend for acceptance the thesis project "SURVEY OF EXISTING DATA MINING TECHNIQUES, METHODS AND GUIDELINES WITHIN THE CONTEXT OF ENTERPRISE DATA WAREHOUSE" submitted by Arman Kanooni in partial fulfillment of the requirements for the degree of MASTER OF SCIENCE in INFORMATION SYSTEMS.

\_\_\_\_\_  
Dr. Lewis Varga, Supervisor

\_\_\_\_\_  
Dr. Mohamed Ally, Chair

\_\_\_\_\_  
Dr. Chunsheng Yang, Examiner

Date: \_\_\_\_\_

## DEDICATION

To my wife Alice and my son Nathaniel.

## **ABSTRACT**

This essay surveys existing data mining techniques, methods, and guidelines within the context of enterprise data warehouses. First, an overview of traditional data mining algorithms is presented. These algorithms have been designed for sequential computers (computers with a single processor) and cannot be scaled to process the huge amounts of data present in enterprise data warehouses. Nevertheless, they will serve the purpose of introducing the goals of data mining in general. In recent years, many of the traditional data mining algorithms have been adapted to high performance computers. The main topic of this essay is a survey of these parallel data mining algorithms along with a survey of the corresponding parallel computing architectures.

## **ACKNOWLEDGMENTS**

First, I would like to thank my supervisor, Lewis Varga, for his guidance. His active interest and valuable feedback on this topic has helped me to prioritize my workload and focus on the problem at hand. Second, I must thank my employer, The Boeing Company, for providing financial support throughout my Master of Science in Information Systems studies and this thesis at Athabasca University through the Learning Together Program.

## TABLE OF CONTENTS

CHAPTER 1 INTRODUCTION .....	1
1.1 The Data Mining Process.....	2
1.2 Taxonomy of Parallel Computers.....	6
1.3 High Performance Data Warehouses .....	10
CHAPTER 2 TRADITIONAL DATA MINING ALGORITHMS.....	14
2.1 Classification (Decision Trees).....	14
2.2 Association Rules .....	16
2.3 Sequence Patterns .....	20
2.4 Clustering.....	22
2.5 Partitioning algorithms.....	23
2.5.1 K-means algorithm .....	24
2.5.2 K-medoids algorithm.....	24
2.6 Neural Nets .....	26
2.7 Traditional Statistical Methods .....	28
CHAPTER 3 PARALLEL DATA MINING ALGORITHMS .....	33
3.1 Parallel Algorithms for Classification .....	33
3.1.1 SPRINT .....	33
3.1.2 Synchronous tree construction approach .....	35
3.1.3 Partitioned tree construction approach.....	37
3.1.4 Hybrid Algorithms .....	38
3.1.5 ScalParC design approach.....	40
3.2 Parallel Algorithms for Association Rules.....	41

3.2.1 Count distribution.....	41
3.2.2 Data Distribution .....	42
3.2.3 Intelligent data distribution.....	43
3.2.4 Hybrid distribution.....	45
3.4 Parallel Algorithms for Clustering .....	48
3.4.1 Parallel k-means.....	48
3.4.2 Parallel k-nearest neighbors.....	49
CHAPTER 4 ARCHITECTURES AND SYSTEMS FOR HIGH-PERFORMANCE DATA MINING .....	50
4.1 Data-Oriented Approaches .....	50
4.2 Algorithm-Oriented Approaches.....	51
4.3 Overview: Parallel Systems .....	52
4.3.1 Distributed-memory machines.....	52
4.3.2 Shared-memory machines .....	52
4.4 Task versus Data Parallelism.....	54
4.5 High-Performance Data Mining.....	54
4.6 Main Data Mining Tasks and Techniques .....	55
4.6.1 Parallelism in Data Mining Techniques.....	56
CHAPTER 5 CHALLENGES AND FUTURE RESEARCH.....	60
5.1 Algorithmic Issues.....	60
5.2 Systems Issues.....	62
5.3 Summary.....	62
REFERENCES.....	64

## LIST OF TABLES

Table 1 <i>The Weather Data</i> .....	17
Table 2 <i>Item Sets with the Required Support in Table 1</i> .....	18
Table 3 <i>Customer Record</i> .....	29
Table 4 <i>SPRINT Data Set Example</i> .....	34
Table 5 <i>SPRINT Applied to Data Set</i> .....	34
Table 6 <i>Data Mining Tasks and Techniques</i> .....	56

## LIST OF FIGURES

<i>Figure 1.</i> Major steps in KDD process .....	3
<i>Figure 2.</i> Data mining methods.....	4
<i>Figure 3.</i> Single instructions multiple data. ....	7
<i>Figure 4.</i> Multiple instructions multiple data.....	8
<i>Figure 5.</i> MIMD shared memory processors.....	9
<i>Figure 6.</i> MIMD distributed memory processors. ....	9
<i>Figure 7.</i> A typical parallel technology for an enterprise data warehouse.....	12
<i>Figure 8.</i> Classification methods.....	15
<i>Figure 9.</i> K-means clustering method. ....	24
<i>Figure 10.</i> K-medoids algorithm.....	25
<i>Figure 11.</i> Neural nets. ....	27
<i>Figure 12.</i> Feed forward neural nets.....	28
<i>Figure 13.</i> Frequency histogram for customers with various eye colors. ....	29
<i>Figure 14.</i> Histogram for customers of different ages grouped by year. ....	30
<i>Figure 15.</i> Linear regressions. ....	32
<i>Figure 16.</i> Synchronous tree construction approach. ....	36
<i>Figure 17.</i> Partitioned tree construction approach.....	38
<i>Figure 18.</i> Hybrid approach. ....	39
<i>Figure 19.</i> Load balancing. ....	39
<i>Figure 20.</i> Communication structure of parallel sparse matrix-vector algorithms. .....	40
<i>Figure 21.</i> Count distribution algorithm.....	42

<i>Figure 22.</i> Data distribution algorithm.....	43
<i>Figure 23.</i> Intelligent data distribution algorithm. ....	44
<i>Figure 24.</i> Hybrid distribution algorithm in 3X4 processor mesh ( $G=3, P=12$ )....	47
<i>Figure 25.</i> Distributed memory architecture (shared-nothing).....	52
<i>Figure 26.</i> Shared disk architecture.....	53
<i>Figure 27.</i> Shared memory architecture (shared-everything). ....	53
<i>Figure 28.</i> Cluster of SMPs.....	53
<i>Figure 29.</i> Task parallelism versus data parallelism. ....	54

## **CHAPTER 1**

### **INTRODUCTION**

Today's manufacturing, engineering, business, and computing processes in public and private organizations around the world are generating massive amounts of data. This explosive growth of data has outpaced the ability to interpret and digest the data. Therefore, data mining techniques and tools for automated data analysis and knowledge discovery are needed. Today's enterprise data warehouse (EDW) focuses on developing, extending, and incorporating knowledge discovery technology into tools and methods for data analysis, including aspects of data modeling, algorithms, and visualization. Knowledge gained by uncovering relationships and structure in the data will enable better understanding of customers, suppliers, and internal as well as external processes. This helps process owners to identify problems, reduce defects, and improve cost, aiding continuous quality improvement.

The proliferation of computer databases, online automatic data collection, and increased storage capacity has, in some situations, led to explosive growth in the size and complexity of data warehouses. The gigantic sizes of these enterprise databases overwhelm standard analysis methods, making it difficult to find useful information, knowledge, and relationships in the data.

There are many software tools implementing data mining and knowledge discovery techniques that are designed to work efficiently over large amounts of data and carry out simple analyses to uncover relationships in the data. The

users and analysts may then perform further investigations and data analysis to confirm or better understand the uncovered relationships.

Once the volume of data surpasses a couple of terabytes, it is necessary to use parallel computers to handle scalability and performance issues. The traditional sequential computing environment cannot produce results in a timely manner; therefore, it is imperative to use new parallel data mining algorithms designed to work in parallel computing environments and to produce results faster.

### ***1.1 The Data Mining Process***

Data mining is defined as "the non trivial extraction of implicit, previously unknown, and potentially useful information from data" (Frawley, Piatetsky-Shapiro, & Matheus, 1991). But extracting useful patterns from data is just the beginning of an iterative process to turn data into information, the information into knowledge, and the knowledge into action. Figure 1 shows the major steps in the process of knowledge discovery in databases (KDD) as defined by Fayyad, Piatetsky-Shapiro, Smyth, and Uthurusamy (1996). These steps are as follows:

1. The target data are selected based on end-user goals and understanding of subject area and the number of variables.
2. These target data now have to be preprocessed to clean up the bad data, decide on missing data, and handle time sequencing and slowly changing data.

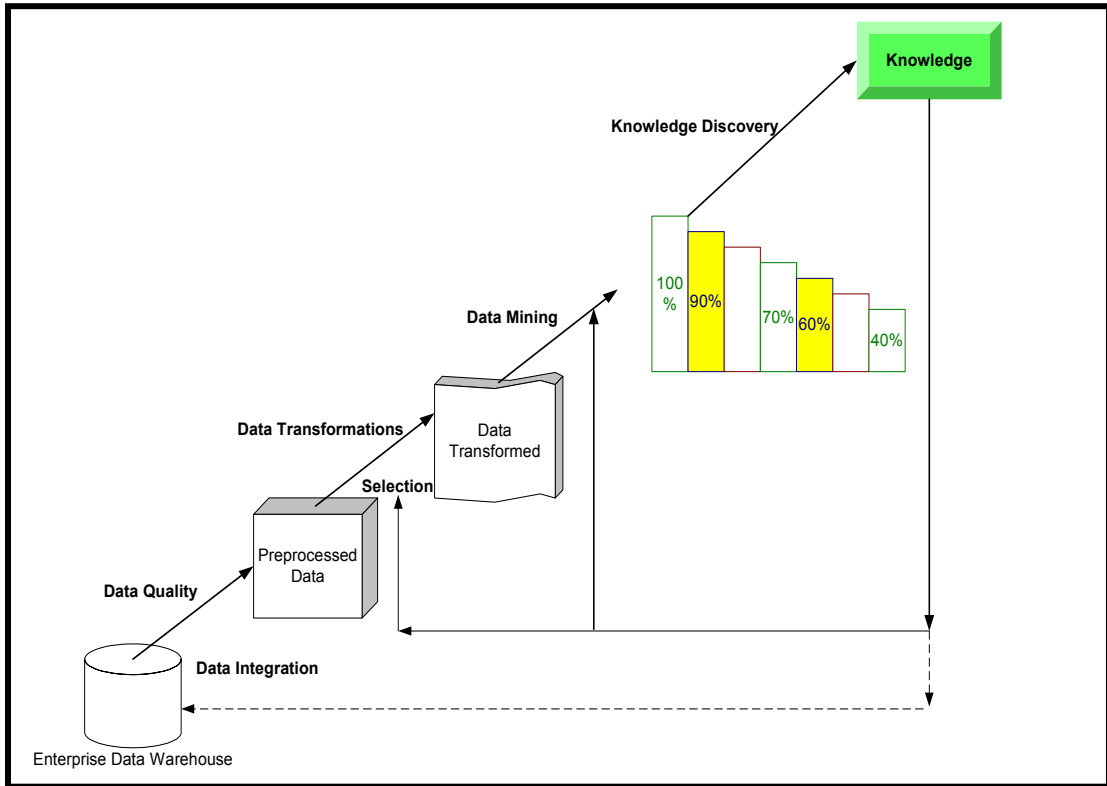


Figure 1. Major steps in KDD process

3. The data transformation step allows an effective number of variables to be created for data mining purposes.
4. Then, it must be decided whether the goal of the knowledge discovery process is classification, regression, clustering, and so on. An appropriate data mining algorithm is chosen for searching for patterns in the data.
5. The end user has to interpret the returned pattern and evaluate the knowledge obtained by the data mining process.
6. It is possible to conduct further iteration based on the interpretation of mined patterns.

7. Finally, the new knowledge has to be integrated into previous knowledge to resolve any conflict and inconsistency.

The KDD process can use significant iteration and may contain loops between any two steps. The data mining component of the KDD process uses particular data mining methods. In general, we can categorize data mining methods into two distinct classes: predictive and descriptive. On the one hand, predictive methods use past experience to learn in order to predict future behavior (Berry & Linoff, 2000).

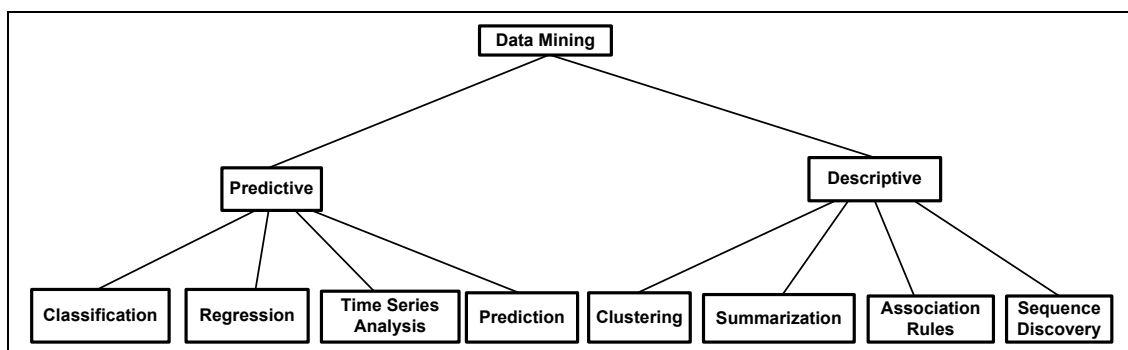


Figure 2. Data mining methods.

On the other hand, descriptive data mining methods help to understand what is going on inside the data at the present time and therefore augment the understanding of the people, products, or processes that generated the data (see Figure 2). Predictive data mining methods in common use are classification, regression, time series analysis, and prediction:

1. *Classification* is the process of dividing a data set into mutually exclusive groups such that the members of each group are as close as possible to one another and different groups are as far as possible from one another,

where distance is measured with respect to specific variable(s) the researcher is trying to predict.

2. *Regression* is a statistical technique used to find the best-fitting linear relationship between a target (dependent) variable and its predictors (independent variables).
3. *Time series analysis* is the analysis of a sequence of measurements made at specified time intervals. Time is usually the dominating dimension of the data.
4. *Prediction* is a structure and process for predicting the values of specified variables in a data set.

The descriptive data mining methods in common use are clustering, summarization, association rules, and sequence discovery.

1. *Clustering* is the process of dividing a data set into mutually exclusive groups such that the members of each group are as close as possible to one another and different groups are as far as possible from one another, where distance is measured with respect to all available variables.
2. *Summarization* maps data into subsets with associated simple descriptions.
3. *Association rules* address a class of problems typified by a market-basket analysis. Classic market-basket analysis treats the purchase of a number of items as a single transaction. The goal is to find trends across large numbers of transactions that can be used to understand and exploit natural buying patterns. This information can be used to adjust

inventories, modify floor or shelf layouts, or introduce targeted promotional activities to increase overall sales or move specific products (Moxon, 1996).

4. *Sequence discovery* is the process of finding sequential patterns in the data. Traditional market-basket analysis deals with a collection of items as part of a point-in-time transaction. Sequence discovery may be considered as a variant of market-basket analysis where some additional information ties together a sequence of purchases (for example, an account number, a credit card, or a frequent buyer or flyer number) in a time series.

Parallel data mining algorithms are closely tailored to the computing environment in which they are executed. A prerequisite to a survey of parallel data mining algorithms therefore is a survey of the architecture of parallel computers.

### ***1.2 Taxonomy of Parallel Computers***

As the size of stored data keeps growing with no end in sight, it becomes imperative to increase the processing power of computers. Parallel processing machines are becoming more common and less expensive. Software is increasingly available to take advantage of parallel and distributed resources for identifying useful patterns in data. The successful introduction of parallel relational database management software by major vendors such as IBM, Informix, NCR, Oracle, Red Brick, Sybase, and Tandem has brought the power

of parallel processing into enterprise data warehouses. These parallel computers provide an excellent environment for high performance data mining.

An overview of parallel computer architectures will help to understand different options and choices available to implement a data mining solution in the context of a large enterprise wide data warehouse. Parallel architectures consist of three building blocks: processors (P), memory modules, and interconnection networks, which connect the processors to each other and sometimes to memory modules as well. Parallel architectures vary in the way they arrange these three main logical components. Flynn (1972) used the stream concept for describing a computer's structure.

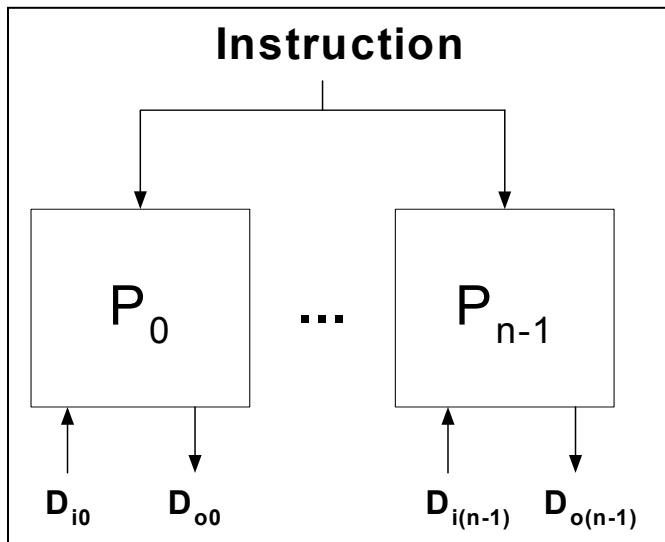
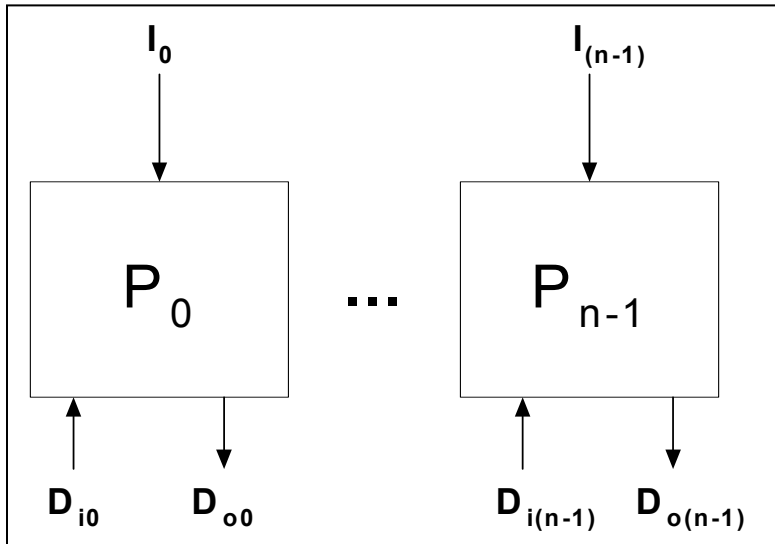


Figure 3. Single instructions multiple data.

In single instructions multiple data (SIMD), a single processor executes a single instruction stream, but broadcasts each instruction to be executed to a number of data processors (see Figure 3). These data processors interpret the

instruction's addresses either as local addresses in their own local memories or as global addresses. At a given instant of time, a given P is either active and doing exactly the same thing as all the other active processes or it is idle.



*Figure 4.* Multiple instructions multiple data.

In multiple instructions multiple data (MIMD) machines, processors are full-fledged central processing units (CPUs) with both a control unit and an ALU (see Figure 4). Thus, each P is capable of executing its own program at its own pace. The executions of MIMD are asynchronous. The processors of a MIMD may have shared memory (Figure 5) or distributed memory (Figure 6).

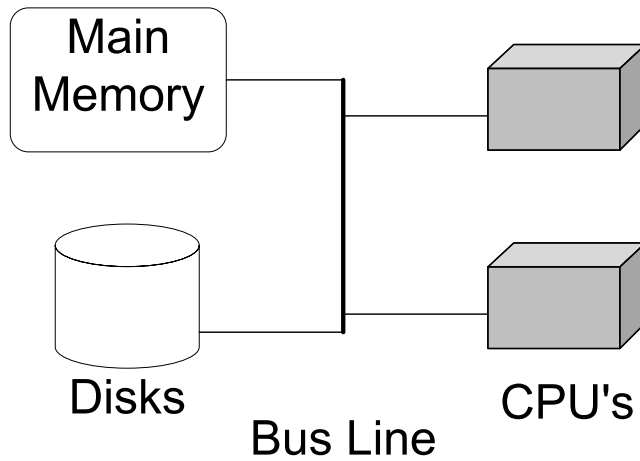


Figure 5. MIMD shared memory processors.

In shared memory MIMD, several P's share the main memory connected via bus line or network. Two problems of this architecture are the following:

1. Processes share variables in the memory; thus extra effort is required to maintain the data consistency.
2. It is difficult to program in this environment because of excessive memory management required for concurrent access to shared memory area.

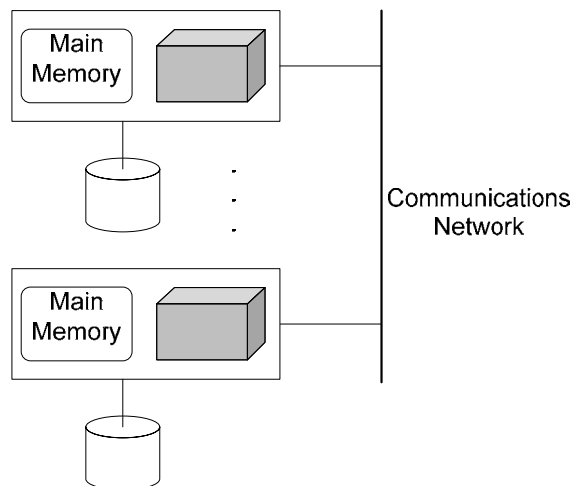


Figure 6. MIMD distributed memory processors.

The following are some problems of parallel architecture:

1. Data distribution among multiple processors is very complex.
2. Task parallelization requires decomposing high-level data mining tasks into subtasks that can be executed in a parallel manner.
3. There are complex scheduling and recovery issues.
4. Parallel processors often share the same resources (disk, memory, etc.), causing a slowdown from the interference of each new process as it competes with existing processes for commonly held resources.
5. Data skew occurs when workload is not balanced adequately among processors.
6. Possible communications overload, as processes may have to communicate with other processes. In a synchronized environment, one process may wait for another process before execution can proceed.
7. In certain phases of the parallel algorithms, the results from processors must be gathered into a single processor to consolidate partial results into final result.

### ***1.3 High Performance Data Warehouses***

One of the most important characteristics of an enterprise wide data warehouse environment is the continual growth of vast amounts of data. There are many design and software techniques that can help with the complex management of huge amounts of data. For example, it is possible to store data

on multiple storage media. One can summarize data when details of the data are not important; and encoding and partitioning of the data can provide some relief in processing and response time. While these approaches are useful and should be considered in any hardware configuration, parallel computing technology provides an ultimate processing powerhouse that cannot be achieved using sequential computing architecture. As reviewed in the previous section, parallel architecture uses processors that are tightly coupled together but work independently. Figure 7 shows a typical configuration of CPUs working together and managing data in parallel (Morse & Isaac, 1998). There are five components in this figure: the transaction or query request, the memory or queue the query goes in, the network connection, the processor, and the data controlled by the processor. A typical use case for this system is as follows:

1. A request enters the system.
2. A processor analyzes the request and determines its location in the memory.
3. If the query is large, then it is broken down into a series of smaller requests for data and processing that in turn are routed to the appropriate processors.
4. The request enters the memory.
5. The processor starts the execution of the request.
6. When the work is done, a report is sent to the workstation.

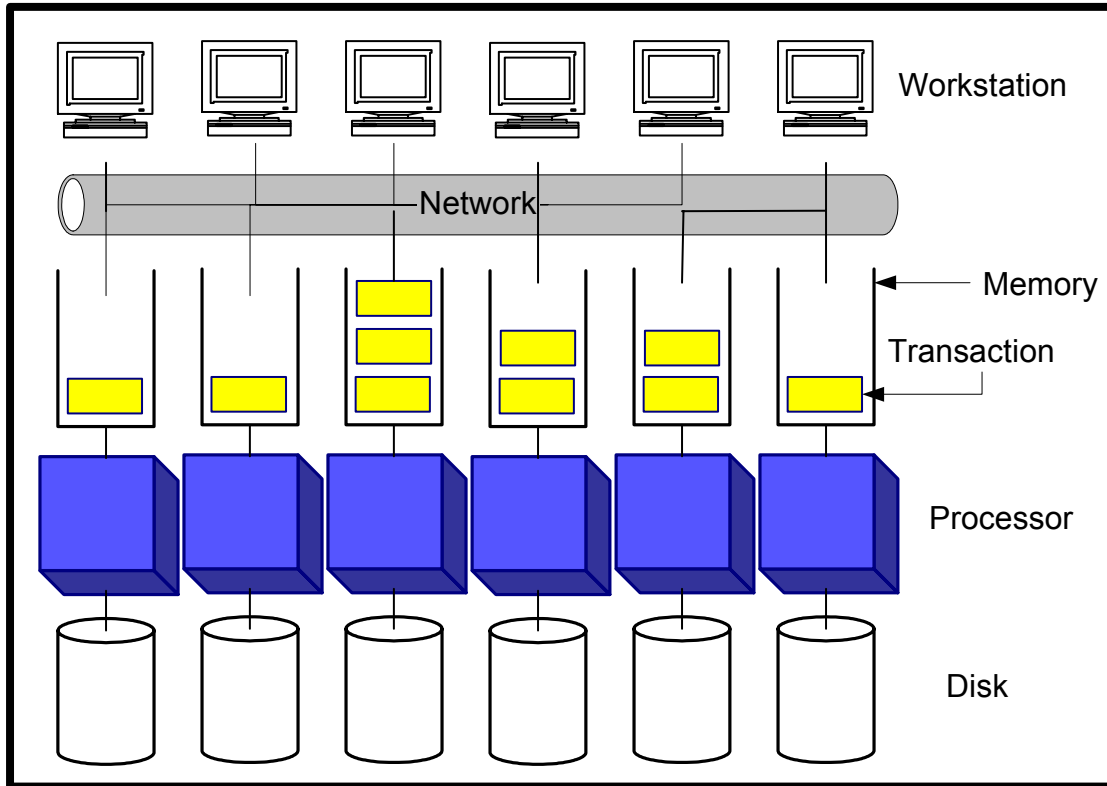


Figure 7. A typical parallel technology for an enterprise data warehouse.

The advantage of this parallel architecture is that while one processor is executing the queries, another processor can be launched to respond to the requests that have been channeled to it independently of the work done by other processors. This independence of processing power enables large amount of data to be managed in enterprise data warehouses; to manage more data, more processors can be added to the tightly networked configuration. The performance of parallel architecture depends on the data being distributed evenly across the different processors. If there is an imbalance in the distribution of data across the parallel processors and there is a corresponding imbalance in the workload, then a so-called hot spot develops and the effectiveness of the parallel architecture is jeopardized. A remedy for a hot spot requires the redistribution of data to other

(or more) processors and thus depends on a foreknowledge of the usage of data. Finding usage patterns in an enterprise data warehouse with ad hoc query capability is a difficult task and requires constant monitoring by database administrators.

Finally, one of the main benefits of the enterprise data warehouse founded on parallel technology is that data can be accessed quickly and randomly. However, parallel management of data is expensive.

## CHAPTER 2

### TRADITIONAL DATA MINING ALGORITHMS

#### ***2.1 Classification (Decision Trees)***

Classification is a typical goal of data mining. For example, customers of an insurance company can be classified as high-risk, medium-risk, and low-risk insurance policy holders. To formalize the classification problem, we need some terminology. Let us abstract a database  $D$  as a set of  $n$  records (or transactions, or instances),  $D = \{t_1, t_2, \dots, t_n\}$ , where each record  $t_i$  is an  $m$ -tuple of attribute-values. Let  $C$  be a set of  $k$  classes,  $C = \{C_1, \dots, C_k\}$ . The classification problem is to define a mapping function  $f$  that maps  $D$  into  $C$ . We write  $f: D \rightarrow C$  to signify that  $f$  assigns each member of  $D$  to a class in  $C$ .

In a typical case of classification, we are given a training database  $D$  wherein each record is already assigned to a class. The goal in such a case is to create a specific model that defines "class" as a function of some (possibly all) of the  $m$  attribute-values of the records, and then use this model to classify any new record whose class is not yet known. The most common classification methods are either partition based or distance based.

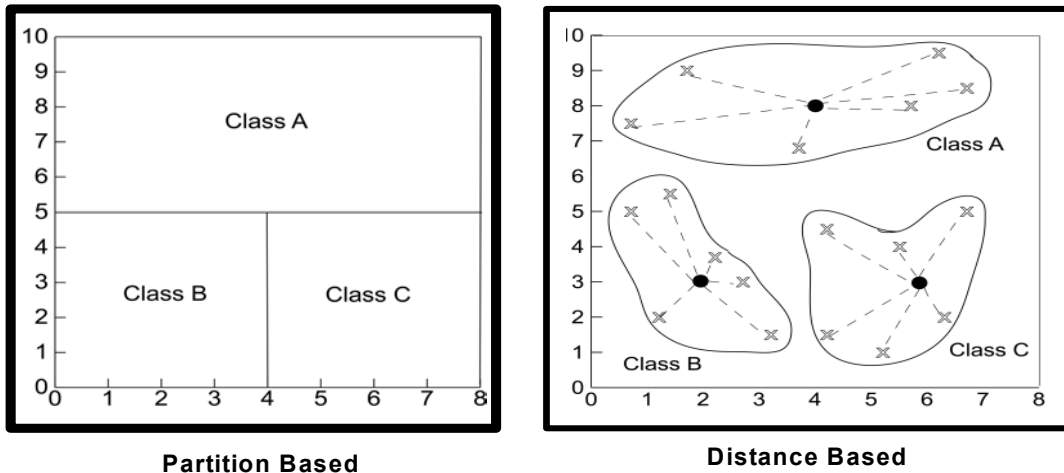


Figure 8. Classification methods.

Decision trees are partition based. They divide the search space into rectangular regions. A record (transaction) is placed into a class based on the region within which it falls. Decision tree approaches differ in how the tree is built. The most well-known algorithms are ID3, C4.5, CART, and CHAID.

Let us consider a database  $D = \{t_1, \dots, t_n\}$  where each record  $t_j$  is an  $m$ -tuple of attribute-values,  $t_j = \langle t_{j1}, \dots, t_{jm} \rangle$ . This database schema has the set of attributes  $\{A_1, A_2, \dots, A_m\}$  and predefined classes  $C = \{C_1, \dots, C_k\}$ . The normal procedure to build a decision tree precedes top down in recursive divide-and-conquer fashion. First an attribute is selected for the root node and a branch is created for each possible value of that attribute. Then  $D$  is split into subsets (one subset for each branch extending from the root node). Finally, the procedure is repeated recursively for each branch, using only records that reach the branch. If all records at a branch have the same class, then that branch is not expanded further.

Decision trees raise the following issues: choosing the most important attribute as the root node, ordering of splitting attributes, splits, tree structure, stopping criteria, training data, and pruning. The goal of classification is to reduce surprises and have entropy of zero. Information entropy measures the amount of randomness or uncertainty. The ID3 algorithm uses a criterion called *information gain* to compare potential splits. In this case, the important consideration is that the number of bits required to describe a particular outcome depends on the number of possible outcomes. For example, if there are eight equally probable classes, it takes  $\log_2(8)$ , or three bits, to identify a particular one. If, however, there are only four classes, then it takes only  $\log_2(4)$ , or two bits, to identify any one of them. So, a split that takes a node that has records from eight classes and splits it into nodes that average four classes is said to have an information gain of one bit.

The C4.5 algorithm is an improved version of the ID3 algorithm. These improvements include methods dealing with missing data, continuous data, pruning, and generating rules from trees, and utilizing the concept of gain ratio (Witten & Frank, 2000).

## **2.2 Association Rules**

The task of association rule is to find relationships among attributes. Association rules are similar to classification rules except that they can predict combinations of attributes. Before giving a formal definition of association rules, consider the sample data in Table 1. This fictitious table shows the conditions that are suitable for playing some unspecified game.

Table 1

*The Weather Data*

Outlook	Temperature	Humidity	Windy	Play
Sunny	Hot	High	False	No
Sunny	Hot	High	True	No
Overcast	Hot	High	False	Yes
Rainy	Mild	High	False	Yes
Rainy	Cool	Normal	False	Yes
Rainy	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Sunny	Mild	High	False	No
Sunny	Cool	High	False	Yes
Rainy	Mild	Normal	False	Yes
Sunny	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Rainy	Mild	High	True	No

Table 2 below is derived from the data in Table 1. The first column of Table 2 shows the individual items (attribute=value pairs) that occur in Table 1 together with their frequency of occurrence (in parentheses). These are the one-item sets. As the next step, generate the two-item sets by taking pairs of one-item sets. Suppose that we require our association rules to be supported by at

least two instances (records) in Table 1. Then any item sets that occur in fewer than two instances are discarded. These leaves 47 two-item sets, some of which are shown in the second column along with the number of times they appear. The next step is to generate the three-item sets, of which 39 have two or more instances in Table 1. Similarly, there are 6 four-item sets, and no five-item sets with the required support. For example, the first row of Table 2 shows that in Table 1 the one-item set {Outlook = Sunny} occurs in five instances, the two-item set {Outlook = Sunny, Temperature = Mild} occurs in two instances, the three-item set {Outlook = Sunny, Temperature = Hot, Humidity = High} occurs in two instances, and the four-item set {Outlook = Sunny, Temperature=Hot, Humidity=High, Play=No} occurs in two instances.

Table 2 *Item Sets with the Required Support in Table 1*

	<b>One-item sets</b>	<b>Two-item sets</b>	<b>Three-item sets</b>	<b>Four-item sets</b>
1	Outlook=Sunny (5)	Outlook = Sunny Temperature=Mild (2)	Outlook=Sunny Temperature=Hot Humidity = High (2)	Outlook=Sunny Temperature=Hot Humidity=High Play=No (2)
2	Outlook=Overcast (4)	Outlook = Sunny Temperature=Hot (2)	Outlook=Sunny Temperature=Hot Play=No (2)	Outlook=Sunny Humidity=High Windy=False Play=No (2)
3	Outlook=Rainy (5)	Outlook = Sunny Temperature=Normal (2)	Outlook=Sunny Humidity=Normal Play=Yes (2)	Outlook=Overcast Temperature=Hot Windy=False Play=Yes (2)
4	Temperature=Cool (4)	Outlook = Sunny Humidity=High (3)	Outlook=Sunny Humidity=High Play=No (3)	Outlook=Rainy Temperature=Mild Windy=False Play=Yes (2)
5	Temperature=Mild (6)	Outlook = Sunny Windy=True (2)	Outlook=Sunny Windy=False Play=No (2)	Outlook=Rainy Humidity=Normal Windy=False Play=Yes (2)

6	Temperature=Hot (4)	Outlook = Sunny Windy=False (3)	Outlook=Sunny Windy=False Play=No (2)	Temperature=Cool Humidity=Normal Windy=False Play=Yes (2)
7	Humidity=Normal (7)	Outlook = Sunny Play=Yes (2)	Outlook=Overcast Temperature=Hot Windy=False (2)	
8	Humidity=High (7)	Outlook = Overcast Temperature=Hot (2)	Outlook=Overcast Temperature=Hot Play=Yes (2)	
9	Windy=True (6)	Outlook = Overcast Temperature=Hot (2)	Outlook=Overcast Humidity=Normal Play=Yes (2)	
10	Windy=False (8)	Outlook = Overcast Humidity=Normal (2)	Outlook=Overcast Humidity=High Play=Yes (2)	
11	Play=Yes (9)	Outlook =Overcast Humidity=High (2)	Outlook=Overcast Windy=True Play=Yes (2)	
12	Play=No (5)	Outlook=Overcast Windy=True(2)	Outlook=Overcast Windy=False Play=Yes (2)	
...				
47		Windy=False Play=No (2)		

Witten and Frank (2000) and Agrawal and Srikant (1994) provided the following formal definition for association rules:  $X \Rightarrow Y | (c, s)$ . Here  $X \Rightarrow Y$  denotes an association rule,  $X$  and  $Y$  are sets of binary (true/false) attributes called *item sets*,  $X \cap Y = \emptyset$ ,  $c$  is the confidence of the rule, and  $s$  is the support of the rule. The *confidence* is a measure of the rule strength, that is, the percentage of records with all attributes in  $Y$  having value true within records with all attributes in  $X$  with value true. The *support* is a measure of the statistical significance of the rule, that is, the percentage of records that have all attributes in  $X \cup Y$  with value true.

Returning to the algorithm by which we derived Table 2 from Table 1, observe its main disadvantage: It generates a large number of candidate item sets, most of which are to be eliminated later due to insufficient support. To improve the performance, Agrawal and Srikant (1994) proposed the so-called Apriori algorithm. The key idea of the Apriori algorithm can be summarized as follows.

1. Test the support for item sets of size 1, called 1-itemsets, by scanning the database. Discard 1-itemsets having support smaller than  $s$ . (An itemset is called *large* if it has a support at least as large as the required support,  $s$ .)
2. Extend each large 1-itemset into 2-itemsets (by adding a large 1-itemset each time) to generate all candidate 2-itemsets. Test the support of the candidates and discard all 2-itemsets having a support smaller than  $s$ .
3. Repeat the above steps; at step  $k$ , the previously found  $(k - 1)$  large item sets are extended into  $k$ -itemsets and tested for minimum support.

The process is repeated until no new large item sets can be found.

### **2.3 Sequence Patterns**

Sequential pattern mining is the process of analyzing a collection of data to identify trends among them (Agrawal & Srikant, 1995). It is an important data mining task with many potential application domains (Agrawal & Srikant; Garofalakis, Rastogi, & Shim, 1999; Rastogi, Garofalakis, Seshadri, & Shim, 1999). Sequential pattern mining is closely related to association rule mining. Association rule mining is focused on finding intra-transaction patterns, whereas

sequential pattern mining is concerned with finding inter-transaction patterns. In sequential pattern mining, the input data are a set of sequences. Each sequence is a list of elements. A sequence containing  $k$  elements is called a  $k$ -sequence. The elements of a sequence can be either simple items or item sets. The support for a particular sequence is defined as the percentage of sequences that contain the particular sequence as a subsequence. The sequences with a user-specified minimum support are called frequent sequences. In a set of sequences, a sequence is maximal if it is not contained in any other sequences. The goal of sequential pattern mining is to find the maximal sequences among frequent sequences. Each such sequence represents a sequential pattern.

Based on the type of sequences, sequential pattern mining can be classified into two categories: spatial sequential pattern mining and temporal sequential pattern mining. Analyzing a collection of data over a period of time to identify trend is temporal sequential mining. Spatial sequential pattern mining is the process of finding the spatial arrangement of a collection of data. Mining that covers all data-sequences is called full sequential pattern mining, while partial sequential pattern mining focuses only on specific part of sequences.

The sequential pattern mining algorithms fall into two categories: count-all and count-some (Agrawal & Srikant, 1995). The count-all algorithms count all the frequent sequences, including non-maximal sequences. The non-maximal sequences must then be pruned out. The count-some algorithms can avoid counting non-maximal sequences by counting longer sequences first.

We examine a sequential pattern-mining algorithm called SPIRIT (Garofalakis et al., 1999). In this algorithm, the input consists of a set of sequences,  $D$ , together with a user-specified constraint,  $C$ . The output is a set of frequent sequences in the database  $D$  that satisfy constraint  $C$ .

SPIRIT-like algorithms, or those like the Apriori algorithm discussed above, work in passes, with each pass resulting in the discovery of longer patterns. There are two crucial differences between the SPIRIT and Apriori-like algorithms:

1. In the SPIRIT algorithm, the constraint  $C$  is relaxed by inducing a weaker constraint  $C'$ .  $C'$  is weaker than  $C$  if every sequence that satisfies  $C$  also satisfies  $C'$ .
2. The relaxed constraint  $C'$  is used in the candidate-generating and pruning phases of each pass.

Two different types of pruning techniques within each pass are applied: constraint-based pruning and support-based pruning. For constraint-based pruning, a relaxation  $C'$  of the user-specified constraint  $C$  is used to ensure that all candidate  $k$ -sequences in  $C_k$  satisfy  $C'$ . In support-based pruning, only the minimum support constraint for subsequences that satisfy  $C'$  is checked to ensure that all subsequences of a sequence  $s$  are in  $C_k$ .

## **2.4 Clustering**

A cluster is a collection of data objects that are similar to one another within the same cluster and are dissimilar to the objects in other clusters. Cluster analysis is the grouping of a set of data objects into clusters. Clustering is

unsupervised classification without predefined classes. Typical uses of clustering can be as a stand-alone tool to get information about data distribution or as a preprocessing step for other algorithms. Clustering is typically used as a stand-alone tool in pattern recognition, spatial data analysis, image processing, market research, and Weblog.

The following are some of the major clustering approaches:

1. Partitioning algorithms construct various partitions from the set of data and then evaluate them by some criterion.
2. Hierarchy algorithms create a hierarchical decomposition of the set of data using some criterion.
3. Density-based algorithms are based on connectivity and density functions.
4. Grid-based algorithms are based on a multiple-level granularity structure.
5. Model-based algorithms use a model hypothesized for each of the clusters and the idea is to find the best fit of that model to each other.

### ***2.5 Partitioning algorithms***

They partition a database  $D$  of  $n$  records into a set of  $k$  clusters. Given  $k$ , the goal is to find  $k$  clusters that optimize a chosen partitioning criterion. The following are two of the proposed criteria:

1. Global optimal: It may require an exhaustive enumeration of all partitions.
2. Heuristic rules: Two algorithms in common use are *k-means* and *k-medoids*. In *k-means* (MacQueen, 1967), each cluster is represented by the center of the cluster. In *k-medoids* or PAM (partition around medoids)

(Kaufman & Rousseeuw, 1987), each cluster is represented by one of the records in the cluster.

**2.5.1 K-means algorithm**

In the first step, we select  $k$  data points to be the seeds. MacQueen’s (1967) algorithm simply takes the first  $k$  rows. In the second step, we assign each record to the cluster whose centroid is nearest. In the third step, we calculate the centroids of each point in the cluster along each dimension. In the fourth step, once the new clusters have been found, each point is once again assigned to the cluster with the closet centroid. The process of assigning points to a cluster and then recalculating centroids continues until the cluster boundaries stop changing (see Figure 9).

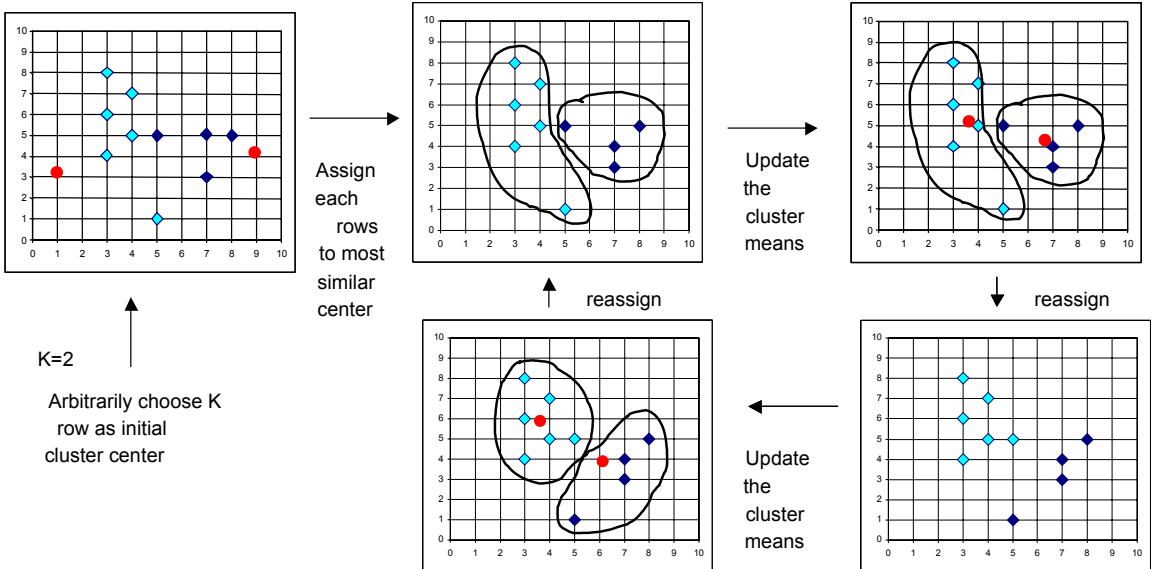


Figure 9. K-means clustering method.

**2.5.2 K-medoids algorithm**

Start with an initial set of  $k$  medoids ( $k$  representative records) and iteratively replaces one of the current medoids by one of the current non-medoids

if the replacement improves the total distance of the resulting clustering (see Figure 10). A somewhat more refined definition of the PAM algorithm may go as follows:

1. Select  $k$  representative records (medoids) arbitrarily
2. For each pair of medoid  $i$  and non-medoid record  $h$  :

Calculate the total swapping cost  $TC_{ih}$ . If  $TC_{ih} < 0$ , replace  $i$  by  $h$  and assign each non-medoid record to the cluster represented by the most similar medoid

3. Repeat Step 2 until there is no change.

PAM works effectively for small data sets, but does not scale well for large data set.

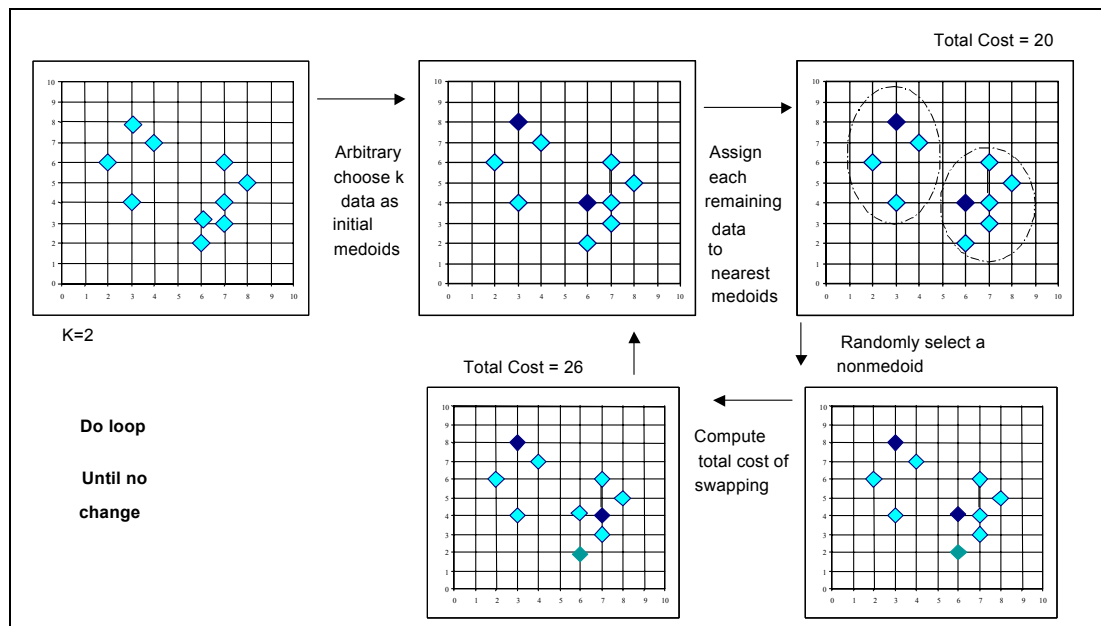


Figure 10. K-medoids algorithm.

## 2.6 Neural Nets

Neural nets have gone through two major development periods—the early 1960s and the mid-1980s. They were a key development in the field of machine learning. Artificial neural networks were inspired by biological findings relating to the behavior of the brain as a network of units called neurons. The human brain is estimated to have around 10 billion neurons; each connected on average to 10,000 other neurons. Each neuron receives signals through synapses that control the effects of the signal on the neuron. These synaptic connections are believed to play a key role in the behavior of the brain. The fundamental building block in an artificial neural network is the mathematical model of a neuron as shown in Figure 11. The three basic components of the (artificial) neuron are as follows (Bishop, 1995):

1. The synapses or connecting links that provide weights,  $w_j$ , to the input values,  $x_j$ , for  $j = 1, \dots, m$ ;
2. An adder that sums the weighted input values to compute the input to the activation function  $v = w_0 + \sum_{j=1}^m w_j x_j$ , where  $w_0$  is called the bias (not to be confused with statistical bias in prediction or estimation), a numerical value associated with the neuron. It is convenient to think of the bias as the weight for an input  $x_0$  whose value is always equal to one, so that  $v = \sum_{j=0}^m w_j x_j$ ;
3. An activation function  $g$  (also called a squashing function) that maps  $v$  to  $g(v)$ , the output value of the neuron. This function is a monotone function.

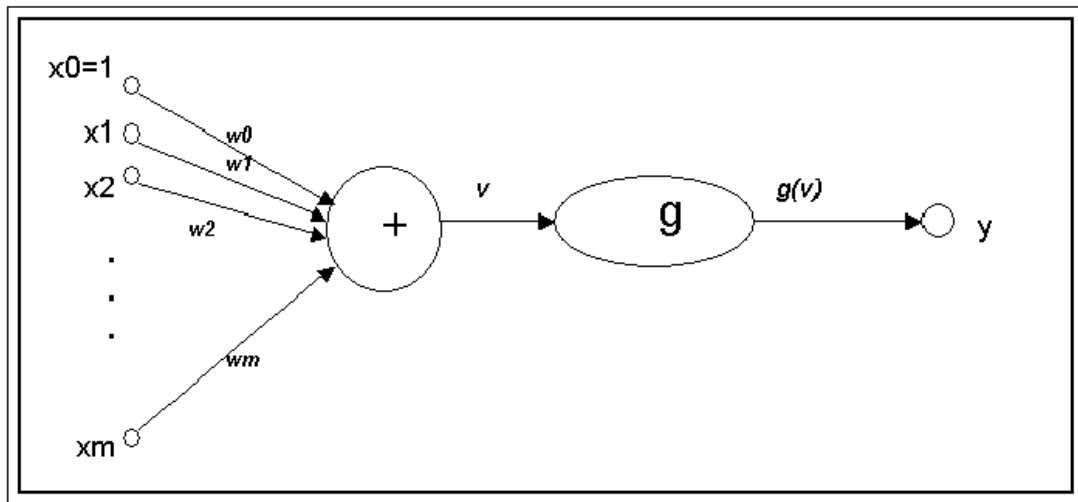
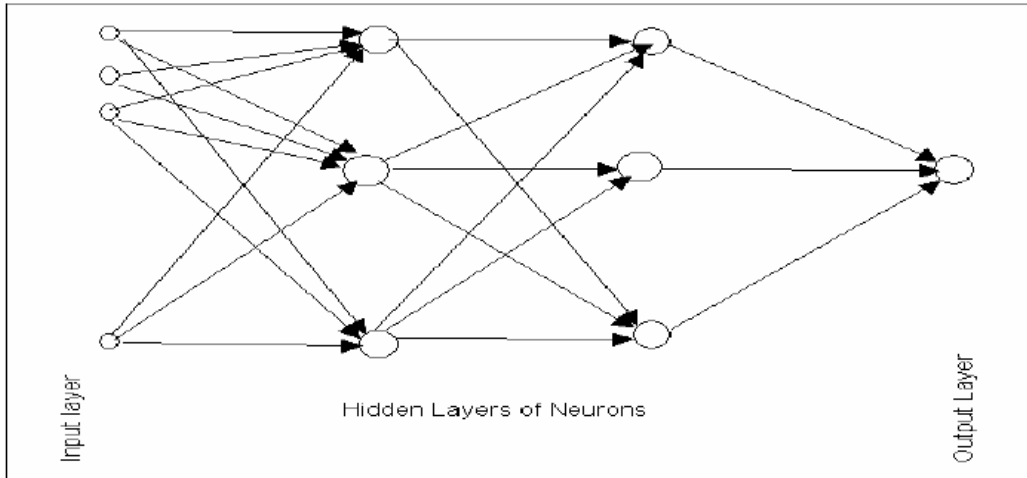


Figure 11. Neural nets.

While there are numerous different neural network architectures that have been studied by researchers, the most successful applications in data mining of neural networks have been multilayer feed forward networks. These are networks in which there is an input layer consisting of nodes that simply accept the input values and successive layers of nodes that are neurons as depicted in Figure 11. The outputs of neurons in a layer are inputs to neurons in the next layer. The last layer is called the output layer. Layers between the input and the output layers are known as hidden layers. Figure 12 is a diagram for this architecture.



*Figure 12.* Feed forward neural nets.

In a supervised setting where a neural net is used to predict a numerical quantity, there is one neuron in the output layer and its output is the prediction. When the network is used for classification, the output layer typically has as many nodes as the number of classes and the output layer node with the largest output value gives the network's estimate for the class of a given input. In the special case of two classes it is common to have just one node in the output layer, the classification between the two classes being made by applying a cutoff to the output value at the node.

### ***2.7 Traditional Statistical Methods***

By strict definition, statistics or statistical techniques are not data mining. They were being used long before the term data mining was coined to apply to business applications. However, statistical techniques are driven by the data and are used to discover patterns and build predictive models. The first step in understanding statistics is to understand how the data are condensed into more

meaningful or more usable forms. One way of doing this is with the use of histograms. Consider the data in Table 3. For this example, the frequency histogram of the "Eyes" attribute is depicted Figure 13.

Table 3  
*Customer Record*

ID	Name	Prediction	Age	Balance	Income	Eyes	Gender
1	Arman	No	41	\$0	Medium	Brown	M
2	Alice	No	40	\$1,800	Medium	Green	F
3	Betty	No	47	\$16,543	High	Brown	F
4	Bob	Yes	32	\$45	Medium	Green	M
5	Carla	Yes	21	\$2,300	High	Blue	F
6	Carl	No	27	\$5,400	High	Brown	M
7	Donna	Yes	50	\$165	Low	Blue	F
8	Don	Yes	46	\$0	High	Blue	M
9	Edna	Yes	27	\$500	Low	Blue	F
10	Ed	No	68	\$1,200	Low	Blue	M

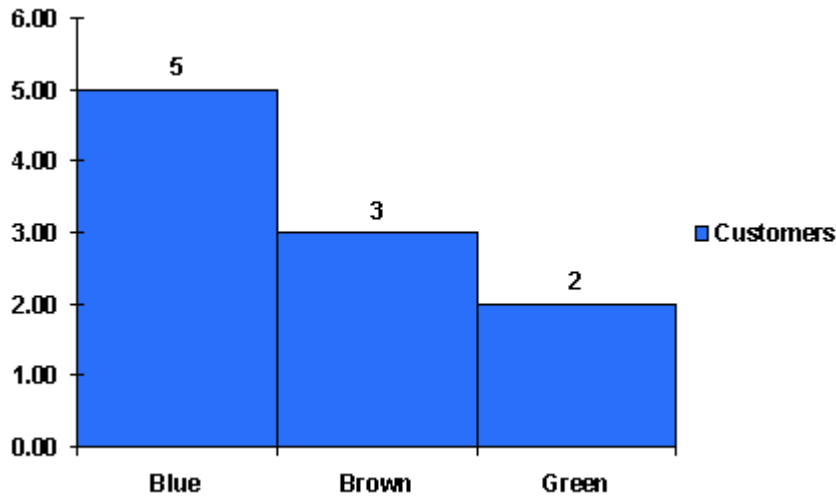
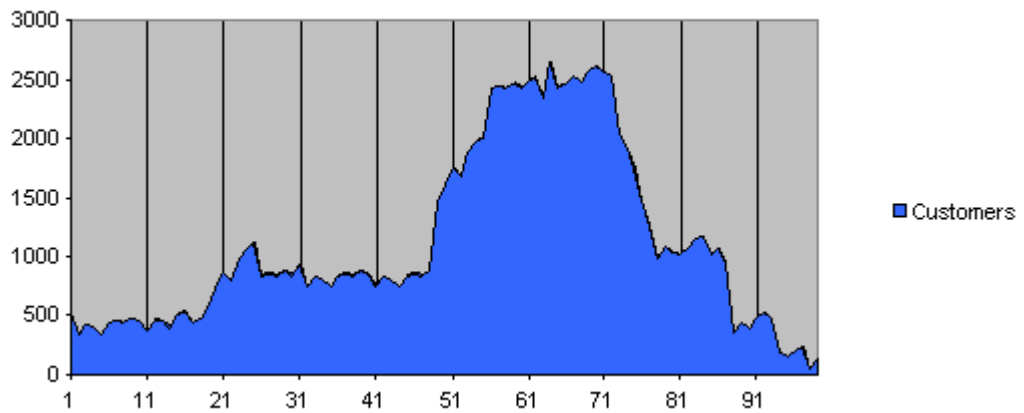


Figure 13. Frequency histogram for customers with various eye colors.

The "Eyes" attribute will have only a few different values no matter if there are 100 customer records in the database or 100 million. Other attributes, however, may have many more distinct values and can result in more complex histograms. Consider, for instance, the histogram in Figure 14 that depicts the distribution of tens of thousands of customers by age.



*Figure 14.* Histogram for customers of different ages grouped by year.

By looking at this histogram it is possible to give reasonable estimates for statistics such as the maximum age, the minimum age, and average age of the population. These statistics are called summary statistics. Some of the most frequently used summary statistics for an attribute include the following:

1. Max—the maximum value of the attribute (in the database)
2. Min—the minimum value of the attribute
3. Mean—the average value of the attribute
4. Median—the value of the attribute that divides the database as nearly as possible into two databases of equal numbers of records

5. Mode—the most common value of the attribute
6. Variance—the measure of how spreads out the attribute-values are around the average value of the attribute.

In statistics prediction is usually synonymous with regression of some form. There are a variety of different types of regression in statistics but the basic idea is that a model is created that maps values from predictors to values of a predicted variable (attribute) in such a way that the lowest error occurs in making a prediction. The simplest form of regression is simple linear regression that contains just one predictor variable and one predicted variable. The relationship between the two variables can be plotted in two-dimensional space so that predicted values are along the *Y*-axis and predictor values along the *X*-axis. The simple linear regression model then could be viewed as the line that minimizes the error rate between the observed values of the predicted variable and the corresponding points on the line (the predictions from the model). Graphically this would look as it does in Figure 15. Of the many possible lines that could be drawn through the data, the one that minimizes the sum of the vertical distances between the line and the data points is the one that is chosen for the predictive model.

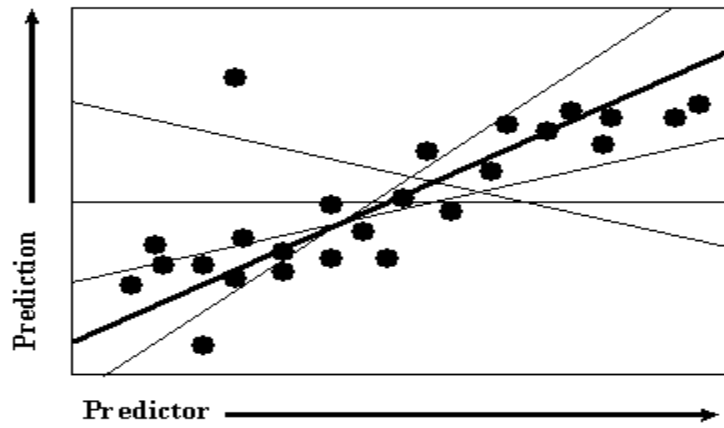


Figure 15. Linear regressions.

## CHAPTER 3

### PARALLEL DATA MINING ALGORITHMS

In the context of an enterprise data warehouse, we need to process very large databases. To make things worse, the volume of data is growing daily. Traditional data mining solutions are not scalable to the ever increasing volume of data, and eventually lead to unacceptable response times. This points to the need for parallel formulations of traditional data mining algorithms.

#### ***3.1 Parallel Algorithms for Classification***

##### ***3.1.1 SPRINT***

SPRINT (Shafer, Agrawal, & Mehta, 1996) was one of the earliest scalable decision tree classifiers based on data parallelism and distributed memory. The algorithm assumes a shared-nothing parallel environment where each of  $N$  processors has private memory and disks. The processors are connected by a communication network and can communicate only by passing messages. SPRINT achieves uniform data placement and workload balancing by distributing the attribute lists evenly over  $N$  processors of a shared-nothing machine. This allows each processor to work on only  $1/N$  of the total data.

The partitioning is achieved by first distributing the training-set examples equally among all the processors. Each processor then generates its own list of attributes from the training-set in parallel. Lists for attributes are therefore evenly partitioned and require no further processing. However, continuous attribute lists must now be sorted using a parallel sorting algorithm. The result of this sorting

operation is that each processor gets a fairly equal-sized sorted section of each attribute list. Tables 4 and 5 illustrate the SPRINT algorithm.

Table 4  
*SPRINT Data Set Example*

ID	Refund	Marital status	Taxable income	Cheat
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

Table 5  
*SPRINT Applied to Data Set*

ID	Refund	Cheat	ID	Marital status	Cheat	ID	Taxable income	Cheat
1	Yes	No	1	Single	No	1	125K	No
2	No	No	2	Married	No	2	100K	No
3	No	No	3	Single	No	3	70K	No
4	Yes	No	4	Married	No	4	120K	No
5	No	Yes	5	Divorced	Yes	5	95K	Yes
6	No	No	6	Married	No	6	60K	No
7	Yes	No	7	Divorced	No	7	220K	No
8	No	Yes	8	Single	Yes	8	85K	Yes
9	No	No	9	Married	No	9	75K	No
10	No	Yes	10	Single	Yes	10	90K	Yes

The SPRINT algorithm has the following characteristics:

1. The arrays of the continuous attributes are presorted. The sorted order is maintained during each split.
2. The classification tree is grown in a breadth-first fashion.
3. Class information is included in each attribute list.
4. Attribute lists are physically split among nodes.
5. Split determining phase is just a linear scan of lists at each node.
6. A hashing scheme is used in the splitting phase.
  - IDs of the splitting attribute are hashed with the tree node as the key.
  - Remaining attribute arrays are split by querying this hash structure.

SPRINT has some shortcomings. The size of hash table is  $O(N)$  for top levels of the tree. If the hash table does not fit in memory (mostly true for large data sets), then it is built in parts so that each part fits. This causes multiple expensive I/O passes over the entire data set. Storing the entire hash table on one processor makes the SPRINT algorithm un-scalable.

### ***3.1.2 Synchronous tree construction approach***

In this approach, all processors construct a decision tree synchronously by sending and receiving class distribution information of local data (see Figure 16). Major steps for the approach are shown below (Srivastava et al. 1999):

Select a node to expand according to a decision tree expansion strategy (e.g., depth-first or breadth-first) and call that node the current node. At the beginning, the root node is selected as the current node.

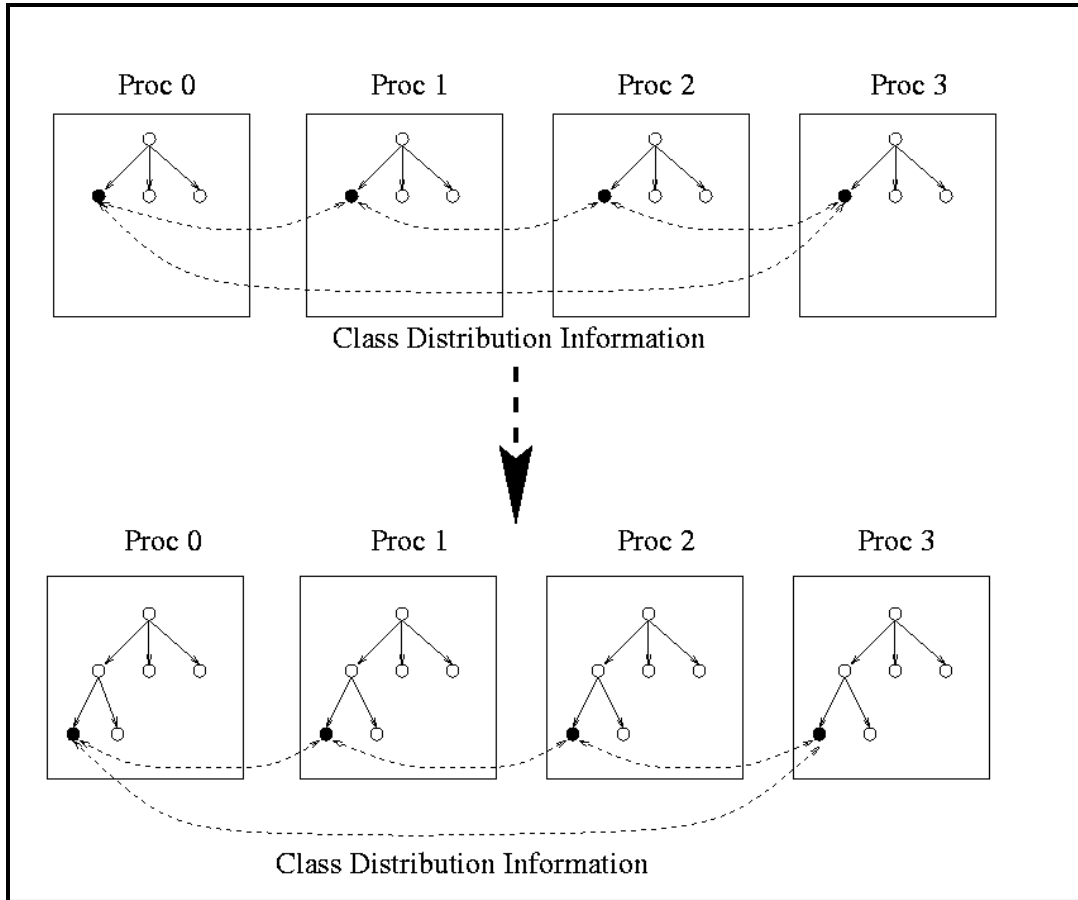


Figure 16. Synchronous tree construction approach.

1. For each data attribute, collect class distribution information of the local data at the current node.
2. Exchange the local class distribution information using global reduction among processors.
3. Simultaneously compute the entropy gains of each attribute at each processor and select the best attribute for child node expansion.

4. Depending on the branching factor of the tree desired, create child nodes for the same number of partitions of attribute values and split training cases accordingly.
5. Repeat above steps until no node is available for expansion.

### ***3.1.3 Partitioned tree construction approach***

In this approach, different processors work on different parts of the classification tree. Figure 17 shows an example of four processors. First, all four processors cooperate to expand the root node. Next, the set of four processors is partitioned in three parts. The leftmost child is assigned to processor 0 and 1, while the other nodes are assigned to processors 2 and 3. Now these sets of processors proceed independently to expand their assigned nodes. Specifically, processors 2 and 3 proceed to expand their part of the tree using the serial algorithm. The group having processors 0 and 1 splits the leftmost child node into three nodes. These three new nodes are partitioned in two parts; the leftmost node is assigned to processor 0, while the other two are assigned to processor 1. Now, processors 0 and 1 also work independently on their respective subtrees.

The advantage of this approach is that once a processor becomes solely responsible for a node, it can develop a subtree of the decision tree independently without any communication overhead. There are a number of disadvantages of this approach. The first disadvantage is that it requires data movement after each node expansion until one process becomes responsible for an entire subtree. The communication cost is particularly expensive in the

expansion of the upper part of the decision tree. The second disadvantage is due to load balancing.

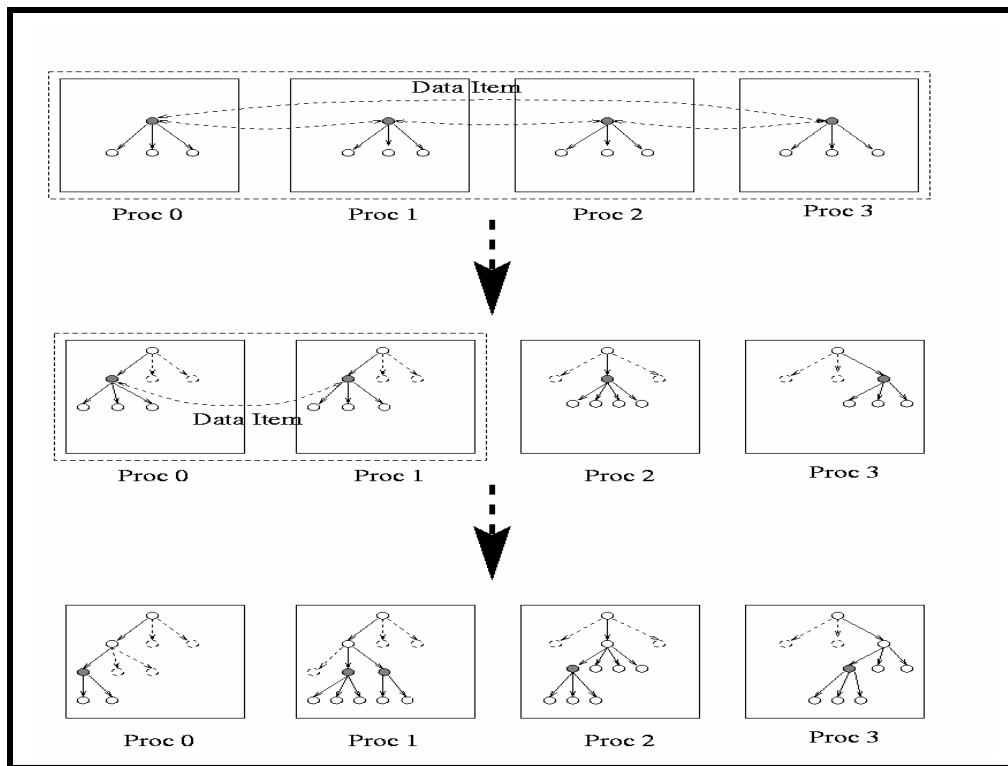


Figure 17. Partitioned tree construction approach.

### 3.1.4 Hybrid Algorithms

The synchronous tree construction approach incurs high communication overhead as the tree depth gets larger. The partition tree construction approach incurs the 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 become high, the processor and the tree are partitioned at the current depth of the tree (Karypis & Kumar, 1994).

Figure 18 shows one example of hybrid parallel formulation. At depth 3 of the tree, no partitioning has been done and all processors are working

cooperatively on each node. At depth 4, partitioning is triggered, and the nodes and processors are partitioned into two partitions.

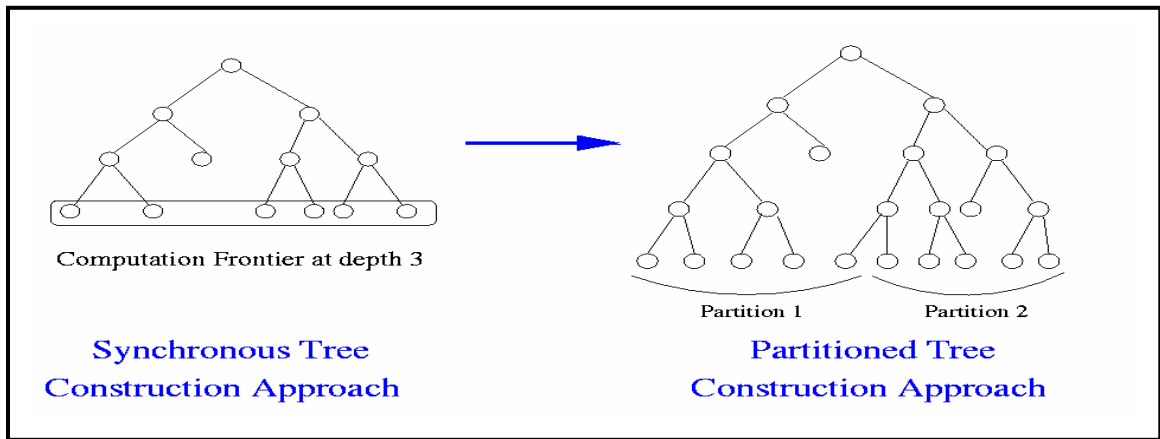


Figure 18. Hybrid approach.

If a group of processors in a partition becomes idle, then this partition joins up with another partition that has work and has the same number of processors. As shown in Figure 19, this can be done by simply giving half of the training cases located at each processor in the donor partition to a processor in the receiving partition.

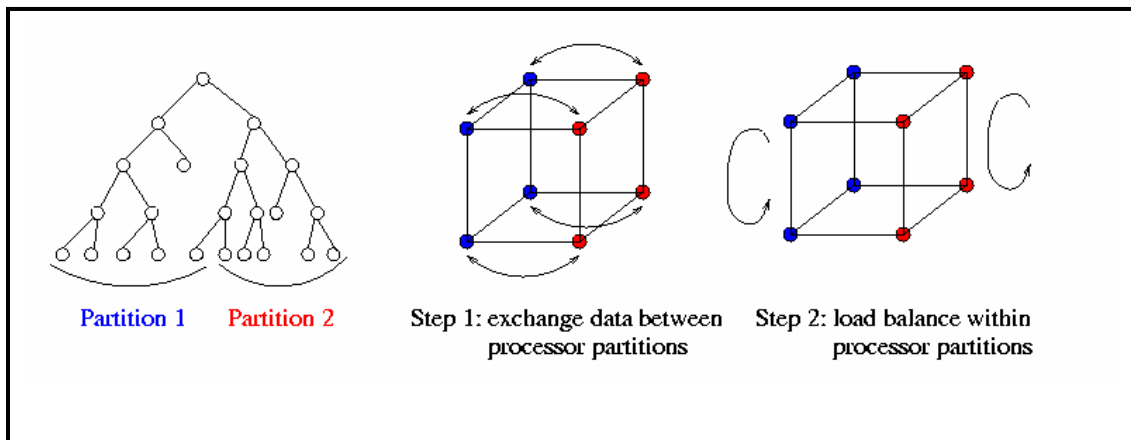


Figure 19. Load balancing.

### 3.1.5 ScalParC design approach

ScalParC stands for scalable parallel classifier, which was defined by Joshi, Karypis, and Kumar in 1998. ScalParC sorts the continuous attributes only once in the beginning. It uses attribute lists similar to SPRINT. The key difference is that it employs a distributed hash table to implement the splitting phase. The communication structure used to construct and access this hash table introduces a new parallel hashing paradigm. The parallel sparse matrix-vector multiplication algorithms motivate the communication structure used to construct and access this hash table (see Figure 20). It was shown in Joshi et al. that with the proper implementation of the parallel hashing, ScalParC is scalable in runtime as well as memory requirements.

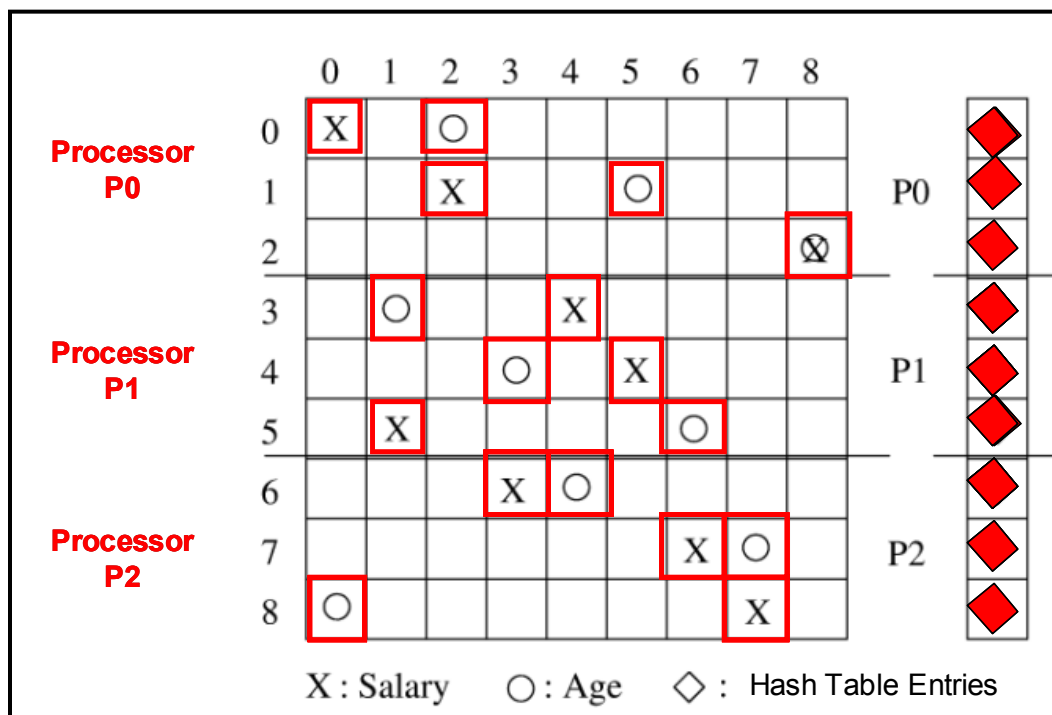


Figure 20. Communication structure of parallel sparse matrix-vector algorithms.

### **3.2 Parallel Algorithms for Association Rules**

Typically, association rules are based on very large data sets with high dimensionality. Therefore the mining of association rules is a good candidate for solution by parallel computation. The process of association rule discovery consists of two parts: the first part is to find frequent item sets and the second part is to generate rules from the frequent item sets. The first part is more expensive in terms of computing resources utilization and CPU power. Therefore, most parallel algorithms focus on frequent item set discovery.

The computation of frequent item sets can be envisioned in two parts: the effort invested in creating the candidates and the effort spent in counting them. There are several ways to distribute this work among processors. The issue is how to distribute the candidates among processors such that their counting and creation is effectively parallelized. There are two possibilities. One is to replicate the candidates on all processors and the other is to avoid replication. Various algorithms will be reviewed based on these possibilities.

#### **3.2.1 Count distribution**

The count distribution algorithm, proposed by Agrawal and Shafer (1996), achieved parallelism by partitioning data. Each of  $N$  nodes gets  $1/N$ th of the database and performs an Apriori-like algorithm on the subset. At the end of each iteration is a communication phase in which the frequency of attribute occurrence in the various data partitions is exchanged between all nodes. Thus, this algorithm trades off I/O and duplication for minimal communication and good

load balance: each node must scan its database partition multiple times and maintain a full copy of the data structures used, but only requires a small amount of per-iteration communication and has a good distribution of work.

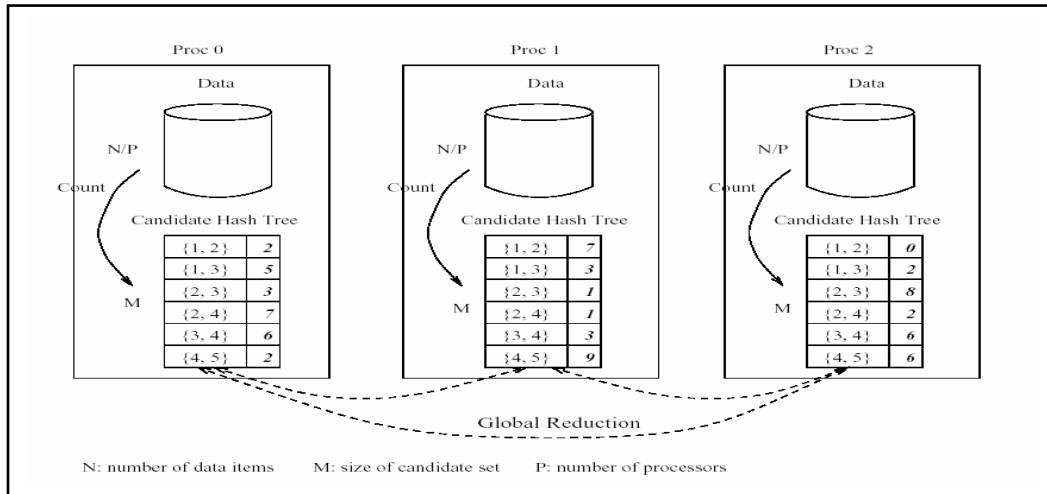


Figure 21. Count distribution algorithm.

Figure 21 illustrates the count distribution (CD) algorithm. Since each processor needs to build a hash tree for all candidates, these hash trees are identical at each processor. Therefore each processor in the CD algorithm executes the serial Apriori algorithm on the locally stored data.

### 3.2.2 Data Distribution

This algorithm (Agrawal and Shafer, 1996) is designed to minimize computation redundancy and maximize use of the memory capacity of each node. It works by partitioning the current maximal-frequency item set data candidates among the processors. This partitioning is done in a round robin fashion. Thus, each processor examines a disjoint set of possibilities. However, each processor must scan the entire database to examine its candidates.

Therefore, this algorithm takes on a large increase in communication (to fetch the database partitions stored on other processors) in exchange for better use of machine resources and for avoiding duplicated work. Figure 22 shows the high-level operations of the algorithm. Note that each processor has a different set of candidates in the candidate hash tree.

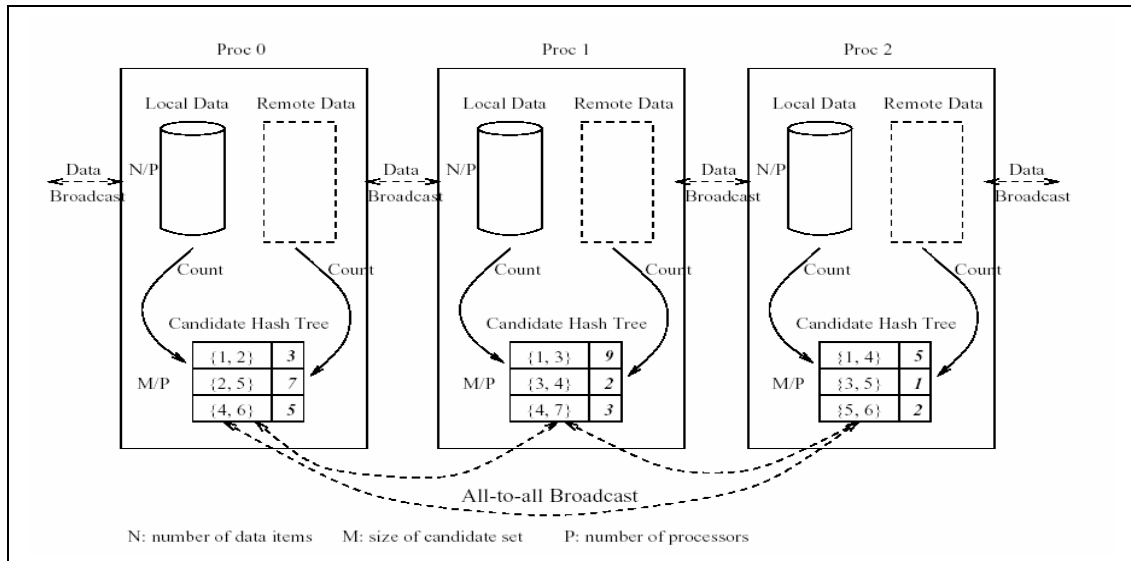


Figure 22. Data distribution algorithm.

### 3.2.3 Intelligent data distribution

In the intelligent data distribution algorithm, the locally stored portions of the database are sent to all the other processors by using a ring-based all-to-all broadcast. Compared to the data distribution algorithm where all the processors send data to all other processors, the intelligent data distribution algorithm performs only point-to-point communication between neighbors, thus eliminating any communication contention. In order to eliminate the redundant work due to the partitioning of the candidate item sets, it finds a fast way to check whether

given data can potentially contain any of the candidates stored at each processor.

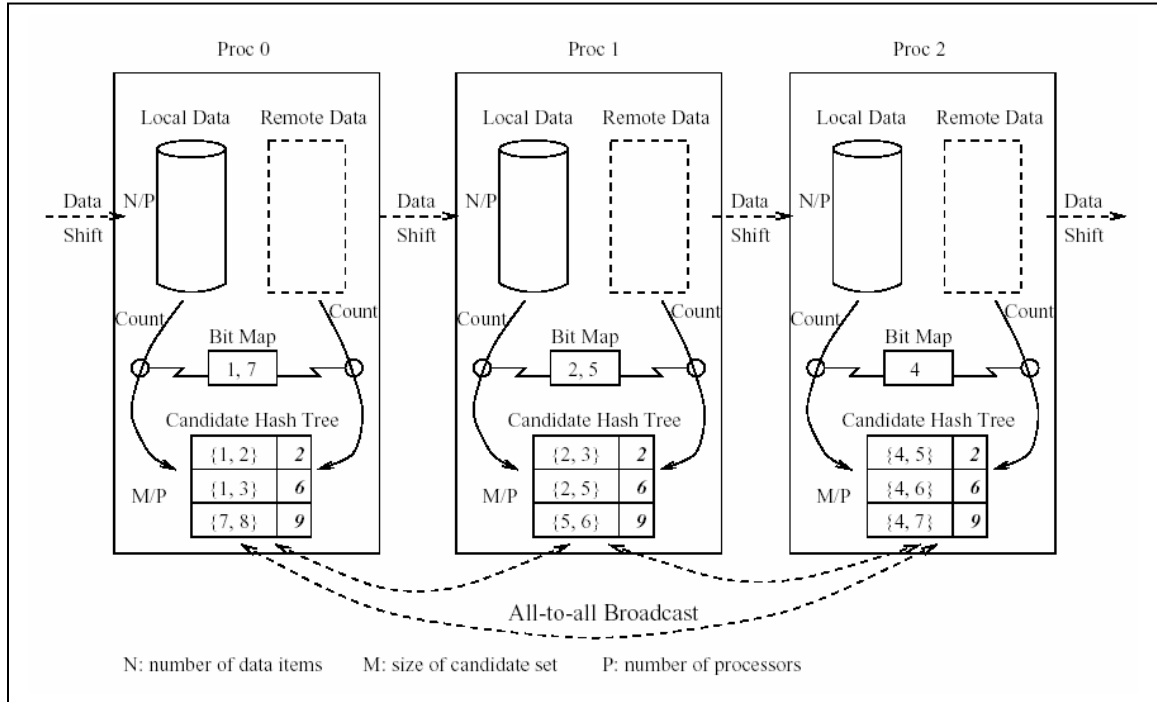


Figure 23. Intelligent data distribution algorithm.

In Figure 23, processor 0 has all the candidates starting with attributes 1 and 7. Processor 1 has all the candidates starting with 2 and 5 and so on. Each processor keeps the first attributes of its candidates in a bit-map. In the Apriori algorithm, at the root level of the hash tree, every attribute in a transaction is hashed and checked against the hash tree. However, in intelligent data distribution, at the root level, each processor filters every attribute of the transaction by checking against the bit-map to see if the processor contains candidates starting with that attribute of the transaction. If the processor does not contain the candidates starting with that attribute, the processing steps involved with that attribute as the first attribute in the candidate can be skipped. This

reduces the amount of transaction data that have to go through the hash tree, thus reducing the computation.

### **3.2.4 Hybrid distribution**

The intelligent data distribution algorithm exploits the total system memory by partitioning the candidate set among all processors. The average number of candidates assigned to each processor is  $M/P$ , where  $M$  is the total number of candidates. As more processors are used, the number of candidates assigned to each processor decreases. This has two implications. First, with fewer candidates per processor, it is much more difficult to balance the work. Second, the smaller number of candidates gives a smaller hash tree and less computation work per transaction. Eventually the amount of computation may become less than the communication involved. This would be more evident in the later passes of the algorithm as the hash tree size further decreases dramatically. This reduces overall efficiency of the parallel algorithm. This will be an even more serious problem in a system that cannot perform asynchronous communication.

The hybrid distribution algorithm addresses the above problem by combining the count distribution and the intelligent data distribution algorithms in the following way. Consider a  $P$ -processor system in which the processors are split into  $G$  equal size groups, each containing  $P/G$  processors. In the hybrid distribution algorithm, we execute the count distribution algorithm as if there were only  $P/G$  processors. That is, we partition 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 data to each one of these groups of processors. Within

each group, these counts are computed using the intelligent data distribution algorithm. That is, the transactions and the candidate set  $C_k$  are partitioned among the processors of each group, so that each processor gets roughly  $|C_k|/G$  candidate item sets and  $N/P$  data. Now, each group of processors computes the counts using the intelligent data distribution algorithm, and the overall counts are computed by performing a reduction operation among the  $P/G$  groups of processors.

The hybrid distribution algorithm can be better visualized (see Figure 24) if we think of the processors as being arranged in a two-dimensional grid of  $G$  rows and  $P/G$  columns. The transactions are partitioned equally among the  $P$  processors. The candidate set  $C_k$  is partitioned among the processors of each column of this grid. This partitioning of  $C_k$  is identical for each column of processors, that is, the processors along each row of the grid get the same subset of  $C_k$ .

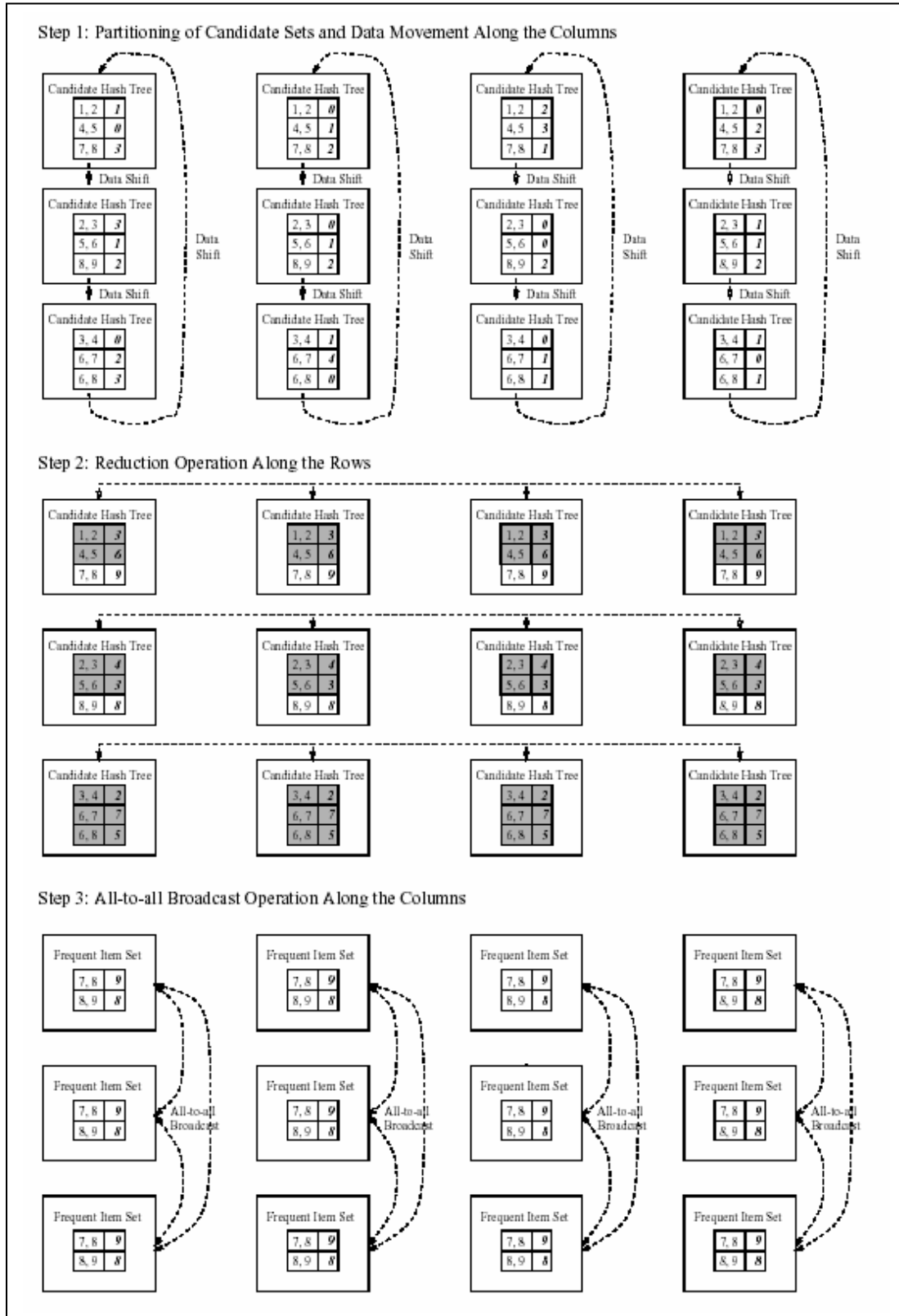


Figure 24. Hybrid distribution algorithm in 3X4 processor mesh ( $G=3, P=12$ ).

### **3.4 Parallel Algorithms for Clustering**

#### **3.4.1 Parallel $k$ -means**

Parallel  $k$ -means were proposed by Dhillon and Modha (1993), Xu and Zhang (2002), and Stoffle and Belkonience (1999) for very large databases. The most computation-intensive task in the parallel  $k$ -means algorithm is the calculation of distances. If  $d$  is the number of records and  $k$  is the number of clusters, then  $dk$  distances must be computed in each pass. The main idea for a parallel formulation of  $k$ -means is that of splitting the data set evenly among processors to speed up the computation. It is important to note, however, that the communication cost between the processors is significant. So, for an efficient parallel algorithm, it is necessary to minimize the communication between nodes. The centroid lists consist of  $k$  vectors. Upon beginning the clustering, each processor will receive the entire centroid list, but only the processor with  $id$  as the root will compute the initial centroids and then broadcast this selected  $k$  initial centroids to all other processors. Each processor takes care of only part of the data (its own subset of the vectors (records)), thus each processor will compute the distances for only its own subset of the vectors to each of the centroids. The computed distances then get stored locally on each processor. A series of assignments are generated mapping vectors to clusters. Each processor then gathers the sum of all vectors allocated to a given cluster and computes the mean of the vectors assigned to a particular cluster. This is repeated for every cluster and a new set of centroids is available on every processor, and the vectors can be reassigned with respect to the newly calculated centroids. To

compute the quality of the given partition, each processor can compute the mean squared error for the portion of the data set over which it is operating. As these values are calculated, the processor will sum its local mean squared error values. A simple reduction of these values among all processes will determine the overall quality of the clustering.

#### **3.4.2 Parallel $k$ -nearest neighbors**

$K$ -nearest neighbor classifier is based on learning by analogy. The training samples are described in an  $n$ -dimensional numeric space. Given an unknown sample, the  $k$ -nearest neighbor classifier searches the pattern space for  $k$  training samples that are closest, using the Euclidean distance, to the unknown sample. Again, this technique can be parallelized as follows. The training samples are distributed among the nodes. Given an unknown sample, each processor processes the training samples it owns to calculate the  $k$ -nearest neighbors locally. After this local phase, a global reduction computes the overall  $k$ -nearest neighbors from the  $k$ -nearest neighbors on each processor.

## **CHAPTER 4**

### **ARCHITECTURES AND SYSTEMS FOR HIGH-PERFORMANCE DATA MINING**

The main architectural problem in high-performance data mining is how to perform efficient data mining in very large databases. A natural solution is parallelism. Many data warehouses are already implemented on parallel database servers. There are seven different approaches for speeding up data mining in large databases. These seven approaches are divided into two categories: data-oriented and algorithm-oriented approaches.

#### ***4.1 Data-Oriented Approaches***

1. Sampling: A sample of data is used to reduce the number of rows.
2. Attribute selection: The number of attributes chosen for data mining is reduced.
3. Discretization: The number of values of attributes is reduced, which in turn reduces rule space size.

All the above approaches modify the data to be mined, so they lead to the discovery of knowledge different from the one that could be discovered in the entire original data set. In particular, sampling often reduces the accuracy of the discovered knowledge. Attribute selection and discretization can either increase or decrease the accuracy of the discovered knowledge.

## **4.2 Algorithm-Oriented Approaches**

1. Design faster algorithms via restriction of the search space: this usually reduces classification accuracy.
2. Design faster algorithms via code optimization: this could result in loss of generality.
3. Distributed data mining:
  - (a) Data are partitioned, and each data subset is sent to a distinct processor.
  - (b) Each processor applies a data mining algorithm to its local data.
  - (c) Partial results are combined.

Although all data are used, each processor has access to its local data only, so usually the discovered knowledge is different from that discovered by a sequential algorithm.

4. Parallel data mining: In principle, parallel data mining discovers the same knowledge as its sequential counterpart (i.e., it gains in speed, keeping the discovered knowledge's quality constant).

Out of the above approaches, in general only parallel data mining and code optimization have the property of discovering the same knowledge as the sequential data mining algorithms. However, parallel data mining is more generic and usually offers a better cost/benefit ratio than code optimization.

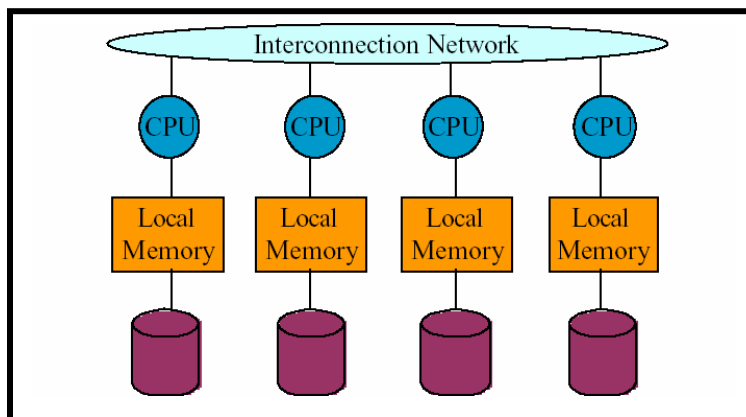
Note that some of the above approaches are not mutually exclusive, so two or more of these approaches can be used at the same time. The above approaches are surveyed in more detail in Freitas and Lavington (1998).

### **4.3 Overview: Parallel Systems**

The parallel architectures fall into the following categories: distributed memory, shared disk, shared memory, and a hybrid cluster of SMPs.

#### **4.3.1 Distributed-memory machines**

In this architecture, each processor has local memory and disk. The communication between processors happens via message passing. It is harder to program since one has to provide explicit data distribution. The main performance goal is to minimize communication between processors (Figure 26).



*Figure 25.* Distributed memory architecture (shared-nothing).

#### **4.3.2 Shared-memory machines**

This architecture has shared global address space and disk. The communication happens via shared memory variables. It is easier to program and the main goal is to maximize locality and minimize false sharing. The current trend is to have clusters of SMPs and grids (Figures 27–29).

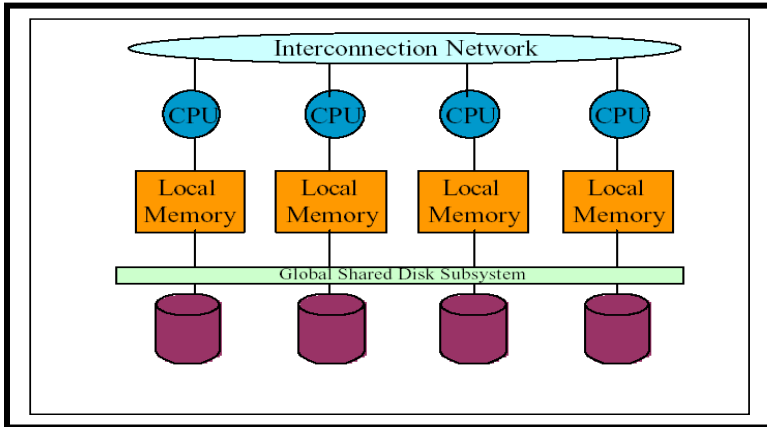


Figure 26. Shared disk architecture.

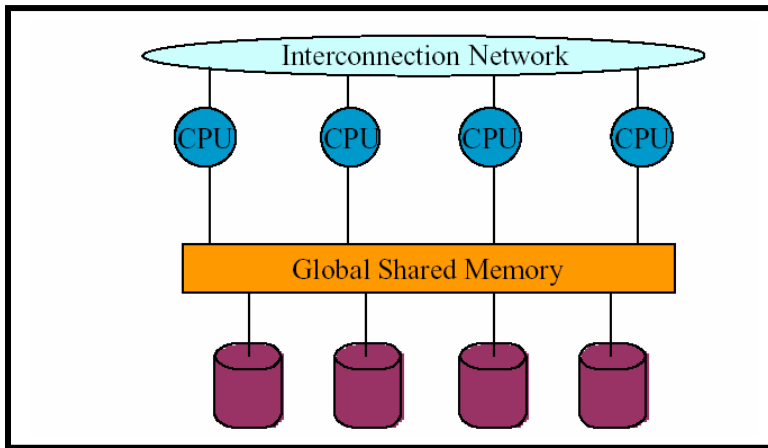


Figure 27. Shared memory architecture (shared-everything).

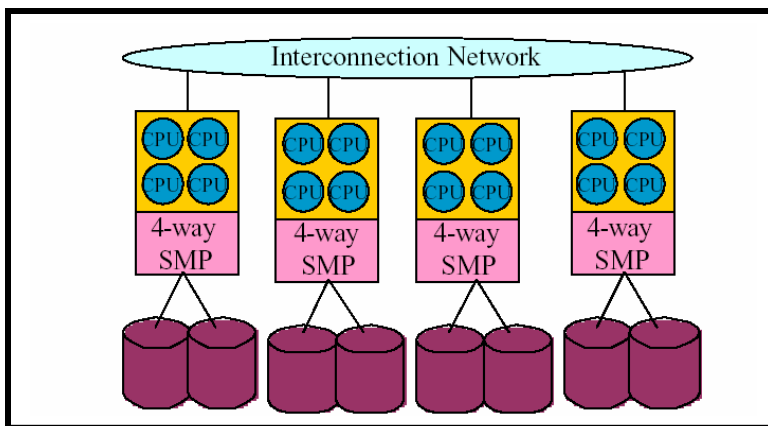


Figure 28. Cluster of SMPs.

#### 4.4 Task versus Data Parallelism

In data parallelism, the data are partitioned among  $P$  processors. Each processor performs the same work on local partitions. In task parallelism, different processors may perform different computations. The data may be selectively replicated or partitioned.

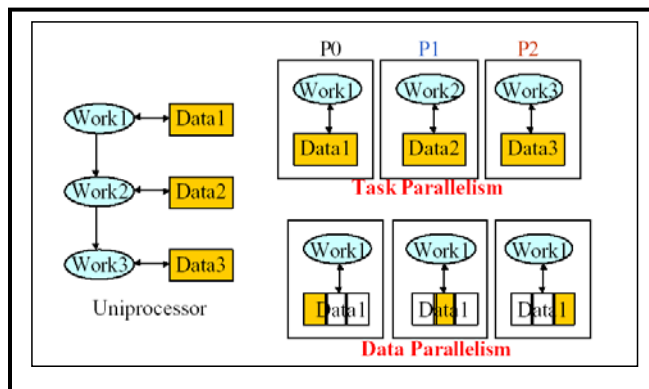


Figure 29. Task parallelism versus data parallelism.

In data parallelism, we have SIMD (single instruction, multiple data). Processes communicate to exchange partial results. In hybrid parallelism, we use a combination of basic strategies to improve both performance and accuracy of results. The data and task parallel strategies are not necessarily alternatives for speeding up data mining algorithms. Hybrid strategies can be used to improve performance, scalability, and accuracy of results.

#### 4.5 High-Performance Data Mining

Exploring useful information from large databases will require efficient parallel algorithms running on high-performance computing systems with powerful parallel input/output capabilities. Without such systems and techniques,

invaluable data and information will remain undiscovered. We also need to quantify the time it takes to retrieve pertinent information for time-critical applications. High-performance data mining thus refers to developing algorithms for data mining that lend themselves to parallelism and can map efficiently onto parallel computers. Such algorithms should consider the underlying architecture—such as distributed shared memory or a dedicated cluster of workstations—and the standard issues and challenges in parallel processing such as data locality, load balancing, inter-processor communication, and parallel input/output. Parallel I/O becomes very important in this case, because massive data analysis will require substantial data access to secondary storage and tape libraries. Issues of data organization, data pre-fetching, and latency hiding by overlapping the mining tasks with the I/O will soon become even more critical, and many new research challenges and opportunities will materialize. We can use three basic data partition strategies to improve performance, scalability, and accuracy of parallelizing data mining algorithms. These are the sequential partitioning, cover-based partitioning, and range-based query. In the sequential partitioning, we use separate data partitions without overlapping. In cover-based partitioning, some data can be replicated on different partitions. Finally, in range-based query, data partitions are used on some queries that select the data according to attribute values.

#### ***4.6 Main Data Mining Tasks and Techniques***

Table 6 shows a summary of main data mining tasks and techniques.

Table 6  
*Data Mining Tasks and Techniques*

Tasks	Techniques
Classification	Induction, neural networks, genetic algorithms
Association	Apriori, statistics, genetic algorithms
Clustering	Neural networks, induction, statistics, linear algebra
Regression	Induction, neural networks, statistics
Summarization	Induction, statistics
Episode discovery	Induction, neural networks, genetic algorithms

#### **4.6.1 Parallelism in Data Mining Techniques**

##### 1. Parallel Decision Trees

- a. The tree construction is occurring in parallel and processes deal with subtrees.
- b. The classification task is by assigning new rows to predefined classes. Tree leaves represent classes and tree nodes represent attribute values.
- c. In the task-parallel approach, one process is associated to each subtree. The search occurs in parallel in each subtree. The degree of parallelism  $P$  is equal to the number of active processes at a given time

##### 2. Discovery of Association Rules in Parallel

- a. An association discovery is defined as the automatic discovery of associations in a data set.

- b. In the SPMD approach, the data set is partitioned among the processors but candidate item sets are replicated on each processor. All the  $P$  processes count the partial support in parallel of the global item sets on its local partition of the data set. At the end of this phase the global support is obtained by collecting all local supports. The replication of the candidate item sets minimizes communication, but does not use memory efficiently. Due to low communication overhead, scalability is good.
  - c. In the task-parallel approach, both data set and candidate item sets are partitioned on each processor. Each process counts the global support of its candidate item set on the entire data set. After scanning its local data set partition, a process must scan all remote partitions for each iteration. The partitioning of data sets and candidate item sets minimizes the use of memory but requires high communication overhead. Due to communication overhead this approach is not scalable as the previous one.
3. Parallel Neural Networks
- a. There is an exploitation of parallelism for the training, layers, neurons, and weights.
4. Parallel Genetic Algorithms
- a. The computation of fitness and mutation are processed in parallel.
5. Parallel Cluster Analysis

- a. Clustering arranges data items into several groups, called clusters, so that similar items fall into the same group. This is done without any suggestion from an external supervisor, so classes are not given a priori but the algorithm must discover them.
- b. When used to classify large data sets, clustering algorithms are very computationally resource intensive.
- c. Parallelism in clustering algorithms can be exploited in the computation of both the similarity and the distance among the data items, by computing on each processor the distance or similarity of a different partition of items.
- d. Three main parallel strategies can be adopted in the clustering strategy: (1) In the independent parallel strategy, each processor uses the whole data set and it performs a different classification based on a different number of clusters. To get the load among the processors balanced a new classification is assigned to a processor that completed its task. (2) In the task parallel approach, each processor executes a different task and cooperates with other processors exchanging partial results. In partitioning methods, processors can work on disjoint regions of the search space using the whole data set (different attributes). In hierarchical methods a processor can be responsible for one or more clusters. It finds the nearest neighbor cluster by computing the distance between its cluster and the others. Then all the local shortest distances are

exchanged to find the global shortest distance between two clusters that must be merged. (3) In the SPMD approach, each processor executes the same algorithm on a different partition of the data set to compute partial clustering results. Local results are then exchanged among all the processors to get global values on every processor. The global values are used in all processors to start the next clustering step until a convergence is reached or a certain number of steps are executed. The SPMD strategy also can be used to implement clustering algorithms where each processor generates a local approximation of a model (classification) that at each iteration can be passed to the other processors that can use it to improve their clustering model.

Data mining comprises a variety of goals and tasks. Knowledge discovered by a mining algorithm is highly application dependent and has an inherently subjective aspect. Knowledge evaluation requires subject-matter experts and concepts from artificial Intelligence and statistics. The efficiency of a mining algorithm is much less dependent on its particular applications, and it can be significantly improved in an objective way. For example, by using parallel processing. Efficiency evaluation requires only algorithmic and database related concepts. A good data mining algorithm must be effective and discover high-quality knowledge as well as be efficient and run fast.

## CHAPTER 5

### CHALLENGES AND FUTURE RESEARCH

#### *5.1 Algorithmic Issues*

Most of the algorithms in data mining assume the data to be noise-free. As a result, the most time-consuming part of solving problems becomes data preprocessing. The concept of noisy data can be understood by the example of mining web logs. A user may have gone to a Web site by mistake (by typing an incorrect URL or pressing a wrong button). In such a case, the information entered in the web log is useless. Web logs may contain many such data items. These data items constitute data noise. A database may constitute 30–40% such noisy data and preprocessing this data may take up more time than the actual algorithm execution time. Data mining systems rely on databases to supply the raw data for input and this raises problems in that databases tend to be dynamic, incomplete, noisy, and large. Other problems arise as a result of the adequacy and relevance of the information stored. A database is often designed for purposes different from those of data mining, and sometimes the properties or attributes that would simplify the learning task are not present—nor can they be requested from the real world. Inconclusive data cause problems because if some attributes essential to knowledge about the application domain are not present in the data it may be impossible to discover significant knowledge about a given domain. Databases are usually contaminated by bad data, so it cannot be assumed that the data they contain are entirely correct. Attributes, which rely

on subjective or measurement judgments, can give rise to errors such that some examples may even be misclassified. Since noise badly affects the overall accuracy of the generated rules, it is highly desirable to eliminate noise where possible to do so. Missing data can be treated by discovery systems in a number of ways such as simply disregarding missing values, omitting the corresponding records, inferring missing values from known values, treating missing data as a special value to be included additionally in the attribute domain, or averaging over the missing values using Bayesian techniques. Noisy data in the sense of being imprecise are characteristic of all data collection and typically fit a regular statistical distribution such as Gaussian. In contrast, wrong values are data entry errors. Statistical methods can treat problems of noisy data and separate different types of noise. Uncertainty refers to the severity of the error and the degree of noise in the data. Data precision is an important consideration in a discovery system. Databases tend to be large and dynamic in that their contents are ever-changing as information is added, modified, or removed. The problem with this from the data mining perspective is how to ensure that the rules are up-to-date and consistent with the most current information. Also the data mining system has to be time-sensitive as some data values vary over time and the discovery system is affected by the timeliness of the data. Another issue is the relevance or irrelevance of the fields in the database to the current focus of discovery.

## **5.2 Systems Issues**

Parallel computation offers many challenges. For example, load-balancing among processors while keeping the cost of communication among processors as low as possible; reducing the requirement of synchronization of the processors; and using aggregate memory effectively. Data mining requires computation-intensive operations on large data sets. These types of operations would not be practical without the emergence of powerful SMP workstations and high-performance clusters of workstations supporting protocols for high-performance computing. Distributed data mining can require moving large amounts of data between geographically separated sites, something that is now possible with the emergence of wide area high-performance networks. A time-consuming part of the data mining process is preparing data for data mining. This step can be streamlined in part if the data are already in a data warehouse. Some algorithms, such as association algorithms, are closely connected to databases, while some of the primitive operations being built into tomorrow's data warehouses should prove useful for some data mining applications. Massive data sets often generated by complex simulation programs required graphical visualization methods for best comprehension. Recent advances in multi-scale visualization allow the rendering to be done far more quickly and in parallel, making these visualization tasks practical.

## **5.3 Summary**

Data mining is a computer science discipline in development. New techniques and software appear frequently, many of them untested. Old

techniques have been prematurely rejected. In recent years the following new areas of data mining have been emerging:

1. *Text and Web-content mining*: Web mining will reach new levels with XML and semantic Web applications. The Web is already a vast repository of useful knowledge, but with advances in intelligent agents, widespread XML adoption, and Web services, Web mining will reach a new level.
2. *Relational mining and link analysis*: This will have applications in many fields, including biology, business, library and information science, marketing, and security and perhaps will create completely new areas.
3. *E-commerce*: We already see that the successful e-commerce companies such as Yahoo, Amazon, and eBay are investing in data mining in order to extract value from their data. It is expected these efforts will continue and expand also into intelligent bots.
4. *Bioinformatics*: This includes analysis of genomic data and DNA microarrays. Successful results have already been reported, for example, highly accurate microarray-based diagnostics.
5. *Multimedia data mining*. Images and video can already be found based on a keyword. Combined with image recognition, many interesting tasks can be envisioned based on actual understanding of images, video, and audio.

## REFERENCES

- Agrawal, R., & Shafer, J. (1996). Parallel mining of association rules. *IEEE Transactions on Knowledge and Data Engineering*, 962–969.
- Agrawal, R., & Srikant, R. (1994, September). *Fast algorithms for mining association rules*. Paper presented at the Proceedings of the 20th International Conference on Very Large Databases, Santiago, Chile.
- Agrawal, R., & Srikant, R. (1995, March). *Mining sequential patterns*. Paper presented at the Proceedings of the 11th International Conference on Data Engineering, Taipei, Taiwan.
- Berry, M. J. A., & Linoff, G. S. (2000). *Mastering data mining: The art of science of customer relationship management*. New York: John Wiley.
- Bishop, C. (1995). *Neural networks for pattern recognition*. Oxford: Oxford University Press.
- Cheeseman, P., & Stutz, J. (1996). Bayesian classification (AutoClass): Theory and results. In U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, & R. Uthurusamy (Eds.), *Advances in knowledge discovery and data mining* (pp. 61–83). Cambridge, MA: AAAI Press/MIT Press.
- Dhillon, I. S., & Modha, D. S. (1993). *A data-clustering algorithm on distributed memory multiprocessors*. London: Springer-Verlag.
- Fayyad, U. M., Piatetsky-Shapiro, G., Smyth, P., & Uthurusamy, R. (1996). *Advances in knowledge discovery and data mining*. Menlo Park, CA: AAAI Press.

- Flynn, M. J. (1972). Some computer organizations and their effectiveness. *IEEE Transactions on Computers*, 21, 948–996.
- Frawley, W., Piatetsky-Shapiro, G., & Matheus, C. (1991). Knowledge discovery in databases: an overview. In G. Piatetsky-Shapiro & W. Frawley (Eds.), *Knowledge discovery in databases* (pp. 1–27). Cambridge, MA: MIT Press.
- Freitas, A., & Lavington, S. H. (1998). *Mining very large databases with parallel processing*. Dordrecht, The Netherlands: Kluwer.
- Garofalakis, M., Rastogi, R., & Shim, K. (1999, September). *SPIRIT: Sequential pattern mining with regular expression constraints*. Paper presented at the Proceedings of the 25th Very Large Database Conference, Edinburgh, Scotland.
- Joshi, M., Karypis, G., & Kumar, V. (1998). *ScalParC: A scalable and parallel classification algorithm for mining large datasets*. Paper presented at the International Parallel Processing Symposium.
- Kaufmann, L., & Rousseeuw, P. J. (1987). Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20, 53–65.
- Karypis, G., & Kumar, V. (1994, October). *IEEE Transactions on Parallel and Distributed Systems*.
- MacQueen, J. (1967). *Some methods for classification and analysis of multivariate observations*. In L. M. Le Cam & J. Neyman (Eds.), *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics*

*and Probability* (vol. 1, pp. 281–297). Berkeley: University of California Press.

Morse, S., & Isaac, D. (1998). *Parallel systems in the data warehouse*.

Englewood Cliffs, NJ: Prentice-Hall.

Moxon, B. (1996, August). *The hows and whys of data mining, and how it differs from other analytical techniques*. DBMS Data Warehouse Supplement.

Pizzuti, C., Talia, D., Foti, D., & Lipari, D. (2000, May). *Scalable parallel clustering for data mining on multicomputers*. In Proceedings of the 3rd International Workshop on High Performance Data Mining HPDM00-IPDPS, Cancun, Mexico. Lecture Notes in Computer Science (Vol. 1, pp. 390–398). New York: Springer-Verlag.

Rastogi, R., Garofalakis, M., Seshadri, S., & Shim, K. (1999). *Data mining and the Web: Past, present, and future*. Paper presented at the Workshop on Web Information and Database.

Shafer, J., Agrawal, R., & Mehta, M. (1996, September). *SPRINT: A scalable parallel classifier for data mining*. Paper presented at the Proceedings of the 22nd International Conference on Very Large Databases, Bombay, India.

Stoffle, K., & Belkonience, K. (1999). *Parallel k-Means clustering for large datasets*. Paper presented at the Proceedings of EuroPar.

Witten, I. H., & Frank, E. (2000). *Data mining practical machine learning tools and techniques with Java implementations*. San Francisco: Morgan Kaufmann.

Xu, S., & Zhang, J. (2002). An improved parallel algorithm for web document clustering.