Integrating Data Warehouse Methodology into Human Decision Making

Sam Y. Sung
Department of Computer Science
South Texas College
McAllen, TX 78501
sysung@southtexascollege.edu

Robert Ho
Department of Computer Science
South Texas College
McAllen, TX 78501
shusin2@southtexascollege.edu

ABSTRACT

An intelligent decision support system using data warehouse technique is proposed. In this system, the decisions made in the past by each expert, their status (success/failure), and background information such as under what conditions that the decisions were made are all recorded and fed into the data warehouse. Once sufficient data is collected, and on-line analytic approach (OLAP) is used, then the system will be able to efficiently draw scores to each of the decisions. Combining weights with the scores of decisions enables a manager to rank all decisions. The system is generic and can apply to a number of applications.

1. INTRODUCTION

In today's global competitive environment, the need to make better, quicker decisions is critical, and the key to success is information gathering and analysis. Access to past critical information can help companies gain a competitive edge, increase customer satisfaction, expand market share, and enhance profitability. Countless business and economic pressures are forcing companies to radically re-think their information technology infrastructures to take advantage of the vast amounts of data typically stored across multiple, dispersed and non-integrated sites. For the most part, the fundamental value of that data has yet to be uncovered.

Artificial Intelligence (AI) technologies, including neural network (Ritter, Martinetz, and Schulton, 1992) and knowledge base systems (Devanbu, Brachman, and Ballard 1991; Devanbu, Selfdge, Ballard, and Brachman, 1989; Gains and Boose, 1988), have been widely used in a broad range of industrial, medical, and scientific applications. In particular, decision support systems (DSS) have produced significant interest in the recent years and became a promising area of scientific research for applying intelligent computerized system as a substitute for producing human-like decisions. In general, these systems provide either local or global optimization for a set of well defined and constrained environment. However, in real business world, human decisions are always mandatory and impossible to be replaced by a computerized system. This is largely due to the fact that, computation complexity of human decisions, particularly when they are related to the strategic success or failure of the companies and business, people rarely consider making use AI (Sefridge, 1992). As a result, strategic decision-making today is by and large, still dependent on the human.

Once we realize that strategic decision for a business corporation will always require to be made by the corporate decision makers of a company, and further identify the complexity

of the business problem and their data scope are too massive for a computer system to analyze, we concluded it is impractical to use computerized system to simulate the human decision for strategic management process. Rather than designing an autonomous decision support system to replace human decision, we need to design a novel personalized decision-assistance system which can support the corporate decision makers of a company to filter through the complexity of the data domain, to relieve the corporate decision makers from analyzing the problem complexity, to avoid the corporate decision makers from being confused by the environmental artifacts, and to allow the corporate decision makers to make better decisions during critical business situations.

A close study reveals that strategic decisions are very much company-eccentric, namely they would require large numbers of external and internal factors and complex information as the input. They further require not only the present scenario, but also the previous historical background of the company and past decisions. As human thinking is fuzzy in nature, binary computation cannot approach the solution more appropriately than a human. In summary, the exhaustive input data analysis, and the extensive computation that is required for the large neural network, makes the actual implementation totally impractical (Jamshidi, Vadiee, and Ross, 1993; Kosko, 1992; Sung, Shaw, and Fu, 1996).

On the other hand, data warehouses and OLAP (On-Line Analytic Processing) have recently become the focus of corporate information management with the availability of advanced database technology (http://www.tdwi.org). In a simple description, a data warehouse is an integrated data repository containing historical data of an enterprise. In OLAP, data is often stored in summarized form, as a histogram of aggregates (Counts, Sums, or Averages) over specified ranges.

The need for data warehousing and OLAP first came from industry to satisfy business needs and to create competitive advantages over the competition. The research community has begun to pay attention to data warehousing and OLAP only recently, but there is still a big gap between industry and the research community (Barquin and Edelstein, 1997; Chaudhuri and Dayal,1997; Kimbal, 1996).

In this paper, we propose a design of using a data warehouse integrated with human decision making. The rational of our approach is based on the fact that managers often review similar past experiences in a search for information that might be useful in solving the present situation.

In our system, the decisions made in the past by each expert, their status (success/failure), and background information such as under what conditions that the decisions were made are all recorded and fed into the data warehouse. Once sufficient data is collected, and on-line analytic approach (OLAP) is used, then the system will be able to efficiently draw scores to each of the decisions. Combining weights with the scores of decisions enables a manager to rank all decisions, and the ones with the highest ranks cab assist his or her final choice.

The rest of the paper is organized as follows. Section 2 shows the architecture of the proposed system and describes the key components making up the system. In order to illustrate the system, we have chosen the ROLAP (Relational OLAP) as the model and outline the various stages of using the system. In Section 3, we described in details of the decision extraction engine. Finally, we provide concluding remarks in Section 4.

2. PROPOSED INTELLIGENT DECISION MAKING SYSTEM

The system we proposed is generic and can be applied to different business decision templates such as insurance risks, medical treatments, and so on. There are basically three components in the architecture:

A Data Warehouse to Store Past Decision Information. Initially, knowledge elicitation from a group of domain experts is required. The data warehouse will encapsulate the experts' knowledge in structuring the various background information and reduce it into a smaller set of attributes. Information gathered for different business such as the medicine, insurance, telecommunication and so on, must be carried out independently since different business can exhibit specific environmental factors leading to a choice of different attributes. In the next step, the user clients (managers) will be requested by the system to feed into the data warehouse a substantial volume of past decisions (together with each decision's background).

OLAP Interface. Once the information collected in the data warehouse becomes statistically usable, it can be used for facilitating current decision making. In this phase, the managers are prompted with numerous OLAP GUI interfaces to allow the managers to input the correct present situation. The managers' assessments are subsequently assimilated to guide the system to formulate a query.

A Decision Extractor. The queries obtained from the previous component will be used to retrieve the answer from data warehouse. The answer retrieved is a set of decisions, each of them has associated with a number of scores. The decisions are then ranked using scores and weights. The three criteria for our ranking rule are: *confidence*, *support*, and *strength*. This component is also our key focus in this paper.

The present decisions made by the manager, after evaluated, will be stored back into the current database for future use. This is achieved in the post-processing phase which is used to refine and enhance the system.

The advantage of our proposed computerized decision system is that, although the decisions initially need to be made by the human manager, it will be able to, after a sufficient period of accumulative experience (i.e. the past decisions are continuously recorded and fed into the data warehouse), gain a strong self support.

Details of the three components are described in the following. The first component is to formulate the data warehouse.

2.1 Building the Data Warehouse

Our goal here is to create and maintain a *specialized* data warehouse for the purpose of storing information to be used by decision making. First, we describe the data model we use.

2.1.1 The Data Model

Star schema: A star schema is an arrangement of tables in a relational database where a central "fact" table is connected to a set of "dimension" tables, one per dimension. The name "star" comes from the usual diagrammatic depiction of this schema with the fact table in the center and each dimension table shown surrounding it, like the points on a starburst (Ramakrishnan, 1998).

In our model, we assume a company has a number of managers to give their decisions on a number of events. The company has accumulated all the managers' decisions over a certain period of time, and each decision can be verified whether is correct or not. The company has stored this information in a table DECISION. A tuple *t* in the DECISION table looks like

$$t = \langle E5, T14, P2, \ldots, D3, "1" \rangle \quad (1)$$

Tuple *t* means manager *P2* made his/her decision on event *E5* at time *T14*, and this decision is *correct*.

When an event *E* occurs, the company wants to benefit from this DECISION table. The decision-maker would input the background information of event *E*, and the system will find all past events that match E's background, and return *all decisions* made on them together with the "*correct or not*" information.

For example, suppose there are 6 past events that match a current event *E*, and 4 of them made D1 as their decision, the rest 2 made D2 as decision. Further, assume the correct rate found is 75% and 50% on D1 and D2, respectively. The decision-maker then decides to choose D1 as his/her decision because D1 has the highest successful rate (75%) and statistical support (4).

However, in reality, there are several issues that need to be concerned.

2.1.2 Issues to Concern

The above assumes that we have enough data on the particular event *E* that the company wants to decide on. However, in case there is not enough events matching on this particular event *E*, what we can do is to use the events that is close (or similar) to the event *E*.

In the table DECISION, the domain of attribute *result* is 1 or 0, indicating correct or incorrect decision, respectively. In reality, of course, a decision can also be evaluated by "how" successful it is, instead of just "whether" it is successful; in that case, a real value between 0 and 1 may be used. However, we do not consider this scenario in this paper since it does not affect our discussions in general.

The most important task in building the data warehouse is to set up the background attributes. There are three types of attributes: *categorical* attributes, *quantitative* attributes, and *dependent* (or *conditional*) attributes.

An attribute is a categorical attribute if there is no natural ordering among the attribute values. Otherwise, it is a non-categorical attribute or quantitative attribute. Categorical attributes are discrete values and usually, the number of categorical attribute values is small. Examples of categorical attributes are “color”, “model of car”, “zip-code”, etc. Examples of quantitative attributes include “height”, “age”, “salary”, “temperature”, “interest rate”, etc. Notice that artificial ordering can sometimes be imposed on a categorical attribute using a “ranking” function. For instance, an insurance company may rate a neighborhood as {“good”, “average”, “bad”} using zip-code attribute.

Quantitative attributes has small to large range of values, and two neighboring (or two consecutive) values have strong relationship. Example of a small range quantitative attribute is the degree of competition: weak, medium, and strong. Example of a large range quantitative attribute is "budget", or an attribute that uses percentage.

Data dependency represents another important type of knowledge. A dependency exists between two attributes if the value of one attribute can be used to determine (or derive) the value of the other: $A=a \rightarrow B=b$, where A and B are attributes and a and b are values. We call A as *premise* attribute(s) (or *premise* for short), and B a dependent attribute. Notice that A can be a collection of attributes and a premise can be itself a dependent attribute. Thus, dependent attribute can be generalized to multiple levels.

In recent years, dependent attribute has been studied by researchers in the area of generalized or multi-level association rules (Han and Fu, 1995; Srikant and Agrawal, 1995). However, they are usually the "is-a" type of dependency (or taxonomy), and most dependencies are exact rather than probabilistic. For example, Jacket is-a Outerwear, Outerwear is-a Cloth, etc. Often, dependencies are more complex in social and economical behavior. Dependent attributes are derived from other attributes, and they may be derived using complex fuzzy rules. An example of a complex dependent attribute is shown in Figure 1.

Type of Business				
Premises	1 (Fragmented)	2 (Declining)	3 (Mature)	4 (Emerging)
Number-dominant competitors	high			

Economies-scale	high			
Product-differentiation	high			
Regionally-dispersed	high			
business-unit-sales		decreasing	stable	increasing
product-lines		decreasing	stable	increasing
R&D-budgets		decreasing	stable	increasing
advertising-budgets		decreasing	stable	increasing
number-competitors		decreasing	stable	increasing
technology		mature	mature	emerging
proportion-new companies		Low	Average	high
proportion-new customers		low	average	High

Figure 1. Example of a dependent attribute

In Figure 1, the dependent attribute is *type_of_business*. There are 12 premises and 4 rules accepting fuzzy variables like decreasing, stable, increasing, low, average and high. These premises serve to determine the type of business involved. The 4 rules are described as below.

Rule-1: If (number of dominant competitors is high) and (economies-scale is high or product-differentiation is high or regionally-dispersed is high) then type-of-business is fragmented.

Rule-2: If (business-unit-sales is decreasing and product-lines is decreasing and R&D-budgets is decreasing and advertising-budgets is decreasing and number-competitors is decreasing and technology is mature and proportion-new companies is low and proportion-new customers is low) then type-of-business is declining.

Rule-3: If (business-unit-sales is stable and product-lines is stable and R&D-budgets is stable and advertising-budgets is stable and number-competitors is stable and technology is mature and proportion-new companies is average and proportion-new customers is average) then type-of-business is mature.

Rule-4: If (business-unit-sales is increasing and product-lines is increasing and R&D-budgets is increasing and advertising-budgets is increasing and number-competitors is increasing and technology is emerging and proportion-new companies is high and proportion-new customers is high) then type-of-business is emerging.

In fact there are many ways to formulate rules and to combine them. The above is just one way to systematically formulate the rules. The rules can be made more complicated such as "if 60% of premises appear in the antecedent of rule-2, then the business is declining with probability 60%". This rule will lead a probabilistic type of dependency.

Premises that are combined to derive a dependent attribute value usually have similarity among themselves, but the similarity is more in concept rather than just numeric values. These are the attributes that need domain experts to focus on.

Similar to taxonomy information, the dependent attribute values can be either stored in the database, or derived on the fly (Han and Cercone, 1993; Han and Fu, 1995) The trade-off, of course, is between storage space and processing time. For each of the past decision made, we assume that all dependent attributes are stored although this assumption is not a necessary requirement. The type of decision and whether it is successful are also stored.

The second component is a facility that enables the users to assess the correct present situation, and guide the OLAP toward a final query.

2.2 Building the OLAP Interface

To formulate a user's query, the user is asked questions about the current situation. The questions need to be asked sequentially one by one in logical order so that the user can be

guided toward to a final assessment. There are two kinds of approaches: (1) top-down and (2) bottom-up.

In top-down processing, the more general information is asked first, then, if conclusions cannot be made, gradually down to more specific questions. Or, ask the dependent attribute (parent) first, then gradually down to the premise attributes (children). Bottom-up processing is similar but reverse in which detail information is asked first, then general question followed if there is no conclusion can be made. The advantage of top-down approach is the user can stop earlier if he feels the answer is good enough, but the tradeoff is refined information may be missed.

A prototype model has been developed in (Sung et al., 1996) that is useful in assisting the system to draw specific questionnaires. These questionnaires provide a framework for formulating the user's query space. In arriving at the final query, two factors must be emphasized.

- The questions asked will have to be precise and to the point but with options given to the user. For example, the answers to the questions will have to be expressed as a percentage of 100 or divided into various relative values such as very weak, weak, average, strong, and very strong. Sub-questions may be needed in certain situations.

- The formulation of certain questions for some areas can be difficult. In the event when getting a direct answer is difficult, SURROGATE question should be formulated. For example, instead of asking "what is the morale of the workers in the company?" we could use "How is the punctuality and overtime rate of the workers?" These should give a guideline to getting an answer in an indirect way.

A user when assessing a situation may prefer to give a range of values for quantitative type of attributes. The rational for this is:

First, human knowledge tends to be fuzzy. Many times we know something but at the same time we may not know everything, e.g., we are not exactly sure but have strong confidence that the growth rate is going to be between 5% and 7%. Or, we knew the number of competitors currently is 3 but may become 2 or 4 anytime in the near future.

Second, we wish the decision to be resilient, so that in case of situations get better or worse, the decision may still be correct. For example, allowing the interest rate to vary between 3.5% to 5.5% to give a buffer of 2%.

Third, if there simply has no enough historical cases to satisfy a point query. Therefore, our system needs to handle range query as well. Notice that each range query has a center but which is not necessarily the geometric center of the range. A center can be user specified. For examples, interest rate is currently at 4% but estimated to be within the range of 4~5% in the next 3 months, and the center of the estimation is considered at 4.3% which is the most likely to happen. Or, a center can be system generated. For example, in a cooperative decision-making environment, the mean and standard deviation are used as the center and the range, respectively.

The managers' assessments collected through the interfaces are assimilated to formulate into a query which then be used by the decision extractor to retrieve answers, and that is the third component described in the next section.

3. A DECISION EXTRACTOR

This part of system concerns that, given a query, to extract all the *qualified* decision alternatives to facilitate users to make a final decision. The *qualification* is measured by *similarity* between a decision's background attribute values and the query's specified values. The similarity is computed as the distance between the candidate decision and the center of

the query (i.e., the difference of their attribute values). In our work we use Manhattan distance but other distance like Euclidean can also be used. In general, the smaller the distance is, the more qualified a decision will be.

3.1.1 Concept used in Retrieval

Now we are to give our rule extraction method. First, we assume the query input is in the form of a rectangular region ***Q*** and a center point ***c***, shown as below

$$Q =< A_1 = a_1, A_2 = a_2, ..., A_m = a_m, l_1 \le B_1 \le u_1, l_2 \le B_2 \le u_2, ..., l_n \le B_n \le u_n > \quad (2)$$

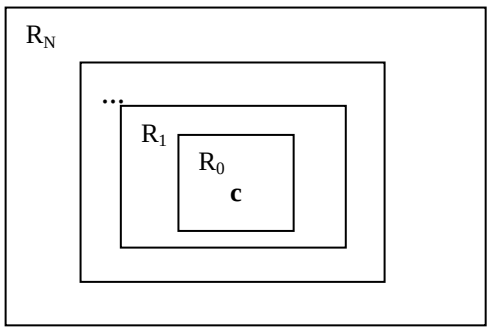
$$c =< A_1 = a_1, A_2 = a_2, ..., A_m = a_m, B_1 = b_1, B_2 = b_2, ..., B_n = b_n > \quad (3)$$

Where a_i is the value of the fixed-point attribute A_i, and b_j is the center value of the range attributes B_j, for $1 \le i \le m$ and $1 \le j \le n$.

A sequence of rectangular regions $R_0, R_1, R_2, ..., R_N$ are constructed in such a way that

$$c \in R_0 \subset R_1 \subset R_2 \subset ... \subset R_N \subseteq Q$$

We can map the regions into a 1-dimensional interval in a linear manner. For example, as shown in figure 3.1, the (N+1) rectangles have been mapped into a 1-dimensional of (N+1) subintervals $\{R_0, R_1\text{-}R_0, R_2\text{-}R_1, \ldots, R_N\text{-}R_{(N-1)}\}$.



decision ***d'***

u	50	50	50	...	...	40
v	45	40	35	...	...	30

R_0	R_1-R_0	R_2-R_1	...	..	R_N-$R_{(N-1)}$

Figure 3.1 visualization of rule extraction

For each rectangular region, *2n* values are stored, where n is the number of range attributes. For each type of decisions, two vectors ***u*** and ***v*** of size *(N+1)* each are maintained. The vector *u* is used for storing all tuples, and vector *v* is used for storing only tuples that have decisions made are successful. Thus, suppose there are *n* range attributes and *d* decisions, the total space is O(N×(n+d)). In practice, this volume can be completely stored in main memory.

For each tuple in the database, if the tuple falls into one of the rectangular regions then the *u* and *v* vector of the corresponding decision is updated. For example, suppose the decision made by a tuple *t* is the decision *d'* which has currently *u* and *v* values as shown in figure 3.1. If the tuple *t* is found belonging to region R_2 but not R_1, and the decision made is successful, then the corresponding (*u, v*) value is changed from (50, 35) to (51, 36).

To find which rectangular region a tuple belongs to takes O((*log*N) × n) of comparisons. The database is scanned only once. The I/O cost is therefore one database scanning.

3.2.1 Retrieval Implemented by SQL Queries

Instead of implementing the retrieval procedures in terms of file processing, we choose to implement by using SQL (embedded in Pro C). This is due to the popularity that SQL has become the standard language used in relational databases.

The details are outlined as follows.

1) Define a minimum support, MINIMUM, that a decision must have.
2) Define a time point THIS_TIME (can be optional), consider only those events after this time point. This may be necessary if the company has accumulated many data on this query, and wants the decision to be up to date, not greatly affected by the long history data which may not reflect the rapidly changing world.
3) Establish the following view (can be table) which selects those events that satisfy the query conditions.

   CREATE VIEW total_event AS
   SELECT *
   FROM event_background
   WHERE $l_1 \le con\_1 \le u_1, \ldots, l_n \le con\_n \le u_n,$ //i.e., conditions satisfy the ranges of query Q//
4) Establish the view which stores all qualified decisions together with their supports.

   CREATE VIEW qualified_decision AS
   SELECT did, count(*) as total_number, SUM(result) as success_number
   FROM total_event T, decision D
   WHERE *T.eid=D.eid and timeid > This_time*
   GROUP BY did
   Having count(*)>=MINIMUM

The view qualified_decision will give a preliminary answer. To get a more advanced answer, we need to derive the vectors ***u*** and ***v*** for all (*n+1*) regions.

5) Formulate the following view (or table).

   CREATE VIEW qualified_event AS
   SELECT T.*, did, result //T.* here represents all attributes in T//
   FROM total_event T, decision D
   WHERE T.eid=D.eid and did in (SELECT did FROM qualified_decision)
6) We then perform *n* iterations to generate the decisions and their corresponding ***u*** and ***v*** values for all (*n+1*) regions. The results are stored in the view qualified_decision we created in 4) above.

   // For each iteration *k* (*k=1,…,n),* perform the following//
   INSERT INTO qualified_decision
   SELECT did, count(*) as U, SUM(result) as V
   FROM qualified_event
   WHERE $l_1 \le con\_1 \le u_1, \ldots, l_n \le con\_n \le u_n,$ //i.e., conditions must satisfy the ranges Of rectangular region R_{n-k} //
   GROUP BY did

3.3 Tests and Results

We have performed tests on a real data warehouse containing a loan-application database. The database has three tables, namely customer, evaluator, and application-results. There are 100,000 customers and 1,000 evaluators. The loan-application database can be downloaded from (http://www.cs.uh.edu/~ssung).

The queries are classified into two types: fixed and partitioned. The difference between fixed and partitioned queries is that the answer returned for a partitioned query is a set of (u, v) values, with each (u, v) corresponding to a region, as we described in the Section 3.1. In comparison, a fixed query has only a pair of (u, v) value returned.

We have performed several tests on the loan-application database. A typical example is described here.

"Given a customer having the following specified:
{Gender = 'male', Marriage_Status = 'single', Race = white, nation = 'American citizen', Education = 'Bachelor', Own_Rent_Other = 'Rent', Annual_Salary=38K, Other_Income=1K, Age=25, Number_of_dependents=0, Length_at_current_addr = 2, Occupation = 'Prof-specialty'},

Find out the probability that this application is to be approved, based on the past similar cases".

A user first formulates a query with point query. The answers returned are exactly matched with the query specification. Suppose the result returned is 1 approved and 0 reject. The user is not satisfied because the number of past case is below the minimum support level, say 10. Next, the user issued a range query like:

{Gender = 'male', Marriage_Status = 'single', Race = white, nation = 'American citizen',
Education = 'Bachelor', Own_Rent_Other = 'Rent', Annual_Salary = [36K, 40K],
Other_Income = [0,2K],
Age = [24,26], Number_of_dependents=0, Length_at_current_addr = [1,3], Occupation =
'Prof-specialty' }

Suppose the answer returned is: The number of cases satisfying criteria is 24. There are 20 approved, 4 rejected. At this point, the user might stop if satisfied with the result. Or, the user can opt for issuing a partitioned query. The partitioned query will allow the user to find more detailed summary information in each and every region, so that better decisions can be made.

4. CONCLUSION

In essence, a scheme in integrating data warehouse technique into the decision support system is introduced. The system will be able to model the manager's decisions after a certain period of training provided sufficient data are recorded into the system.

As we launch into our system research, we found that decision-makers and strategic planners often review their past experiences in a search for similar patterns that might be useful in solving the present situation. We further found that strategic planning knowledge-based systems are feasible because:

- Expert decision skills are required to do the strategic planning job: This provides us the confidence that if a system can be designed to collect the knowledge and experience of these experts, then such system should be able to assist humans in making business decisions.
- There are recognized experts in the field: This implies that the knowledge is highly specialized to particular industries. Naturally, a knowledge base with data analyzer provides the technological tools to learn and use such knowledge.
- The job requires information judgment: This implies that a preprocessor such as the Fuzzy Logic Reasoning subsystem (Yamakawa, 1990; Yamakawa and Furukawa, 1992) is ideal as it reduces the data volume and removes the ambiguity that commonly exist in business knowledge and information.
- A number of specific company strategy planning tasks, especially where small companies or strategic business units are involved, involve manageable tasks that can be defined with sufficient precision to enable knowledge-based systems development.

We are encouraged with our findings to date. However, we also see the need for future work that proposes additional challenge. We will further continue the comparative study of our system with the expert system based system in order to identify the performance trade off. We will also explore and select a variety of neural network model for decision making. Finally, we will attempt for a broader range of applications for our computation model.

REFERENCES

Barquin, R. and Edelstein, H. (1997), *Planning and Designing the Data Warehouse*, Prentice-Hall.

Chaudhuri S. and Dayal, U. (1997), "An overview of data warehousing and OLAP technology", *SIGMOD Record*, 26(1):65-74.

Devanbu, P., Brachman, R.J., Ballard, B. (1991), "LaSSIE: A Knowledge -Based Software Information System", *CACM* 34:35-49.

Devanbu, P., Selfdge, P.G., Ballard, B., and Brachman, R.J. (1989), "Steps Toward Knowledge Based Information System", Proc. *IJCAI-89*: pp 110 -115.

Fukuda T., Morimoto Y., Morishita S., and Tokuyama T., Data Mining Using Two-Dimensional Optimized Association Rules: Sceme, Algorithms, and Visualization. *In Proc. of 1996 SIGMOD Conference*, pp. 13-23. Montreal, Quebec, Canada, June 1996.

Gains, B.R., and Boose, J.H. (1988), Eds., *Knowledge -Based Systems: Knowledge Acquisition for Knowledge -Based Systems*, vol. 1, Academic Press, NY.

Han J., Cai Y., and Cercone N., Data-driven discovery of quantitative rules in relational databases. *IEEE Trans. Knowledge and Data Engineering*, 5:29-40, 1993.

Han J. and Fu Y., Discovery of Multiple-Level Association Rules from Large Databases. *In Proc. of 1995 VLBD Conference*, pp. 420-431. Zurich, Switzerland, 1995.

Jamshidi M., Vadiee N., and Ross T. (Eds) (1993), *Fuzzy Logic and Control, Software and Hardware Applications*, Prentice Hall.

Kimball, R. (1996), *The Data Warehouse Toolkit*, John Wiley and Sons.

Kosko, B. (1992), *Neural Networks and Fuzzy Systems*, Prentice Hall, 1992.

Ramakrishnan, R. (1998), *Database Management Systems*, McGraw-Hill.

Ritter, H.J., Martinetz, T.M., and Schulton, K.J. (1992), *Neural Computation and Self-Organizing Maps*, Addison-Wesley.

Srikant R. and Agrawal R., Mining Generalized Association Rules. *In Proc. of 1995 VLBD Conference*, pp. 407-419. Zurich, Switzerland, 1995.

Sefridge, P.G. (1992), Workshop Report: *Applying Artificial Intelligence to Software Problems: Assessing Promises and Pitfalls*, IEE Expert, July, 1992.

Sung, S., Shaw, V., Fu, J. (1996), "Integrated Strategic Management System", *First International Conference on Industrial Engineering Applications and Practice*, Houston.

Data Warehouse Education and Solution. (www.dw-institute.com)

Department of Computer Science, *University of Houston.* (http://www.cs.uh.edu/~ssung)

Yamakawa, T. (1990), "Pattern Recognition System Employing a Fuzzy Neuron", *International Conference on Fuzzy Logic and Neural Networks*, Iizuka, Japan, pp 943 -948.

Yamakawa, T. and Furukawa, M. (1992), "A Design Algorithm of Membership Functions for a Fuzzy Neuron Using Example-Based Learning", *IEEE International Conference on Fuzzy Systems*, pp 75 -82.