1995

Semi-autonomour knowledge discovery in databases

David M. Grube
Lehigh University

Grube, David M.

Semi-Autonomous Knowledge Discovery in Databases

January 14, 1995

Semi-Autonomous Knowledge Discovery in Databases

by

David M. Grube

A Thesis

Presented to the Graduate and Research Committee

of Lehigh University

in Candidacy for the Degree of

Master of Science

in

Computer Science

Lehigh University

December 7, 1995

This thesis is accepted and approved in partial fulfillment of the requirements for the Master of Science.

December 4, 1995

Date

Thesis Advisor

Co-Advisor

Chairperson of Department

Table of Contents

Abstract - Knowledge Discovery in Databases and Database Mining is currently a very active research area. Database use and the volume of data in these databases grows larger each day, creating an important need to extract knowledge and reach informed conclusions from this data. One of the critical problems with this extraction of knowledge and patterns from the data is the extraction and presentation of only the data that an individual user finds interesting, according to each user's own, subjective criteria, without restricting the search of data as to overlook potential patterns and discoveries. The focus of this paper will be on the specific problem area of embodying user-oriented, domain-independent subjective interestingness in a knowledge discovery system. Specifically, this paper attempts to provide a framework of techniques that may be employed to capture an individual user's subjective interests via the underlying database objects, expanding these objects to include related database objects, and saving and representing these sets of objects to represent the user's beliefs of the database. In this manner, discovered patterns and knowledge may then be related directly to the user's interests via stored abstractions and concepts. This paper does not discuss specific pattern matching and discovery algorithms; rather, it provides a framework for capturing the user's subjective interests and then performing knowledge discovery on the objects that represent these interests. The techniques discussed in this paper build off of previous research in these areas, and provide an approach for knowledge discovery based upon subjective interests, in the hopes of laying a groundwork for future research in this area.

1. Overview

Knowledge discovery in databases is a very wide open, and multi-disciplinary research topic. To provide a point of reference for this paper, a definition of knowledge discovery is useful. Knowledge discovery is the process of extracting implicit, previously unknown, and potentially useful information from data [1]. The process of extracting the information from the data typically involves patterns found within the data. A pattern can be defined as a statement about a set of facts, or data, that describes the relationships among the facts (data), such that the statement is simpler than the individual facts (data) themselves [1]. Patterns are interesting according to subjective and objective measures, typically defined by whom is viewing the patterns. Patterns that are interesting and can provide an acceptable degree of certainty, are consider knowledge [1][2].

In the literature, the general process of knowledge discovery is broken down into two segments; knowledge discovery and database mining. It is useful to clarify the terms "knowledge discovery in databases" and "data mining", or "database mining", since they will be used through-out this paper. The phrases "data mining", or "database mining" may be used interchangeably, and will be defined as the algorithms employed to extract patterns from data. The phrase "knowledge discovery in databases" will be defined as the process of retrieving data from the data store, preparing (filtering, etc.) it, and then finding and interpreting any patterns that may exist in the data. The knowledge discovery process uses data mining to perform the pattern extraction and interpretation, and typically employs domain knowledge and hypothesis as part of the overall discovery process. From these definitions it is evident that data mining is a component of the overall knowledge discovery system. Note: for brevity, the capitalized "KDD" will be used as an abbreviation for the phrase "knowledge discovery in databases" and the capitalized "DM" will be used as an abbreviation for "data mining" or "database mining.

Figure 1 depicts a general model for the KDD/DM process. For the purposes of this paper, it is useful to have a basic understanding of the components of a KDD/DM system; however, it is not particularly critical how the actual KDD/DM system is implemented. The KDD/DM model in figure 1 is typically found in the literature, so it is a reasonable choice to give a general graphic depiction of the process. A brief

explanation of the components depicted in figure 1:

- User Interface: Responsible for user data entry and display of messages, results, etc.
- Controller: Makes decisions and directs the overall system execution based upon user input and domain knowledge.
- Focus: Determines what database data should be retrieved based upon direction from the controller. The focus component and the database interface typically determine data caching issues, what remains in core, what remains on disk, etc.
- Database Interface: Interprets input from the focus component, and performs the actual database data retrieval.
- Pattern Extraction: Looks for relationships among the elements of data returned from the database. Some simple forms of pattern extraction are dependency analysis, class identification, concept description, deviation detection, characterization.
- Pattern Evaluation, or Interestingness: Determines the relative degree of interest among the extracted patterns, and determines what to display to the user, and in what order.

The KDD/DM model depicted in figure 1 is generally accepted in the literature. However, KDD/DM system models and architecture is constantly changing; for the purposes of this paper, we use the model in figure 1.

A typical KDD/DM session would proceed as follows:

1. The user enters domain knowledge/bias, etc., as part of the user input. Note: this implies an interactive session. A non-interactive session would imply that the KDD/DM system performs it's knowledge discovery without being directed at all from the user, and would instead rely upon the database itself, and any domain knowledge supplied by a domain expert.
2. This information is passed to the controller, which potentially includes domain knowledge, and then directs the focus module what to do with the information.

3. The focus module potentially uses domain knowledge to assist it in it's process, and then determines which database objects and data are needed to complete the request. Once determined, it passes it's request to the database interface.
4. The database interface retrieves the data from the database, and returns the data to the focus module.
5. The focus component notifies the controller that it has completed it's process.
6. The controller notifies the pattern extraction module that the data is available.
7. The pattern extraction module performs the actual pattern extraction/discovery process, and when completed, notifies the controller that it is done.
8. The controller then directs the evaluation module (interestingness evaluator) to determine what is "interesting", and filter out only those patterns that are interesting.
9. The patterns and discoveries are displayed to the user in some format, and potentially fed back into the knowledge base for future use.

This session/process flow is typical ([1] [3] [5]); there are some variations on the basic model and process flow, but this is the general model and process that will be used for the purposes of this paper.

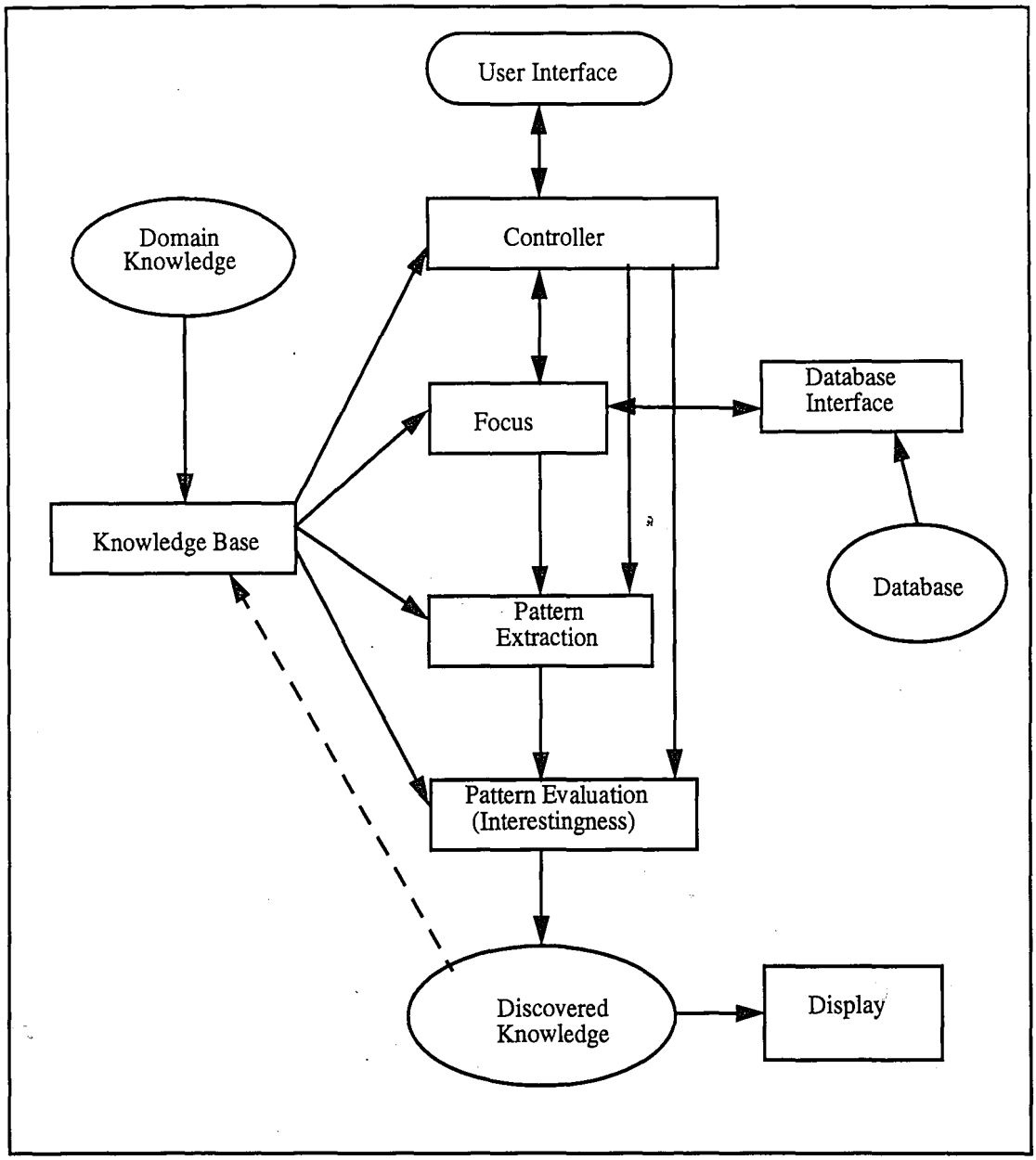

Figure 1: General model of a KDD/DM system.

2. Discovered Knowledge and Interestingness in KDD/DM

Before discussing the main topics of this paper, a prerequisite background on domain knowledge, measures of interestingness, and system autonomy as they apply to KDD/DM systems is in order, since the techniques discussed in this paper build upon prior research done in these areas.

Paramount in KDD/DM is the value of the discovered knowledge. In KDD/DM, this value can be measured in many ways, and may differ from user to user, but to summarize, we can state that the value of discovered knowledge is based upon the following criteria:

1. Certainty: the degree of certainty with which a given piece of knowledge is conveyed allows the user to place a degree of faith in that data. Without a reasonable degree of certainty, the discoveries cannot be considered knowledge. Typically, statistical significance (confidence factors, etc.) are used to determine degrees of certainty.
2. Efficiency: the KDD/DM process itself should produce tangible results within a reasonable time period. Part of the value of a KDD/DM system is that it can be used for decision support, market analysis, etc. A KDD/DM process that does not provide results in a timely fashion defeats the purpose of the system in the first place. One measure of efficiency is that an algorithm is considered efficient if the run time and space used are a polynomial function of low degree of the input length [1].
3. Interesting: the discovered knowledge, and indeed the entire KDD/DM process, are not useful if the end result is not considered of interest to the user. Some background concepts for what constitutes an interesting discovery is based upon several things:
    - Non-trivial: a discovery is not interesting if, for example, with a very small amount of effort, the knowledge could have been discovered by the user. An example would be performing a simple summary of the values of a given attribute on a database relation. Simple arithmetic computation does not necessarily equate to knowledge discovery; albeit, arithmetic computations are used at various phases through-out the KDD/DM process.

- Novel: the discovery is interesting if it is outside the scope of either the user and/or the systems' knowledge. One simple, but effective, measure of whether something is interesting or not is the "unexpectedness" of the discovery [2]; i.e., a discovery is also interesting to the user if it is surprising. This particular measure of interestingness falls in the category of a *subjective* measure.
- Utility: the discovered knowledge is interesting if the user can do something with this discovery. This can be referred to as the "actionability" of the discovered knowledge, which also falls into the category of a subjective measure of interestingness.
- Relevance: discovered knowledge can also be uninteresting if it has no significance to the problem domain currently being searched. For example, discovered knowledge about physician's home addresses while searching a medical database for disease prevention may in fact be discovered knowledge, but is not relevant to the current problem domain. Typically, relevancy falls into the category of an *objective* measure of interestingness. [2][3][4][6][8]

The subject of interestingness is one of the most difficult areas of KDD/DM, primarily due to it's subjective nature. A user's frame of reference, expected outcomes, domain knowledge, all impact what the user interprets as interesting knowledge. However, the interestingness of discovered knowledge is one of the primary factors for determining the value of the discoveries; regardless of how statistically significant and fast a KDD/DM system produces discoveries, the value ultimately relies upon if the user finds the information worthwhile, and/or can take action with or upon the discovered knowledge.

2.1 Autonomy, Bias, and Domain Knowledge

Related to interestingness are the autonomy of the KDD/DM process, bias, and domain knowledge. We will provide definitions of these as they relate to KDD/DM:

- Domain Knowledge: we define domain knowledge as supplementary information supplied by a domain expert to the KDD/DM system with the explicit intention of focusing various portions of the KDD/DM process for the purposes of providing greater efficiency and increased relevancy from the KDD/DM process. Domain knowledge includes the underlying data dictionary (meta-data typically maintained by the DBMS component of the KDD/DM system), domain expert-supplied analysis objectives, restrictions, etc. Domain knowledge may also be supplied as feed-back of interesting discoveries to the KDD/DM knowledge base. Domain knowledge is also referred to as *background knowledge.*
- Bias: related to domain knowledge, but closer to non-domain expert criteria supplied by the end user as input to the KDD/DM system. May also serve to focus portions of the KDD/DM process. An example of a bias would be for the end user of the system to force the KDD/DM process to sample data, rather than to use all data.[1]
- Autonomy: for this paper, we define KDD/DM autonomy as the degree of reliance on user interaction via input, bias, and domain knowledge, to guide the KDD/DM process. A completely autonomous KDD/DM system would not use either bias or domain knowledge; a semi-autonomous KDD/DM system would incorporate some degree of human intervention into the KDD/DM process, and a non-autonomous KDD/DM system would be reliant at all times for the end user/domain expert to direct the system for every step of the process.

Some typical examples of domain knowledge would be:

1. Attribute value constraints: all students in the computer science curriculum, all salaries less than 50000.
2. Attribute list or category constraints: include only the profits and losses attributes.
3. Data set domain constraints: include only the data from relations T1, T2, and T3.
4. Combinations of (1) through (3) above.

1. For the purposes of this paper, we will view bias and domain knowledge collectively under the same heading of domain knowledge and/or background knowledge.

5. Analysis objectives; include only positive examples and rules (inverse; include only negative examples and rules).

6. Applying abstractions or generalization hierarchies; search the abstract concept High_Risk_Customer, or apply the knowledge of concept A is-a B and B is-a C.

One of the critical issues is the relationship between the use of domain knowledge, and system autonomy. Domain knowledge, and it's use in KDD/DM, focuses the discovery and reduces the search space, allowing for a more efficient discovery process, as well as a higher probability for relevant discoveries. However, this exclusion of portions of the search space, however it is accomplished, may lead to missed discoveries. Conversely, not using domain knowledge results in a larger search space and thusly a less efficient discovery session, and may produce discoveries that are not relevant and uninteresting. In addition to the above, the use of domain knowledge is considered controversial in the KDD/DM community. The consensus for those who favor little or no use of domain knowledge is that to use domain knowledge is to taint the process ("telling the KDD/DM system what to look for and where to look for it"), and goes against the basic premise of knowledge discovery in the first place.

We can summarize some of the issues surrounding domain knowledge and system autonomy. Systems with a high degree of autonomy (little or no reliance on human guidance) exhibit:

- Greater potential for discovering unexpected knowledge.
- Greater potential for discovering non-relevant knowledge.
- Higher degrees of versatility; i.e., not rooted to specific search paths, algorithms, or filter criteria.
- Lower degree of efficiency, due to a large(r) search space.

Systems with low degrees of autonomy (high reliance on human input) exhibit:

- Greater potential to overlook and exclude relevant discoveries.
- Greater potential for discovering relevant knowledge.
- Lower degrees of versatility.

- Higher degree of efficiency, due to a restricted search space.

2.2 Relationship Between Autonomy, Domain Knowledge, and Interestingness

There is a close relationship, as described above, between KDD/DM system autonomy, domain knowledge, and interestingness of discovered knowledge. More specifically, domain knowledge is the binding factor between these three concepts. Since domain knowledge and the use of it in KDD/DM is to bias and/or focus the search and discovery session, the amount of it's use is proportional to the relevancy and interestingness of the patterns and discovered knowledge and efficiency of the discovery session, and inversely proportional to volume of patterns and discovered knowledge [1][3][4][5][6]. Figures 2 and 3 depict this relationship graphically.

Intuitively, this relationship between domain knowledge and volume of discovered patterns and knowledge is obvious. If, for example, a set of n relations exists, the total cartesian product of those relations is proportional to the number of tuples in those relations and the total number of relations. Since KDD/DM derives patterns from the data stored in relations (or, objects), the total amount of possible patterns to be derived from a given set of relations will be larger than the total amount of possible patterns derived from a proper subset of those same relations by simple multiplication. The purpose of domain knowledge is to focus the search on some subset of the overall data set derivable from the set of database relations or objects, thereby reducing the data set, and reducing the potential search space.

A secondary consideration is that the use of domain knowledge can reduce the degree of certainty of the results, if statistical sampling methods are employed as part of the domain knowledge. Thusly, the use of domain knowledge will reduce the search space and the total number of possible patterns and possible discoveries. However, the reduced search space also implies a more efficient search than if the total possible set of relations and data were considered. We have defined a highly autonomous KDD/DM system as a system that uses little or no domain knowledge. Thusly, a highly autonomous system has a tendency towards producing a larger volume of patterns and discovered knowledge, but exhibits a tendency towards inefficiency, due to the potentially large(r) search space for any given KDD/DM session.

The relationship between domain knowledge and interestingness is not as intuitive. Domain knowledge can focus the search, and provide a higher degree of relevancy, which is a primary factor in the measure of the interestingness of a discovery. However, relevancy alone does not constitute an interesting discovery. Relevant discoveries may or may not be interesting to the user, depending upon the user's frame of reference for a given KDD/DM session. Further, the use of a domain expert's domain knowledge may not embody a specific user's interests. For example, a typical technique for codifying knowledge in a database is to form a rule in the knowledge base of the type: IF <condition> THEN <result>. Assuming we have a "student" database for some university, an example of IF-THEN domain knowledge might be: "IF (students = GRADUATE) THEN (yearsInCollege > 4)". Domain knowledge of this type is generally used for classification/characterization [7], where classification knowledge, or rules, are used to discriminate one class of data, concepts, etc., from another, and characterization rules, or knowledge are used to group related data, concepts, etc., into one class. Some KDD/DM techniques derive this type of knowledge automatically, such as the induction techniques developed by Cai [3], et al. However, it may also be defined by a domain expert, and used as domain knowledge to focus a KDD/DMM session. The problem with this type of domain knowledge (using the student IF-THEN example), is that while it may focus the search, the results may not only be uninteresting to the user, but in this case, may also be tautological. The use of this type of explicit domain knowledge, while achieving higher degrees of efficiency and relevancy to the KDD/DM process, serves to focus and bias the KDD/DM process, but the author of this paper argues that it does not imply that the results derived from using domain knowledge will be interesting, as defined by the user of the system.

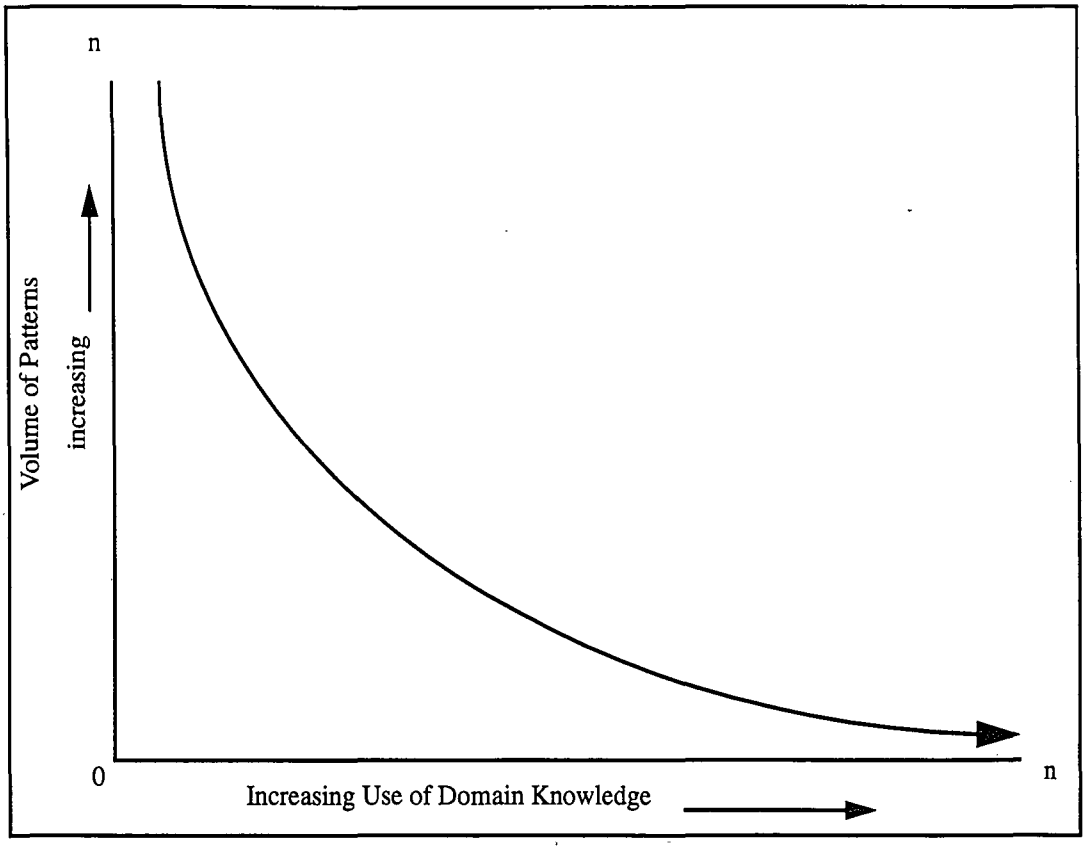

Figure 2: Relationship between domain knowledge use vs. volume of discoveries

There are, however, some KDD/DM systems that infer a domain-dependent measure of interestingness on discoveries, such as KEFIR[2][4]. In KEFIR, a domain expert has defined a subjective measure of interestingness based upon the benefits accrued from certain measures. Unfortunately, KEFIR's system bases it's measures of interestingness upon domain-dependent values, and does not fit into a domain-independent context.This type of domain-dependent interestingness does not embody all users' interests, in reality, it embodies only the interests of the domain expert. In this paper, we seek a more domain-independent, user-oriented method of determining interestingness.

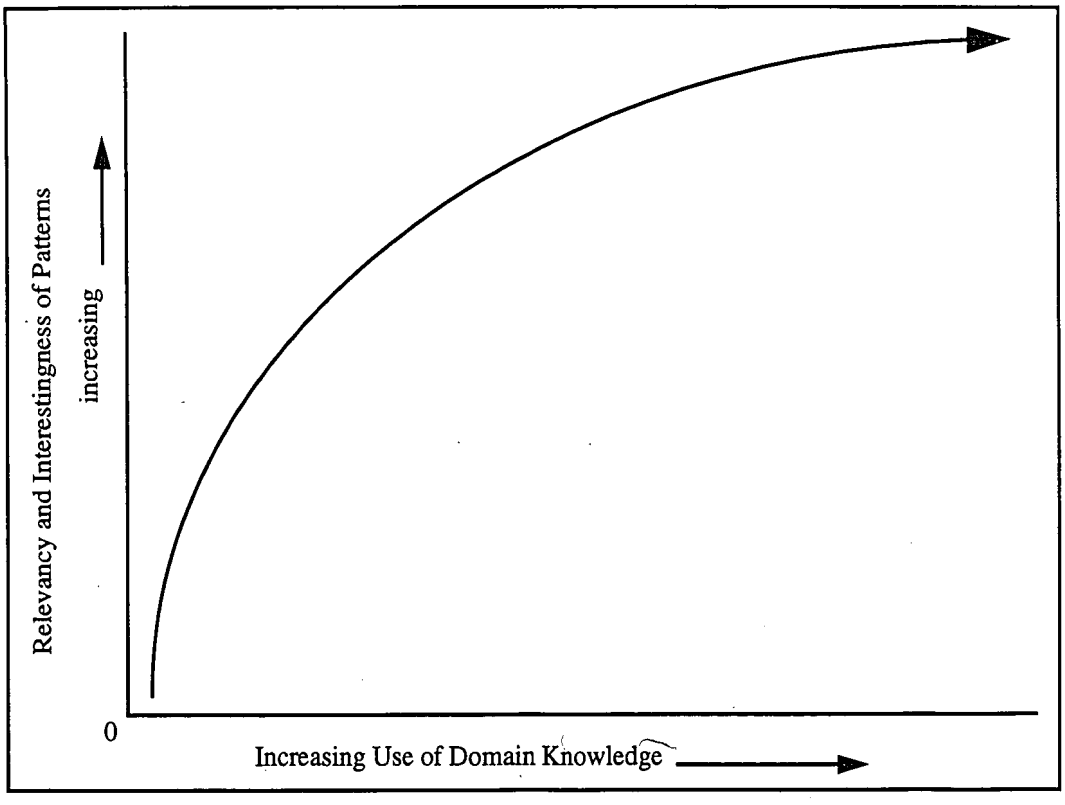

Figure 3: Relationship between domain knowledge use vs. relevancy and interestingness of patterns.

2.2.1 Measures of Interestingness

In [2], Silberschatz and Tuzhilin propose a domain-independent classification for measures of interestingness for KDD/DM; specifically, subject and objective interestingness, and *belief systems*. (Figure 4 depicts this classification) In objective terms, the interestingness of a pattern is based upon quantitative measures and statistical significance; i.e., a pattern or rule is measured in terms of its structure and underlying data. [2][8][9][10]. Specific objective measures of interestingness are the "information content, certainty factor, and strength" [2][8][9][10].

However, objective measures of interestingness do not capture all of the complexities of knowledge discovery; in particular, objective measures of interestingness based upon statistical significance. In fact, as Matheus[11], et al., point out, the exact opposite could be considered more interesting to any given user. For example, a small percentage increase can be more interesting than a larger percentage increase, even

though the statistical significance of the larger increase may be higher.[1]

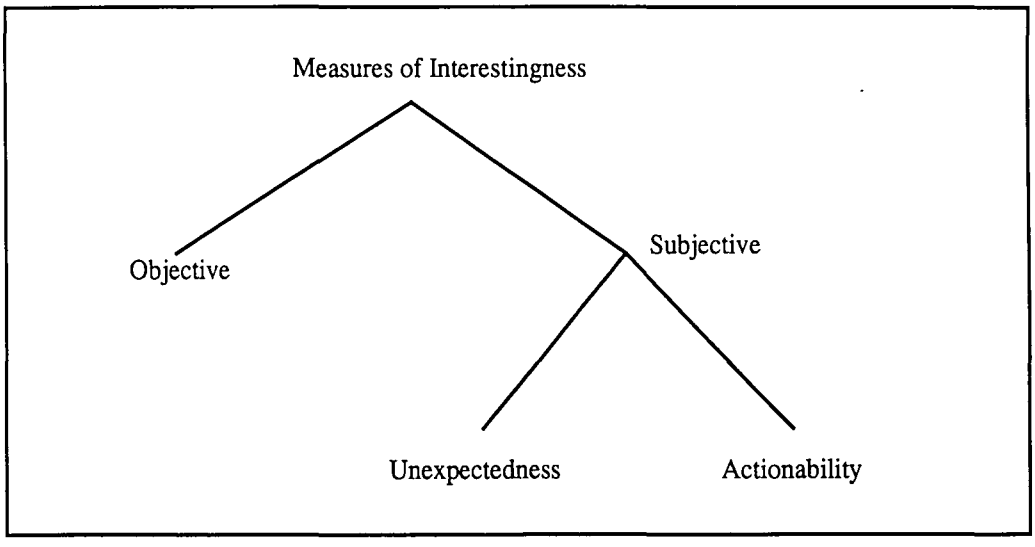

Figure 4: Classification of Interestingness Measures [2].

This type of semantic complexity is difficult, if not impossible, to be measured in objective terms. Beyond the objective measures of interestingness, subjective measures are needed to help define the interestingness of discoveries. Subjective measures of interestingness do not depend only on the structure of the rule and the data used for the discovery, but also on the user who examines the discoveries [2]. Subjective measures can be used to recognize that a pattern that is of interest to one user may be of no interest to another [2]. The notion of subjective interestingness is of extreme importance in KDD/DM, since one of the main problems with KDD/DM is the propensity of these types of systems to produce a "glut" of patterns; methods that minimize this volume of patterns and produce only those that are of subjective interest to the given user is very desirable. Given the premise of subjective interestingness, several issues are raised:

- What constitutes subjective interestingness.

1. The specific example pointed out in [11] is that a "5% increase of sales in the western region may be more interesting than a 50% increase in sales in the easter region", for a variety of reasons, such as: the western region has a larger sales volume and thusly the 5% increase translates to a larger income growth, etc.

- Can subjective interestingness be measured.
- Do methods exists to determine subjective interestingness on a per user basis.
- Do methods exists to determine subjective interestingness on a per user and domain-independent manner.

In [2], Silberschatz and Tuzhilin identify two primary reasons as to why a given pattern, rule, or discovery is interesting, from the subjective (user-oriented) viewpoint:

1. *Unexpectedness:* a pattern is interesting if it is "surprising" to the user.
2. *Actionability:* a pattern is interesting if the user can do something with it to his or her advantage.

We will examine these further, and look at their relationships to one another, relative to what is referred to as a *belief system.*

Intuitively, actionability is an important subjective measure of interestingness; from any given set of data, we are most likely to be interested in that which we can take some specific action with, or upon. This is one of the most critical reasons behind KDD/DM systems. KDD/DM began out of a desire to take action upon volumes of data. Unexpectedness is equally important[1], because unexpected data contradicts our expectations. Silberschatz and Tuzhilin in [2] argue that unexpectedness and actionability are independent, however, based upon the following:

1. Patterns can be both actionable and unexpected [2].
2. Patterns can be actionable and expected [2].
3. Patterns can be unexpected and un-actionable [2].

Given the above, it is reasonable to assume that while both actionability and unexpectedness are good measures of interestingness, we cannot assume that any type of "interestingness filter", or algorithm, can simply choose one or the other as it's interestingness criteria. Nor may we assume that an interestingness filter can find an actionable pattern and expect that the pattern is also unexpected, and vice-versa. Ideally,

1. Silberschatz and Tuzhilin in [11] use the statement "how many times have we exclaimed, ' Oh, this is interesting' when we discover something unexpected".

user-oriented interestingness from a KDD/DM perspective, should be based upon both actionability and unexpectedness.

2.2.2 Unexpectedness and Belief Systems

Silberschatz and Tuzhilin [2] discuss the idea of belief systems in the context of unexpectedness. A pattern or discovery from a KDD/DM system is unexpected and interesting to a user when it contradicts the users' expectations. This set of expectations forms what is referred to as a *belief system.* In the realm of artificial intelligence, there are two primary theories to beliefs and belief revision. One approach argues that we either believe in something or we do not. If a new belief is considered, it is either (a) added to our existing set of beliefs in the case where it does not contradict them, or (b) previous beliefs must be removed to accommodate the new belief in the event that it does contradict them. The second approach assumes that beliefs can be partial beliefs, in that each person assigns a measure, or degree, of confidence to each belief [2]. Using this approach, there are two primary schools of thought to the assigning of this confidence factor, the Dempster-Shafer theory of evidence and Bayesian approaches. Using the Dempster-Shafer approach, the degree of belief is based upon previous evidence, or case histories. The Bayesian approach assigns a belief function to each belief.

Silberschatz and Tuzhilin further classify beliefs relative to unexpectedness as either *soft* or *hard* beliefs. The definitions are as follows:

- Soft beliefs: beliefs that the user is willing to change as new patterns and discoveries provide the user with the new evidence [2].
- Hard beliefs: constraints that cannot be changed with new evidence. If the new evidence contradicts these beliefs, then errors have been made in acquiring this new evidence. No degree of certainty is assigned to hard beliefs, since they cannot be changed [2].

An important point is that for both hard and soft beliefs, Silberschatz and Tuzhilin emphasize that these are subjective, and vary from one user to another.

2.2.3 Current State of Interestingness Evaluation in KDD/DM

In the literature, interestingness evaluation and filtering tends to be objective and rule-based; i.e., heuristics based upon domain-specific criteria. Typically, statistical evaluation is used, or built into the domain knowledge, and interestingness is weighted based upon numeric evaluation. In terms of subjective interestingness, the KEFIR [2][14] system made some progress in terms of subjective interestingness. KEFIR measured subjective interestingness by evaluating discoveries[1], and then weighing the estimated benefits of acting upon those discoveries. The corrective action that brought about the largest benefit for any given discovery deemed that discovery as the most interesting. While a step towards subjective interestingness, KEFIR's subjective interestingness evaluation was domain-dependent, and heavily domain-expert based. KEFIR based it's notions of benefits accrued against key health care indicators, not against what a general user might find interesting.

Typically, in objective rule-based KDD/DM systems, this "interestingness filter" is developed by a domain expert, stored as a knowledge base, and incorporated as domain knowledge. The interestingness filter is used to delimit the patterns, rules, discoveries, etc., and determines which, if any, should be displayed to the user. An alternative (and more desirable) approach, is to use the interestingness filter along with the discovery process, focusing the search either before or during the actual data searching, delimiting and focusing the search prior to displaying the discoveries to the user. This second approach searches only for interesting patterns, rather than finding all patterns, and then displaying to the user.

While extremely useful for focusing the search, the primary failing of domain expert-supplied domain knowledge for interestingness evaluation is its lack of evaluating interestingness from the *user's* objective and subjective interests. However, two interesting approaches for embodying the user's interests as the domain knowledge were taken by Dhar and Tuzhilin [10] and Yoon and Kerschberg [15]. Dhar and Tuzhilin take the approach that interesting patterns are expressed not only in database value terms, but also in user-defined terms. They present an approach called "Abstract-Driven Pattern Discovery" [10], which is based upon the following criteria:

---

1. Piatetsky-Shapiro refers to the most significant discoveries as "key findings" [2].

1. A vocabulary of user-defined predicates, where the user-defined predicate is either a database relation, a previously existing user-defined predicate, or a concept (condition) involving attributes from database relations.
2. A classification hierarchy that groups the user-defined predicates into a partial ordering.
3. User-defined abstraction functions, which map the domain values of a given attribute into some other domain. Abstraction functions can be grouped into abstraction hierarchies.

Based upon these three concepts, patterns are discovered based upon the user-defined predicates, the list of abstraction functions the pattern should contain, and a user-specified aggregation function (summarization, classification, etc.). This approach by Dhar and Tuzhilin is novel, from the perspective that the users interests are embodied in the search, *a priori,* since the user has explicitly defined the predicates, and has specified which predicates should be used in a given discovery session.

A second approach was taken by Yoon and Kerschberg [15], primarily targeted at characterizing database exceptions. Like Dhar and Tuzhilin in [10], Yoon and Kerschberg seek to "discover rules which match and support a user's interests..." [15]. They argue that "... users' interests, intentions, insights, or background knowledge are conveyed by means of a query to support knowledge discovery, which delimits the learning space. Unless a users's interests or intentions are used to discover rules, the discovered rules may not be useful for his (her) applications" [15]. They argue further that not only does a query embody a user's interests, but the answer does, as well. To implement this premise, Yoon and Kerschberg use a standard SQL query to access the database.

Both the Dhar and Tuzhilin and Yoon and Kerschberg approaches are unique from the standpoint that they attempt to embody the user's interests up-front in the KDD/DM process by allowing the user to indicate what is interesting to the user either as part of the search criteria, or as the overall criteria of the search itself. However, some potential issues with these approaches:

- In Yoon and Kerschberg, straight reliance on an SQL query may not capture all of the necessary relationships of the underlying database. For example, selecting attributes from relation R only may overlook important and necessary relationships from other relations R1, R2, etc. This has the affect of potentially missing interesting discoveries and patterns.
- In both Yoon and Kerschberg and Dhar and Tuzhilin, the assumption is made that the user understands the underlying database schema, in order to effectively construct correct SQL statements and/or correct user-defined predicates, respectively.
- Yoon and Kerschberg assume the user understands and is proficient in SQL.

The above issues not withstanding, the approaches of Yoon and Kerschberg and Dhar and Tuzhilin represent the basis for this paper; i.e., embodying the user's interests either before or during the search, using as little domain knowledge as possible other than what was provided by the user so as to concentrate on the user's interests, yet expand the search space so as not to overlook potential discoveries, and codify these discoveries into a knowledge base that not only represents the users' interests, but may also be used to feedback into future discovery sessions.

3. Semi-Autonomous Knowledge Discovery

With consideration to the prior issues and discussion, the author of this paper proposes a framework for semi-autonomous knowledge discovery that emphasizes domain-independent, user-oriented interestingness techniques to maintain a reasonable degree of efficiency by restricting the problem space, yet expanding this same problem space so as not to miss potentially interesting and important discoveries. To do this, a technique of "reference tracing" is employed, which attempts to satisfy these goals.

With regards to system autonomy, the author of this paper argues that a system cannot be completely autonomous (independent of user domain knowledge) and embody the users' interests, other than if the user has built up a sufficient knowledge base that is used to drive subsequent discovery sessions. Conversely, a KDD/DM system that relies entirely on the user to guide the system (no autonomy) has a high potential for overlooking interesting discoveries, since the user may direct the system too much towards "expected" results. Ideally, there exists a trade-off between embodying the users interests, yet being "autonomous enough" so as to provide for a reasonably autonomous discovery session. The research of this paper argues for a semi-autonomous KDD/DM process, iterative and interactive in nature. Semi-autonomous KDD/DM is defined as follows for the purposes of this paper:

- Semi-autonomous knowledge discovery is an iterative and interactive process by which the user specifies an initial database object domain, from which other related database objects and data will be derived. The initial database object domain may consist of user-specified database relations, views, attributes, and/or user-defined abstractions or concepts; i.e., anything that is part of the extensional and intensional database. The initial database object set is used to focus the search and maintain relevancy in terms of discovered patterns and knowledge, based upon the user's interests as they are embodied in the initial database object domain.

3.1 Semi-Autonomous Knowledge Discovery and Reference Tracing

Semi-autonomous knowledge discovery may or may not be an iterative process; it is anticipated that the most benefits will be derived from an iterative, interactive, incremental learning process. However, as the knowledge base grows, the amount of user intervention and iteration may be reduced. The definition of

semi-autonomous knowledge discovery indicates that related database objects and data will be derived from the initial set. The method employed to do this is referred to as *reference tracing*. Reference tracing is simply a means of using the minimal set of a domain experts knowledge implicitly, via the data model and underlying relationships encoded by the data modeler into the database design, to aid in the KDD/DM process. In this manner, a minimal amount of user-oriented input is used to focus the discovery session on only those objects of interest to the user, yet expand the initial domain, autonomously, so as to not overlook related and interesting concepts. For the purposes of this paper, we define reference tracing as follows:

- Reference tracing is the process by which a subset of strongly-related relations are derived from the extensional database from an initial set of relations that are also part of the extensional database via their referential integrity constraints and attributes of the initial set of relations.

More simply stated, reference tracing attempts to expand the initial, user-supplied database object set beyond this set by means of relationships between the initial database object set and other database objects. In this manner, reference tracing seeks to maintain relevancy by only using the user-oriented objects of interest, yet expand this set to related objects so as not to miss potentially interesting and important discoveries.

3.2 Semi-Autonomous KDD/DM and Reference Tracing Concepts and Issues

We will look at the issues behind semi-autonomous knowledge discovery and reference tracing, and how this framework provides a means for determining relevancy and interestingness of database data and patterns.

3.2.1 Reference Tracing and Implicit Background Knowledge

Reference tracing uses background knowledge to derive the final database object set from the initial user-supplied database object set. Typically, this comes in the form of analyzing primary key/foreign key relationships to determine "strong" relationships, and examining other, non-key attributes of the initial database object set. The benefits of this approach are:

- The final database object set will be related to the initial database object set; thusly, all discovered patterns will be relevant to the user's interest, since they will use either wholly or in part, the initial database object set. In this manner, the user's interests are embodied in the input as well as the output.
- By using the implicit background knowledge, rather than explicit domain expert-supplied rules to focus a KDD/DM session, a minimal amount of non-user oriented bias is injected into the KDD/DM process. Further, the background knowledge employed is used to expand the search beyond the original database object set, not focus the search and miss potentially important discoveries; in this manner, a higher degree of relevancy is achieved.
- Since the user focuses the search on a subset of database objects that only the user determines are interesting for this session, the problem space is reduced, and the KDD/DM process will be more efficient. To further enhance the overall efficiency, the user-supplied objects focus the search prior to the search being executed itself, so that the problem domain is reduced a priori.
- While the user supplies the initial database object set, this set is expanded based upon relationships to this original object set, expanding the problem domain to the final, minimal database object set. Thusly, the potential for missing potentially relevant discoveries is minimized relative to only using the initial database input object set.
- It is user-oriented in that the user supplies all of the initial objects of interest.
- It is domain independent in that no specific data domain-dependent, domain expert-supplied rules are used to focus the KDD/DM session. No explicit knowledge is used to focus the search other than the user's. Additionally, the focus can vary from session to session, and intra-session.
- By determining relationships among the database objects that the user may not have known existed, a method for the display of the discovered patterns in order of their relationship "strength" is achieved.

- By expanding the initial object set using the constraints specified by the implicit domain knowledge of a domain expert as embodied in the database design, we can achieve the effect and results of a fully autonomous KDD/DM process, yet in a controlled manner. This ultimately leads to potentially important knowledge discovery beyond what was within the initial database object set.

It is recognized in the literature ([1],[3], [4]) that the use of this type of background knowledge (i.e., primary key/foreign key relationships and database attributes to focus and/or expand the search) is the least controversial, in that it does not bias the KDD/DM process.

3.2.2 Iteration and Interaction to Achieve Semi-Autonomous Knowledge Discovery

The author of this paper argues for an iterative knowledge discovery process, and the framework described in this paper considers iteration and interaction as part of a semi-autonomous process. This, in part, satisfies several goals. One of the problems of the Dhar and Tuzhilin [10] and Yoon and Kerschberg [15] approaches is that the user must supply a set of user-defined predicates or be knowledgable of SQL, respectively. Each of these requires an understanding of the underlying database schema, and in the case of the latter, the SQL language, as well. The semi-autonomous knowledge discovery framework proposed in this paper does not require the user to supply either; rather, it only requires that the user be knowledgeable of the general domain of the database; i.e., the database is a "Personnel" database, a "Financial" database, etc. And, while the user must supply an initial database object set, which can be composed of Dhar and Tuzhilin-type concepts and abstractions, it is possible for the user to be completely unfamiliar with the underlying database. However, this implies iteration and interaction. The following issues are identified for the advocacy of iteration and interaction with semi-autonomous knowledge discovery:

- Initial object set introduction: the user may not know, or be familiar with the database domain, and may thusly need to be prompted with an initial set of "potential" database objects. This initial set would be the set of all database relations, perhaps, or could be filtered in some way (perhaps based upon database-level user grants). A simple query of the database data dictionary would be used to derive this initial set, which would then be displayed to the user, who would

choose from this set to create the initial, user-supplied database object set. The user-supplied database object set would then be used as the initial set for reference tracing. However, this implies at least one iteration and user interaction.

- Intermediate and final database object sets: As the KDD/DM process progresses from the initial user-supplied database object set, the user may wish to add or remove to the intermediate and/or final database object sets used for the discoveries. This is best handled by iteration and user interaction.
- Addition of additional bias: During the course of KDD/DM the user may wish to further bias the results.

To give examples of the above, suppose a user wishes to find out what objects they may include in a given KDD/DM session. In the framework discussed in this paper, the user would request the initial database object set, which, for the sake of this example, consists of relations R1, R2, and R3. The user selects R1 and R3 to take part in this particular KDD/DM session, and submits these database objects as the initial user input database object set. Reference tracing is executed against this object set, and assume that relations R4, R10, and R20 are found to be strongly related to the initial input database object set of R1 and R3. The additional relations found via reference tracing are displayed to the user, and the user either selects all, a subset or none of these additional database objects. At this point, the user may specify additional database objects, or choose to execute the KDD/DM session against only the original R1 and R3. This illustrates the benefits and necessity of iteration and interaction to achieve the desired results of semi-autonomous knowledge discovery. Note that at no time in the above are database objects selected for use in a given KDD/DM session other than what was specified by the user, or derived via reference tracing.

3.2.3 Derived Database Objects and Relationship Strength

Reference tracing produces what is referred to as the *derived database object set.* The derived object set will either be a temporary, intermediate set, or the final database object set to be used in the KDD/DM session. We define the derived object set as follows:

- The derived database object set is the subset of database objects that are part of either the intensional or extensional database used for the given KDD/DM session. The derived database object set is explicitly derived through reference tracing.

The derived database object set is used to expand the initial user input database object set, to look for relevant and interesting discoveries. Since the derived database object set is based upon the initial database object set, discoveries will be relevant, with respect to the initial user input database object set. The derived set is only used in a KDD/DM session if the user selects some or all of the derived objects during the iterative phases of KDD/DM.

Derived objects have a "strength" relative to how closely related they are to the initial user input database object set. The derived object sets have decreasing strength as their relationships with the initial user input object set diminishes. Further, derived objects' relationships are only classified in degrees of "strength"; derived objects relationships cannot be considered to be "exact", since an "exact" relationships implies one that is always correct. Derived objects are based upon the implicit knowledge of the underlying database schema; relying on this type of background knowledge implies a strong relationship, which in turns admits the possibility for error in the database design, and specifically, the database relationships which are used for reference tracing. We define strong relationships as follows:

- A strong relationship for a derived database object is a relationship that is almost always correct [8].

The framework discussed in this paper employs the underlying implicit knowledge provided by the logical data model, and hence, that of the logical data modeler who designed the logical database. Thusly, the underlying schema embodies the domain experts viewpoints, experience, case histories of known, partially known, and perhaps unknown outcomes. Additionally, this includes incorrect and irrelevant knowledge [18]. If we assume that there is a probability that the logical database design is not an exact database design, then the best we may assume is that the relationships are strong relationships [8][18].

Based upon the premise that there are no exact relationships due to the implicit use of background knowl-

edge, then we classify all derived database objects as related to the original set in terms of their strength. A simple classification scheme for derived database objects is as follows, in order of decreasing strength:

1. All derived database objects that have primary key/foreign key relationships with the initial user input database object set are considered to have a strength of 1, or "strength1". This is the highest (strongest) level of relationship between database objects. These sets of database objects are directly related via the primary key/foreign key relationships. The primary key/foreign key relationship is considered a strong relationship in conceptual and logical data models, and the strength of that relationship is carried forward to this framework.
2. All derived database objects that have a primary key/foreign key relationship with the derived database objects of strength 1 are considered to have a strength of 2. All derived database objects that have primary key/foreign key relationships with the derived database objects of strength 2 are considered to have a strength of 3, and so on.

This classification scheme is recursive. Note in step (2), that all database objects that are derived from a set of database objects of strength n have strength n+1. Figure 5 depicts this classification scheme:

It is observed that reference tracing produces an n-ary tree of arbitrary height. Note that it is possible for reference tracing to produce a tree that includes the entire database object set. This defeats part of the goal of semi-autonomous reference tracing; i.e., limiting the problem search space. However, since the framework provided here for semi-autonomous knowledge discovery advocates an iterative, interactive session, it is left to the discretion of the user for any given KDD/DM session to selectively include/exclude both initial user input as well as derived database objects.

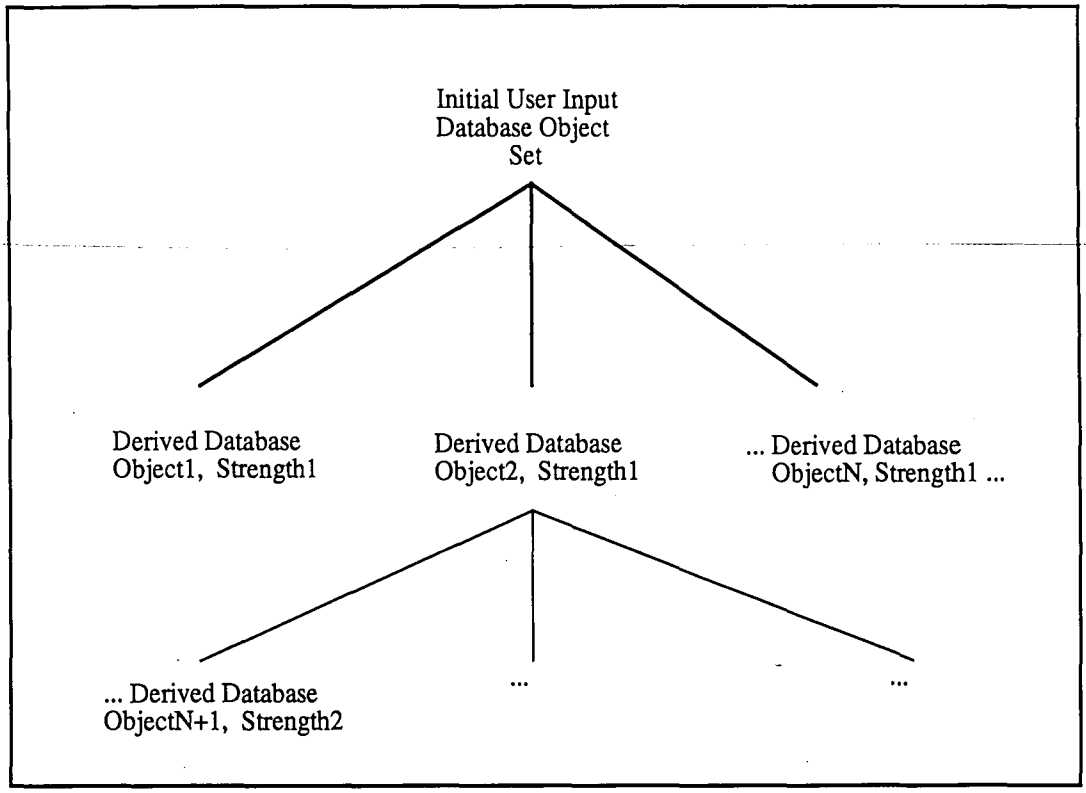

Figure 5: Classification Scheme for Strengths of Derived Database Objects

3.2.4 Subjective Interestingness Under Semi-Autonomous KDD/DM and Reference Tracing

Referring back to Silberschatz and Tuzhilin in [2], they describe measures of subjective interestingness in terms of unexpectedness and actionability. Semi-autonomous knowledge discovery with reference tracing identifies several key issues with respect to subjective interestingness:

1. Derived database objects are either unexpected or unknown to the user, or, the user was aware of the relationships and chose to ignore making the related database objects part of the initial database object set. For the purposes of this paper, we assume that the user was not aware of either the derived objects or their relationships. If this turns out to be an incorrect assumption, the iterative nature of this framework would indicate that the user would either confirm or reject the derived database objects during one or more iterations before creating the final database object set.

2. The lesser the strength of the relationship to the initial user input database object set, the higher the probability for unexpectedness.

3. The strength classification of the derived database objects relative to the initial user input database object set permits for an ordering of presented patterns and discoveries in terms of their strengths.

We examine these on an intuitive level. Starting with issue (1), it follows with the goal behind this framework for semi-autonomous KDD/DM; given that the user doesn't understand or perhaps is just not aware of, the underlying database design, but may be familiar with several high-level concepts or relations[1], the initial user input database object set may not be inclusive of all of the related database objects. For the purposes of this paper and discussions, any database object derived from this initial set is considered unexpected. Issue (2) follows from issue (1), in that the more un-related the derived database object is, the higher the probability that patterns and discoveries that include this object will be increasingly unexpected to the user. In other words, we may be expecting relationships between the initial user input database object set and the derived objects of strength 1, but the lesser the strength of the relationship, the more likely the user is to be surprised that the objects are related. Issue (3) is a general statement that we can view the strength classification as a hierarchy for ordering of patterns and discoveries. This is a small but important fact, and aids in identifying specific patterns of interest for the user.

It is precisely these above three issues that are core in the relationship between interesting discoveries and reference tracing. Reference tracing derives database objects that are related to the initial user's interests, and rank orders them in terms of the strengths of the relationship to this initial set. It follows that the most un-related database objects derived from reference tracing contain the most unexpected, and hence, the most interesting patterns. Thusly, the author of this paper posits the following:

- The interestingness of patterns and discoveries from derived objects is inversely proportional to the strength of the derived database objects they involve.

1. This is typical in industry, where most database users are aware that a database may contain specific relations, but may not be aware of the relationships.

More simply stated, the weaker the relationship between the initial user input database object set and the derived database objects that are part of the patterns and discoveries, the more surprising, and thusly interesting in subjective terms, are those patterns and discoveries. Further, the order in which the discoveries are displayed to the user should be in inverse order of the strength, as these comprise the most subjectively interesting discoveries to the user.

A simple example (detailed example to follow in later sections) shows this process. Consider that a user has selected 3 relations, R1, R2, and R3 as the initial user input database object set. For this particular KDD/DM session, reference tracing is used, and it derives the following two sets of related database objects; strong2(R7,R9), and strong3(R21,R33), which indicate that relations R7 and R9 were derived with a strength of 2, and relations R21, and R33 were derived with a strength of 3. One of the typical pattern extraction methods is *dependency analysis* [3][11][19]. Dependency analysis looks for data dependencies; specifically, looking for dependencies that can show with a certain statistical probability that the value of one piece or collection of data can be used to predict another data value or collection. Going back to the example, suppose that a dependency analysis algorithm indicated that the following dependencies exist:

- D1 (dependency 1): indicates that a dependency exists between relations R1 and R7.
- D2: indicates that a dependency exists between relations R2 and R21.
- D3: indicates that a dependency exists between relations R3 and R9.
- D4: indicates that a dependency exists between relations R1 and R33.
- D5: indicates that a dependency exists between relations R9 and R33.
- D6: indicates that a dependency exists between relations R33 and R21.

The specific attributes and/or attribute groupings that constitute the dependency do not matter for this example. More importantly, the results of this dependency, using semi-autonomous KDD/DM and reference tracing have derived several database objects that have exhibited dependencies with the original set of database objects. Under the assumption presented in issue (1) of this section, all derived objects are

considered unexpected. We also assume, for the simplicity of this example, that no iteration has taken place; i.e., the user has chosen to accept all derived objects to take part in the pattern extraction of the KDD/DM session. The order of the discovered patterns would be displayed to the user as in the following order:

1. D2
2. D4
3. D1
4. D3
5. D6
6. D5

The rationale behind this ordering is that we assume the user was not aware of the relationship between the initial user input database object set (relations R1, R2, and R3), and the derived database objects R7, R9, R21, and R33. Therefore, these derived relations are unexpected, and any patterns extracted that involve data or attributes of these relations are considered to have a higher probability of unexpectedness. Since derived objects R21 and R33 have a lesser strength than derived objects R7 and R9, patterns extracted that involve relations R21 and R33 are considered to have a higher probability of unexpectedness than patterns involving relations R7 and R9. Thusly, the presentation of the patterns (in this case, the dependency analysis) is in the inverse order of the strength of the derived objects. In this manner, semi-autonomous reference tracing assists in the identification and presentation of unexpected, and thusly interesting, patterns.

Note, however, dependencies D5 and D6. These dependencies were extracted only from derived database objects; they do not have a direct relationship with the initial user input database object set. Similar to the ordering of the dependencies D1 through D4, patterns that have no database objects in common with the initial user input database objects set are displayed in inverse order of the strength of the derived database object set. However, these patterns that have no objects in common with the initial user set are displayed

after the patterns that do have objects that were part of the initial user set.

It is also important to note that while this example used dependency analysis as the discovery algorithm (pattern extraction) algorithm, this ordering would be imposed on all discovered patterns, under the framework described in this paper.

The expansion of this initial user input database object set is a very important concept. Some fundamental, intuitive, issues that are pertinent to the notion of expanding the initial database object set:

- System autonomy: if we allow only the KDD/DM system to discover patterns, then we receive a glut of potentially uninteresting, irrelevant patterns. By using semi-autonomous discovery, directed by an initial, user-specified set of database objects, we have the initial base to start from for our searches, that is user-oriented, and contains the objects and concepts that the user finds interesting.
- If we restrict the search to only the user-specified object set, we have the potential to overlook interesting patterns and relationships.
- By expanding the initial set via implicit domain knowledge, we can impose logical constraints so as not to revert back to a fully autonomous KDD/DM session.
- By expanding the search to derive related database objects, we emphasize unexpected discoveries, and thusly achieve a potentially higher degree of interestingness.
- By expanding the search beyond the initial user input database object set, we also approach, in a controlled manner, *the effect and results* of a fully autonomous knowledge discovery system and session.

Discussing this last point, one of the goals of KDD/DM is to "explore and discover" databases, finding implicit and interesting patterns and discoveries. This goal is fully realized in a completely automobiles KDD/DM system, with the negative impact of also discovering large amounts of uninteresting and irrelevant patterns. In a semi or fully non-autonomous system, the benefit of unrestricted search and unrestricted discovery is lost, with the positive impact being of more focused, efficient, relevant and

interesting results. In addition, using domain knowledge in any type of KDD/DM system, also reduces the potential benefits of unrestricted search. However, the above example illustrates how we can achieve discovering patterns that are directly related to our areas of interest, as well as indirectly related. Considering the dependencies D5 and D6, these patterns were discovered from an indirectly related set of database objects and data. This is directly equivalent to a fully autonomous system, yet while remaining focused on an initial set of objects and the efficiency and interestingness associated with this method. In this manner, semi-autonomous knowledge discovery with reference tracing approximates a fully autonomous knowledge discovery system, yet with a higher degree of efficiency, relevancy, and interestingness to the discovered patterns.

If we continue with this further on an intuitive level, and referring back to the example in this section, it is entirely possible that the dependencies D5 and D6 are completely uninteresting to the user. However, there is an equal possibility that they are interesting. This can be formulated by the following observation:

- Semi-autonomous knowledge discovery using reference tracing aids the user in an interactive, efficient discovery session of a given database, and may produce discoveries that are not related to the initial user's interests, yet have a high degree of unexpectedness, and are thusly interesting to the user, causing the user to re-assess their interests for any given KDD/DM session.

More simply stated, this is the equivalent of the user "interested and looking for one concept or set of concepts, and stumbled upon another equally or more interesting concept or set of concepts". This is the essence of knowledge discovery.

3.2.5 User Bias

The admission of user-supplied bias for additional focus of search is possible with this framework. The goal of user bias with respect to semi-autonomous knowledge discovery and reference tracing is to allow the derived database object set to continue to be derived, and apply the bias during as late a stage as possible (the actual pattern extraction phase) so as to maintain the aspects of semi-autonomy for knowledge discovery. It is important to draw a distinction between various types of bias:

1. Database object level bias: inclusion/exclusion of entire database objects; i.e., relations, views, etc. This can be part of the initial user input database object set, or during iterations after the initial input. In the simplest form, database object level bias is the initial user input database object set itself. Further iterations of the semi-autonomous process would either add to, or eliminate from, the initial and derived database object sets, prior to producing the final database object set.

2. Database object attribute level bias: inclusion/exclusion of specific attributes of database objects. For example, if the database object R1 includes attribute a1, the user could choose to exclude this particular attribute on this particular database object. Note that this has the potential result of eliminating database objects from the final database object set, if the attribute excluded happened to be a primary or foreign key. The framework discussed in this paper would ignore the attribute exclusion for the sake of reference tracing, but would exclude the attribute from any pattern extraction. This maintains the notion of semi-autonomous knowledge discovery with reference tracing, but still has the potential to result in missed discoveries.

3. Strength bias: limiting the height of the n-ary tree produced by reference tracing. For example, if the number of derived database objects returned is too large, the user could choose to "prune" the reference tracing tree. Note that this has the effect of missing potentially interesting discoveries.

4. Value bias: specific data constraints on specific attributes, or value constraints attribute or object-wide. This has no effect on reference tracing, and effects the actual pattern extraction process; i.e., inclusion/exclusion of specific text or data, data limits, etc., that would have the overall effect of reducing the data volume included in the pattern extraction phase of KDD/DM.

Database object level bias is simply part of the semi-autonomous KDD/DM process. and reference tracing. Recall that this framework provides for an iterative, interactive process. Part of that process is defining the initial input user database object set (which is a form of database object level bias), allowing reference tracing to occur, evaluating the potentially expanded database object set, and either adding to or subtracting from this set. This process iterates until the final database object set is determined. This is the

database object level bias.

Database object attribute level bias is similar to database object level bias, but permits the user to either add to or subtract from specific database objects' attribute list. This allows the user to focus on specific attributes of interest, but may exclude potentially interesting discoveries by removing attributes from the initial and derived database object sets. If the attributes being excluded include primary or foreign keys, these attributes will still be used for reference tracing, but will not be used during pattern extraction. Consider the following example: the user's initial database object set is comprised of database objects R1, R2, and R3. In addition, the user decides to eliminate the attributes a1, a2 from database objects R1 and R3, respectively. Suppose that attribute a1 is the primary key of object R1. Eliminating a1 could reduce the derived database object set. However, the reference tracing model discussed in this paper would continue as planned; i.e., use a1 for reference tracing purposes to derive any and all related database objects, but then exclude a1 from the pattern extraction phase. In this manner, reference tracing and semi-autonomous knowledge discovery are not compromised, but the user's desires to restrict the attribute domain are accommodated. This has the overall effect of reducing the search space and making the pattern extraction phase more efficient, at the downside of potentially missed discoveries.

Strength bias is simply a means for pruning the n-ary tree produced as a result of reference tracing. For example, if the initial user database object set results in reference tracing tree of height 10 (indicating relationships of strengths 1 through 10), the user may decide that this is simply too many database objects for a given session. During the iteration phase, the user may decide to prune the tree to, for example, a strength of 5, in which case all derived database objects of strength 6 through 10 would be excluded from the final database object set. Similar to object and attribute level bias, this has the potential for missed discoveries, but expedites the discovery process by reducing the problem search space.

Value bias has no effect on reference tracing, and limits the data volume used in the pattern extraction process. Similar to attribute bias, it reduces the overall problem search space, but may preclude potential discoveries, as well as jeopardize the certainty of the discovered patterns. For example, if a value bias were something like the following: "EMPLOYEE_SALARY > $50,000", this could eliminate a large

enough volume of data so as to make the extracted results statistically questionable.

3.3 Semi-Autonomous Knowledge Discovery Algorithm and Examples

Having discussed the basic concepts and issues behind the framework for semi-autonomous knowledge discovery and reference tracing, we now proceed with the modified KDD/DM process model and flow, general algorithm, and then give a detailed example of how this framework would work against an example database.

3.3.1 KDD/DM Model and Process Flow for Semi-Autonomous Knowledge Discovery and Reference Tracing

Figure 6 depicts the modified KDD/DM model for semi-autonomous knowledge discovery. The modified process flow is as follows for a typical KDD/DM session using the framework of semi-autonomous knowledge discovery and reference tracing:

1. The user initiates the KDD/DM session by either entering the initial user input database object set, or requesting that the available set of objects be displayed to the user.
2. The controller receives the request, and determines the type:
    - If it is a request for display of available objects, the controller directs the focus module to determine all database objects available to the user. Upon receipt of this available database object set, the controller displays this set to the user.
    - If the request contains an initial user input database object set, the controller passes the object set to the focus module for validation to determine that the database objects exist and that the user has permission to view the objects.
3. Upon a successful initial user input database object set (one or more initial objects), and upon successful validation (validation is required - the user could inadvertently commit a typographical error, which would be caught during validation; i.e., "object does not exist") of all of those objects, the controller directs the reference trace module to derive related objects.
4. The reference trace module determines the related database objects, and notifies the controller that it has created the reference tree.

5. The controller displays the original set of user objects back to the user, as well as the derived objects and their strengths relative to the initial set.
6. The user evaluates the original and derived sets, and then either modifies or accepts the database object sets. If the user modifies the database object sets (either the initial or the derived), this request is passed to the controller again, and steps 2 through 6 iterate. If the user accepts the database object sets, the user applies bias (if any), and passes the request to the controller to initiate the pattern extraction phase.
7. The controller notifies the focus module of the database object sets, and directs the focus module to begin the data extraction phase.
8. The focus module determines which database objects and attributes to retrieve, and passes the database retrieval request to the database interface.
9. The database interface retrieves the data from the database, and returns the data to the focus module.
10. The focus component notifies the controller that it has completed it's process.
11. The controller notifies the pattern extraction module that the data is available.
12. The pattern extraction module performs the actual pattern extraction/discovery process, and when completed, notifies the controller that it is done.
13. The patterns and discoveries are displayed to the user.

In the modified KDD/DM model and process flow, note that the domain knowledge comes from the user, and is only used to form the basis for the initial and derived database object sets. In addition, the reference tracer module uses the implicit domain knowledge from the logical data model, and the initial user input database object set t form the final database object set that it used for pattern extraction.

Another important note is that the discovered knowledge is fed back to the user, rather than directly to the knowledge base. This implies that the user decides what discovered knowledge is worth keeping, and what is not. This particular issue will be discussed in forthcoming sections.

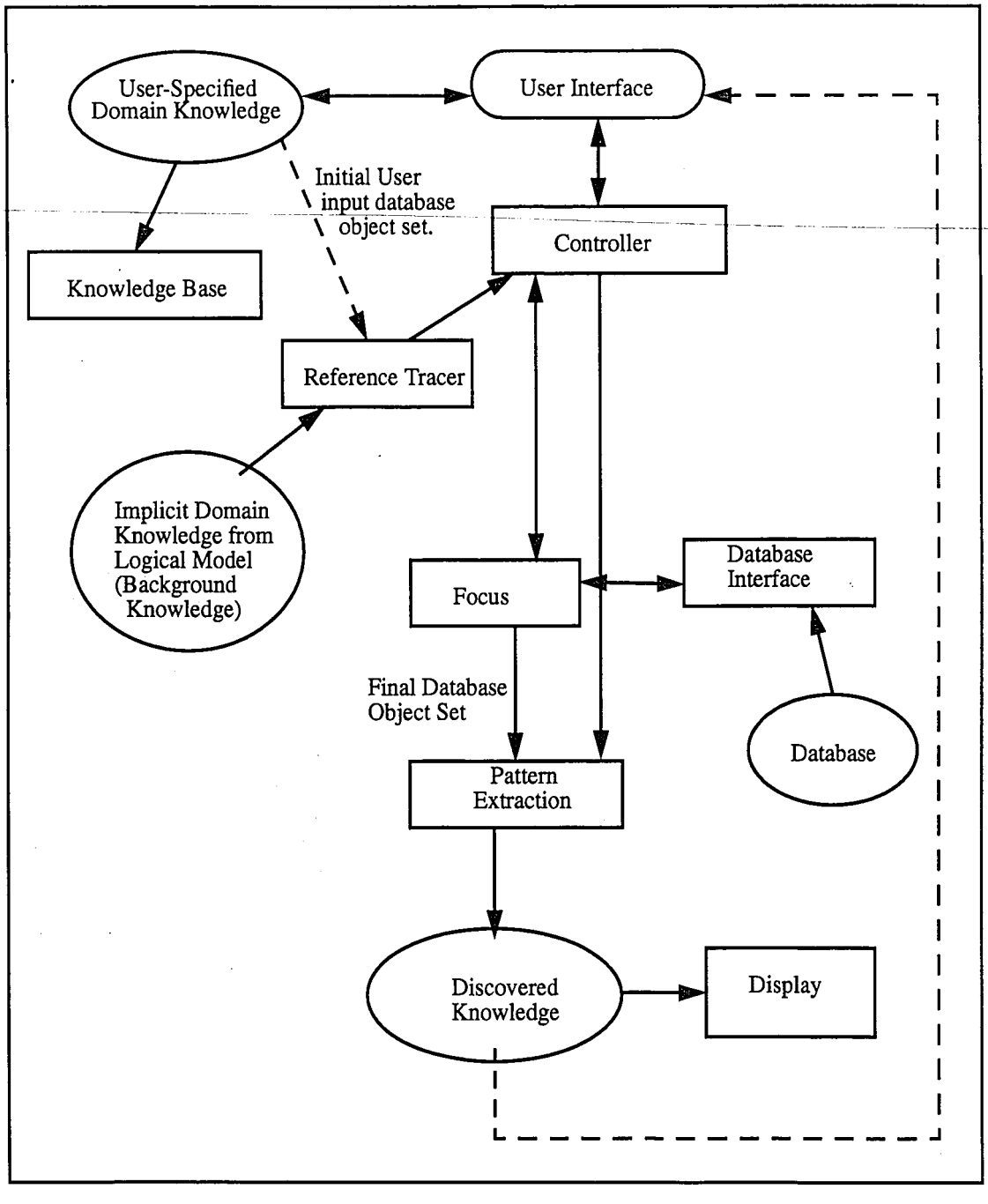

Figure 6: Modified KDD/DM model.

3.3.2 Semi-Autonomous KDD/DM and Reference Tracing Algorithm

The pseudo-code below is divided into two parts. Figure 7 displays the general algorithm for iterative and interactive semi-autonomous knowledge discovery with the retrieval and modification of the initial and

derived database object sets. This part of the algorithm makes the call to the reference tracing method. Figure 8 shows the pseudo-code reference tracing algorithm:

```
{
  dbObjSetFinal = NULL;  userInputDbObjectSet = NULL;
  dbDerivedObjectSet = NULL;
  moreUserInput = True;
  while (moreUserInput) {
    if (prompt("Display available database objects?") == 'Y')
      displayAvailableUserInputObjects();
    }
    userInputDbObjectSet = retrieveUserInputObjects(moreUserInput);
    if (validate(userInputDbObjectSet)) {
      dbDerivedObjectSet = referenceTrace(userInputDbObjectSet,1);
    }
    displayDerivedDatabaseObjectSet(dbDerivedObjectSet);
    if (prompt("Alter initial or derived database object sets?")  != 'Y') {
      moreUserInput = False;
    }
  }
  dbObjSetFinal = userInputObjectSet union dbDerivedObjectSet;
}
```

Figure 7: Pseudo-code for semi-autonomous knowledge discovery.

The pseudo-code for the semi-autonomous knowledge discovery (figure 7) shows the basic loop via which the user enters the initial user input database object set (potentially by selecting from a list of available database objects), and the potential modification of the initial and derived database object sets. The final object set from pattern extraction will be executed against is the dbObjSetFinal, which is the union of the userInputObjectSet and the derived database object set. This is to avoid duplicate objects in the user input object set, and the set derived via reference tracing. Note also that the initial call to the referenceTrace method involves passing an integer argument of 1. This integer argument is the initial derived database object strength value, for any derived objects that may be found.

Figure 8 shows the pseudo-code for the actual reference tracing. It uses a recursive method called referenceTrace, which accepts two parameters, an input database object set, and a strength. The algorithm first assigns the input database object set to a local variable. Then it steps through each object in the input database object set, determines the objects' primary and foreign keys, and queries the underlying database data dictionary using either primary or foreign key as the value to find related database objects. Any

related objects are inserted into the derivation tree along with their strength, and then a temporary derived database object set is built, composed of the current contents of the temporary derived database object set union against the primary or foreign key derived object sets. This is to build a database object set comprised of the current derived database objects; this temporary derived database object set will later be returned to the calling reference trace algorithm once the recursion is complete. Finally, after the primary and foreign key derived database object sets are built, a recursive call to referenceTrace is made, passing the result of a relational minus operation of the input database object set and the temporary local set, and the current strength plus one. This will build the next level of the derivation tree starting with the previous levels' objects. When a call is made to referenceTrace with a NULL input database object set, this indicates that no additional related objects were found between this call and the last call, so the recursion stops, and all of the temporary derived database object sets for any given point in the recursion minus the input database object set for that recursion, are returned to the original caller, and eventually, the semi-autonomous knowledge discovery algorithm, which will now have the complete derived database object set.

```
referenceTrace(InputDatabaseObjectSet, strength) {
 tempDerivedDatabaseObjectSet = InputDatabaseObjectSet;
 if (InputDatabaseObjectSet IS NULL) or (strength > USER_STRENGTH_MAX) {
    return;
 else {
    integer i = 0;
    for (each object in the InputDatabaseObjectSet) {
      currentUserDatabaseObject = InputDatabaseObjectSet[i];
      for (each primary key in the currentUserDatabaseObject) {
        pkDerivedDatabaseObjectSet = queryDatabaseDataDictionary(primary_key);
      }
      tempDerivedDbObjSet = tempDerivedDbObjSet union pkDerivedDatabaseObjectSet;
       insert_into_derivation_tree(pkDerivedDatabaseObjectSet,strength);
      for (each foreign key in the currentUserDatabaseObject) {
        fkDerivedDatabaseObjectSet = queryDatabaseDataDictionary(foreign_key);
      }
      tempDerivedDbObjSet = tempDerivedDbObjSet union fkDerivedDatabaseObjectSet;
       insert_into_derivation_tree(fkDerivedDatabaseObjectSet,strength);
      i++;
    }
     referenceTrace(tempDerivedDatabaseObjectSet minus InputDatabaseObjectSet,
                    strength+1);
    return ( tempDerivedDatabaseObjectSet minus InputDatabaseObjectSet);
  }
}
```

Figure 8: Pseudo-code for reference tracing algorithm.

Notice in Figure 8 that an additional condition for stopping the recursion is if the strength for any given pass of the recursion is greater than the USER_STRENGTH_MAX. This assumes a user-imposed bias that limits the height of the derivation tree. Although figures 7 and 8 do not show input of user bias, this would be determined during the execution time as depicted in figure 7.

3.3.3 Semi-Autonomous KDD/DM and Reference Tracing Example

This section will give a detailed example of the framework discussed in this paper. Figure 9 describes the sample database schema that will be used for this example. The steps that follow are a sample semi-autonomous knowledge discovery session using reference tracing against the database schema described in figure 9.

1. The user begins a session. The user is unaware of the database design, or objects within it, and decides to start off the KDD/DM session by requesting for a display of all database objects.
2. The KDD/DM session returns a list containing all database objects described in figure 9; i.e., SUPPLIER, PART, SUPPLIER_PART, LOCATION, MATERIAL, MTYPE, SUPPLIER_ADDRESS.
3. The user responds to this by selecting the SUPPLIER and PART database objects. These objects comprise the initial user input database object set.
4. The KDD/DM session validates that these objects do in fact exist. The objects do exist, and validation is successful, so reference tracing begins.
5. Reference tracing starts with a call to the reference trace module, passing the SUPPLIER and PART database objects as the initial user input database object set, and strength of 1.
    - The first pass of reference tracing loops through the input database object set consisting of {SUPPLIER and PART}. Starting with SUPPLIER, the primary key of "sno" is used to query the data dictionary, and find related database objects. For SUPPLIER, the sno key derives SUPPLIER_PART. This is unioned with the temporary derived database object set ({SUPPLIER, PART}; this was initialized to with the input database object set) to produce the temporary derived database object set. Then, the database objects derived via the primary key {SUPPLIER_PART} are inserted into the derivation tree with a strength of one. In similar fashion, the related objects via the SUPPLIER foreign keys are derived, which in this case happens to be SUPPLIER_ADDRESS and LOCATION. Again, these are unioned with the temporary database object set (which now consists of {SUPPLIER, PART, SUPPLIER_PART}) and then the foreign key-derived database objects are inserted into the derivation tree. The SUPPLIER object is now finished, and the same process proceeds with the remaining input database object, PART. The database derived object from PART includes {SUPPLIER_PART, LOCATION, MATERIAL}.

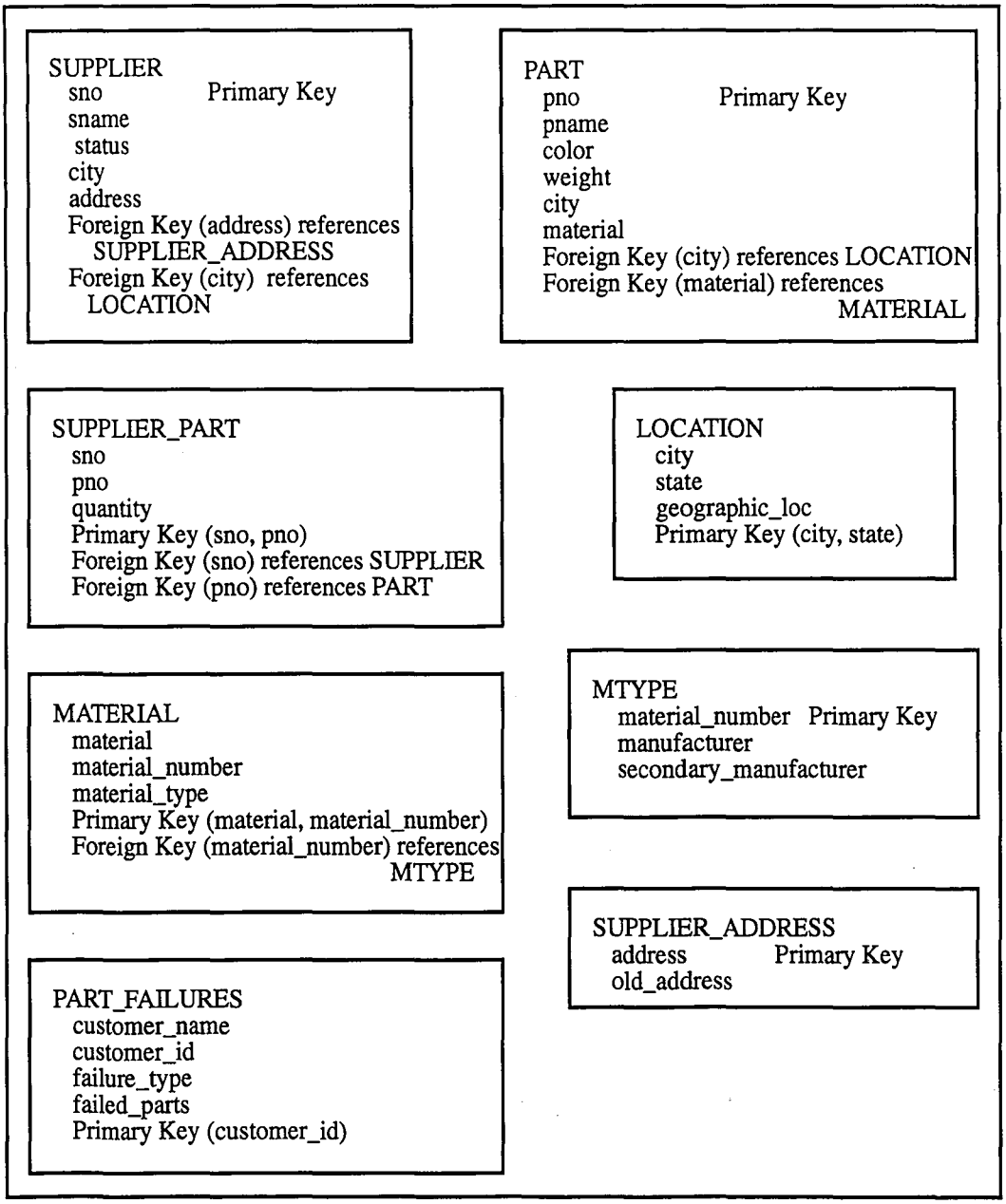

Figure 9: Sample supplier-part database schema for detailed example.

These are inserted into the derivation tree, and then the loop through the input database object set is completed. Finally, a recursive call to referenceTrace is made with temporary derived database objects set minus the input database object set ({SUPPLIER, PART, SUPPLIER_PART, SUPPLIER_ADDRESS, LOCATION, MATERIAL} minus {SUPPLIER,

PART}, yielding {SUPPLIER, PART, SUPPLIER_PART, SUPPLIER_ADDRESS, LOCATION, MATERIAL}), and a strength of 2.

- The second pass of the recursion is called with an input database object set of {SUPPLIER, PART, SUPPLIER_PART, SUPPLIER_ADDRESS, LOCATION, MATERIAL} and strength of 2. Following the same logic of the above, the new derived object set consists of {SUPPLIER, PART, SUPPLIER_PART, SUPPLIER_ADDRESS, LOCATION, MATERIAL, MTYPE}, with MTYPE coming from the composite primary key of material, material_number from the MATERIAL database object. The database objects are inserted into the derivation tree, and a recursive call to referenceTrace is made with this new temporary derived database object set minus the input database object set ({SUPPLIER, PART, SUPPLIER_PART, SUPPLIER_ADDRESS, LOCATION, MATERIAL, MTYPE} minus {SUPPLIER, PART, SUPPLIER_PART, SUPPLIER_ADDRESS, LOCATION, MATERIAL}, yielding {MTYPE}), and a strength of 3.
- The third pass of the recursion is called with the input database object set of {MTYPE} and strength 3. Again, the primary and foreign keys are followed, which yield no additional database derived object sets. Nothing will be inserted into the derivation tree, and when the recursive call is made to referenceTrace with {SUPPLIER, PART, SUPPLIER_PART, SUPPLIER_ADDRESS, LOCATION, MATERIAL, MTYPE} minus {SUPPLIER, PART, SUPPLIER_PART, SUPPLIER_ADDRESS, LOCATION, MATERIAL, MTYPE}, this yields a NULL set being passed to referenceTrace with a strength of 4.
- The fourth pass of the recursion evaluates the input object set of NULL, and exits out of the recursion. This produces the following database objects being returned to the original invocation of referenceTrace:
    - Recursive call 3 returns: ({MTYPE} minus {MTYPE}) returning NULL.

- Recursive call 2 returns: ({SUPPLIER, PART, SUPPLIER_PART, SUPPLIER_ADDRESS, LOCATION, MATERIAL,MTYPE} minus {SUPPLIER, PART, SUPPLIER_PART, SUPPLIER_ADDRESS, LOCATION, MATERIAL}) returning {MTYPE}.
- Recursive call 1 returns: ({SUPPLIER, PART, SUPPLIER_PART, SUPPLIER_ADDRESS, LOCATION, MATERIAL} minus {SUPPLIER, PART}) returning {SUPPLIER_PART, SUPPLIER_ADDRESS, LOCATION, MATERIAL}.

The final derived database object set returned from reference tracing is {{SUPPLIER_PART, SUPPLIER_ADDRESS, LOCATION, MATERIAL}, {MTYPE}}. These represent the derived database objects obtained via reference tracing. Note that the database object PART_FAILURES was not derived from reference tracing, since it had no primary or foreign key relationships with the initial user input database object set.

6. The derived database object set is displayed to the user. The user decides that this derived object set does not require modification, and responds "no" to the prompt. This concludes the initial interactive semi-autonomous knowledge discovery using reference tracing session.
7. The initial user input database object set is unioned with the derived set ({SUPPLIER, PART} union {{SUPPLIER_PART, SUPPLIER_ADDRESS, LOCATION, MATERIAL}, {MTYPE}}, to produce the final database object set to be used for pattern extraction, {SUPPLIER, PART, SUPPLIER_PART, SUPPLIER_ADDRESS, LOCATION, MATERIAL, MTYPE}.
8. The controller notifies the focus component of the final database object set to be used for database data retrieval and pattern extraction.
9. The pattern extraction process begins. Discovered patterns are displayed as per the ordering described in section 4.2.4; inverse order of their strengths, since as described earlier, these will be the most interesting to the user, and related to the user's original interests.

Figure 10 depicts the derivation tree built for this example:

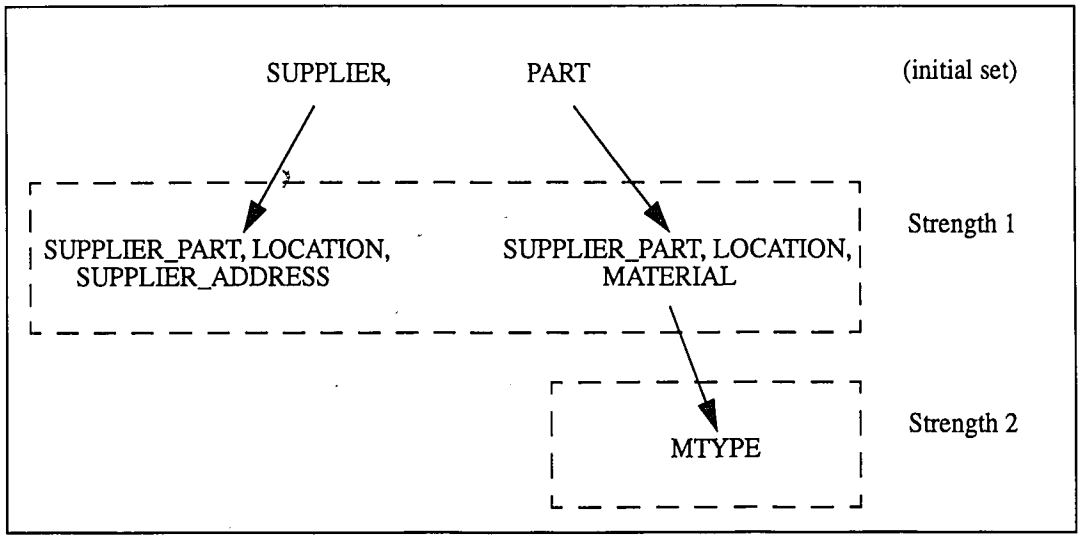

Figure 10: Derivation tree built via reference tracing for example session.

This example illustrates how related objects, based upon what the user determines is interesting, are derived.

To briefly illustrate how discovered patterns would be displayed to the user, let us assume that the pattern extraction methodology is dependency analysis, and that pattern extraction has produced the following dependencies:

1. All suppliers named ABC come from location B.
2. All non-ALUMINUM material is made by manufacturer Z.
3. All parts that come from location A have material name ALUMINUM.

The display of the dependencies would be in the following order: (3) would be displayed first, (1) would be displayed second, and (2) would be displayed third. The rational for this ordering is that dependency (3) has a direct relationship to the initial database user input object set, and the dependency includes a database object that had the lowest strength (strength of 3). Thusly, in the framework described in this paper, and with respect to the "unexpectedness" described in [2], this has the highest probability of being

considered interesting to the user, while retaining relevancy. Dependency (1) is displayed second because this dependency is directly related to the initial database object set, and involves derived database objects of strength 2. Dependency (2) is displayed third, because despite the fact that it involves derived database objects of strength 3, it has no direct relationship to the initial database input object set. However, as discussed previously, this discovery is still presented, and may in fact cause the user to explore this dependency further by altering their input database object set, and execute additional discovery sessions with the modified database object set.

3.4 Benefits of Semi-Autonomous Knowledge Discovery and Reference Tracing

This section of the paper will discuss some of the benefits and use of the output of the semi-autonomous knowledge discovery and reference tracing framework discussed in this paper, with respect to the intended goal of producing discoveries that are relevant and interesting to the user.

3.4.1 Database Understanding and Visualization

The problem of presenting database objects and their relationships to other database objects is an open research area, from both an academic and industry perspective. The techniques discusses in this paper do not break new ground in this area, but do present a set of techniques that base the finding and relating of database objects relative to an initial set of user-chosen database objects which represent the users interests. Further, the relationships are grouped in terms of strengths, which then permits an ordering of results based upon these strengths, which allows the user an easier way of visualizing the database in terms of interesting database objects and discovered patterns.

3.4.2 Named Abstractions as Database Objects

Dhar and Tuzhilin [10] presented a method for knowledge discovery using set of user-defined predicates, which are either base database objects, views across one or more base database objects, or previously defined predicates. This allowed the user to view and classify the database to their own interests, and base the discovery process off of these predicates and hence, their interests. These predicates are stored in an underlying database object, and retrieved as necessary during the discovery process.

Semi-autonomous knowledge discovery allows knowledge discovery to take place based upon the users

interests as well, and presents a similar method for storing the results of reference tracing, which are similar to Dhar and Tuzhilins' predicates. If we consider the example of the supplier-part database schema, the derivation tree is an abstraction of the users' initial interests, as well as autonomously-derived related database objects. This abstraction represents the users interests; in particular, it represents the users interests over potentially several iterations of refinement. The framework described in this paper allows for the storage of the derivation tree, or components of the derivation tree, in an underlying database object, for future reference during knowledge discovery. Further, the user can create a "named-abstraction" by simply classifying the derivation tree, or components thereof.

For example, the derivation tree depicted in figure 10 may be classified as SUPPLIER_PART_INFO. If the user so chose to store this derivation tree as a named abstraction, it would be stored in an underlying database object using the following information:

- Abstraction name; in this example, SUPPLIER_PART_INFO.
- Abstraction database objects; initial and derived.
- Strengths of the objects.
- User bias.

Figure 11 depicts a sample representation of a named abstraction using the relational database model. The benefits of named abstractions are similar to Dhar and Tuzhilin: the user represents their interests and what they are interested in relative to the database, and lets the user define it in their own terms. Further, there is an efficiency gain realized by using previously-defined named abstractions. A named abstraction represents not only the users interests, but also the results of reference tracing (derivation tree), which may be costly in terms of computation, but also the final set of derived database objects may also represent multiple iterations where the user refines the initial and derived database object sets. By permitting the storage and subsequent retrieval of the named abstraction, an efficiency savings is realized, This would be accomplished by the use of the named abstraction as either part of, or as the entire, initial user input database objects set. The KDD/DM framework, upon determining that some part of the initial user

input database object set included a named abstraction, would simply not execute reference tracing on any database objects included in the abstraction, and would only execute reference tracing against database objects that were not part of the named abstraction.

ABSTRACTION_NAME	INITIAL_OBJ	DERIVED_OBJ	STRENGTH	BIAS
SUPPLIER_PART_INFO	SUPPLIER	SUPPLIER_PART, LOCATION, SUPPLIER_ADD.	2	NONE
PART_MATERIAL_INFO	PART	SUPPLIER_PART, LOCATION, MATERIAL	2	NONE

Figure 11: Sample relational representation of a named abstraction.

Consider the database schema again of figure 9. Suppose, for example, the user had previously executed the KDD/DM session that resulted in the derivation tree depicted in figure 10. This derivation tree was stored as a named abstraction called SUPPLIER_PART_INFO. During a subsequent session, the user now decides to use the SUPPLIER_PART_INFO named abstraction, but also decides to include the PART_FAILURES database object, as the initial user input database object set. The semi-autonomous knowledge discovery framework described in this paper would execute as previously described, with the exception that the SUPPLIER_PART_INFO abstraction would be recognized, and reference tracing would only be executed against the PART_FAILURES database object. A simple scan of stored named abstractions would suffice to determine if the SUPPLIER_PART_INFO input database object was a named abstraction, or a database object other than a named abstraction[1]. In addition, named abstractions would be generated as part of the initial list of database objects, if the user chose to have the KDD/DM system initially display available database objects to the user.

One problem that exists with named abstractions is that their stored representation could become false

1. Naming conflicts would generate an error at storage time of the named abstraction. If, for example, the user attempted to store the SUPPLIER_PART_INFO named abstraction as SUPPLIER, the KDD/DM system would raise this as an error, conflicting with the database object SUPPLIER, in which case the user would be prompted to choose another name.

with respect to the current database definition. Over time, it is possible that a named abstractions stored at time T0 becomes out of sync with respect to the database at time T0 + n. This could occur at the database object, attribute, or bias levels. This implies that either a run-time check of the integrity of the named abstraction would occur while it is being used in a KDD/DM session, or a periodic, system-initiated, verification and validation of the database objects, attributes, and biases that the named abstraction represents could be accomplished via the use of the operating systems' time-based job initiation.

3.4.3 Knowledge Base of Named Abstractions and Concept Areas

Over time, a user may build up a substantial knowledge base of named abstractions, each of which is a stored representation of a given users' interests, derived related database objects, and any particular bias that the user felt further represents their interests for the given set of database objects. We can consider groupings of related abstractions as "concept areas". A concept area is nothing more than a high level, user-defined representation of a users' perception(s) of a database. This notion of segmenting the knowledge base is not new, and in terms of belief systems, is referred to as a "belief space" [13]. It is a categorization and segmentation of a wide universe of facts, each of which may or may not have things in common, and each of which we may have different understandings and beliefs about. For example, a concept area for the sample schema in figure 9 might be SUPPLIERS, another might be PARTS, another might be FAILURES. Each one of these areas has related and non-related database objects. Further, each one of these may have corresponding groupings of named abstractions. Similar to a named abstraction, the KDD/DM framework described in this paper allows for the storage and retrieval of high-level concept areas, which in essence point to one or more named abstractions. Figure 12 depicts a concept area representation using the relational model.

CONCEPT	IS_A	HAS_A
THE_SUPPLIERS		SUPPLIER_PART_INFO
THE_PARTS		PART_MATERIAL_INFO

Figure 12: Sample relational representation of a concept area.

Note that the representation depicted in figure 12 has two attributes, "IS_A" and "HAS_A". These attributes indicate that we can have concept classification hierarchies. The IS_A attribute indicates that the given concept is a kind or type of a thing; it is a lower element on a classification hierarchy. For example, THE_SUPPLIERS, if we had an expanded database, might be part of a larger BUSINESS_PARTY concept. The HAS_A attribute indicates that the given concept contains the other concepts; in figure 12, the SUPPLIERS concept contains a named abstraction, SUPPLIER_PART_INFO. Similar to named abstractions, concept areas may be used as part of the initial user input database object set. The overall goal is to expedite the discovery process, and to tie in discovered patterns more closely to users interests.

The named abstractions and concept areas are ways of representing a user's interest in this KDD/DM framework. These classifications and hierarchies represent a given user's notion of the database, as it applies to their interests. the following section will explore how the named abstractions and concept areas integrate with the actual pattern extraction and subjective interestingness areas of this KDD/DM framework.

3.4.4 Subjective Interestingness Relative to Concept Areas and Named Abstractions

Concisely, the author of this paper posits that the users' subjective interests are embodied in the named abstractions and concept areas. The concept areas are composed of one or more named abstractions, and the named abstractions are composed of one or more initial user input database objects and related derived database objects. Any and all discovered patterns that are related to either concept areas or named

abstractions are related to a user's interests, and categorized by the concept area(s) and/or named abstraction(s).

The named abstractions contain derived, related database objects in varying strengths with respect to the initial user input database object set. The interestingness of the discovery is inversely proportional to it's un-relatedness to the initial database objects set, and proportional to it's strength (the higher the strength of the derived database objects that a discovery includes, the higher it's level of unexpectedness and interestingness to the user, based upon [2]). This permits an ordering of subjective interestingness of discovered patterns, based upon concept area and named abstractions. It is based upon this that the author of this paper argues that *all discovered patterns, knowledge, etc., are interesting to the user if they are based upon the concept areas, named abstractions, or initial or derived database objects.*

The focus of this paper is on the user-oriented view of the database and subjective interestingness. The two previous paragraphs argue for the statement that the user has captured their view of a database system in their notions of the database objects and related objects, which are represented as the named abstractions and concept areas. It is a way for the subjective interests of a user to be able to be contained and defined, and for all discoveries to be based upon their relevance to this classification. Intuitively, we see that if a user were asked to explain to us "What is this database comprised of?", they would likely respond with high level responses and concepts, rather than low level attribute names and primary key/foreign key relationships. Additionally, there may be some subject areas of a database that a user does not care about. The named abstractions and concept areas are a means by which the user may encapsulate what their interests about the database are, on a per user basis. Discoveries are then directly related to these areas, and only these areas, and may be presented and displayed using the ordering discussed previously, and also based upon the abstractions and concept areas they belong to.

For example, considering the representations of concept areas and named abstractions discussed in figures 10, 11, and 12, and using the discovered dependency analysis rules as the discovered patterns, the discovered dependency rules would be displayed as follows:

1. Concept: THE_PARTS

   Abstraction: PART_MATERIAL_INFO

   Discovery: IF (MATERIAL_TYPE = ALUMINUM) THEN

   (LOCATION = A)

2. Concept: THE_SUPPLIERS

   Abstraction: SUPPLIER_PART_INFO

   Discovery: IF (SUPPLIER_NAME = ABC) THEN

   (LOCATION = B)

3. Concept: THE_PARTS, THE_SUPPLIERS

   Abstraction: SUPPLIER_PART_INFO, PART_MATERIAL_INFO

   Discovery: IF (MATERIAL_NAME = ALUMINUM) THEN

   (MANUFACTURER = Z)

Alternatively, the abstraction could be chosen not to be displayed, but this provides a nice tracing mechanism, showing which abstractions were used to discover what rule. If the user is in doubt about the discovery, then it is possible for the base database objects to be viewed to determine if a discrepancy exists. Note in the third discovered rule that two abstractions and concept areas were used. this is because the discovered rule was found across abstractions, and similarly, across concept areas.

3.4.5 Interestingness and Belief Systems for Semi-Autonomous KDD/DM

The discussions thus far have centered around using user-defined subjective interestingness to limit the KDD/DM process to only the patterns and discoveries that the user finds interesting. Further, to classify this system of subjective interestingness, the ideas of named abstractions and concept areas were introduced to implement a hierarchical, user-oriented view of a database system and it's discovered patterns. In section 3.4.4, we saw how the discovered patterns and knowledge can be grouped by concept area and

abstraction. This segmentation of the database into concept areas, as pointed out in section 3.4.4, is analogous to belief spaces [13]; i.e., a method for breaking up a belief system of facts, knowledge, etc., into smaller defined segments, each of which has a generalized name, and each of which has a set of beliefs associated with it. For example, two sets of belief spaces might be "machines" and "animals". We have different beliefs about each of these, which may or may not have elements in common. Relative to these belief spaces, we have generalized concepts, and then low level concepts, similar to the breakdown of concept areas into named abstractions, and then the low-level database objects. Data and evidence exist in the universe that either confirms or denies the beliefs associated with these two belief spaces, and if data or evidence proves the beliefs associated with these two belief spaces differently, then the beliefs are adjusted accordingly[1].

Similar to the notion of the belief system and the belief spaces, the author of this paper argues that the framework presented in this paper indicates that:

- The entire collection of implicit and explicit knowledge, data, facts, rules, etc., that the database comprises is representative of a subset of the users' belief system; i.e., the user has a belief system associated with the database and it's domain.
- The classification of low-level database objects and derived objects into named abstractions and concept areas is analogous to taking the belief system and segmenting it into belief spaces relative to the database.

Semi-autonomous knowledge discovery and reference tracing help define the belief spaces/concept areas. And, similar to a belief space, for each database concept area and abstraction, the user has beliefs about it. For example, it is intuitive that for the concept area THE_SUPPLIERS, the user has some basic understanding and beliefs about this concept, and even the lower-level abstractions and database objects. The presentation of the discoveries relative to the concept area will either confirm or refute the users' beliefs about the concept areas. The argument is that the concept area is equivalent to a belief space, and any dis-

1. Bayesian approaches and Dempster-Shafer approaches to belief systems differ. Refer to section 2.2 for the differences between these approaches.

coveries that belong to that concept area will either confirm or contradict the existing data, or beliefs that the user maintains about that area [2].

This forms the basis for using semi-autonomous knowledge discovery and reference tracing for building and storing discoveries against concept areas. All discoveries are grouped according to their respective concept areas (belief spaces) and abstractions, as described in section 3.4.4. For example, consider the following:

1. Concept: THE_PARTS

   Abstraction: PART_MATERIAL_INFO

   Discovery: IF (MATERIAL_TYPE = ICE) THEN

   (LOCATION = MEXICO)

There are several considerations here. The user may not expect that a MATERIAL_TYPE is ICE. Or, the user may believe that the MATERIAL_TYPE is ICE, but was not aware of, or does not believe that, this information can be found in the database. Or, the user may agree that the MATERIAL_TYPE is ICE, but may doubt that this material could come from MEXICO. Or, the user may believe the discovery. In any case, the user either believes or does not believe this discovery, relative to what they already believe about the concept area. This follows with the notions of hard and soft beliefs as discussed by Silberschatz and Tuzhilin in [2].

Discoveries that are made with respect to named abstractions and concept areas may then be stored in the user-specific domain knowledge base, relative to the concept areas and named abstractions they are related to. All of these discoveries will be of subjective interest to the user, and can now be tied to concept areas of interest. The model for concept areas and related discoveries is depicted in figure 13.

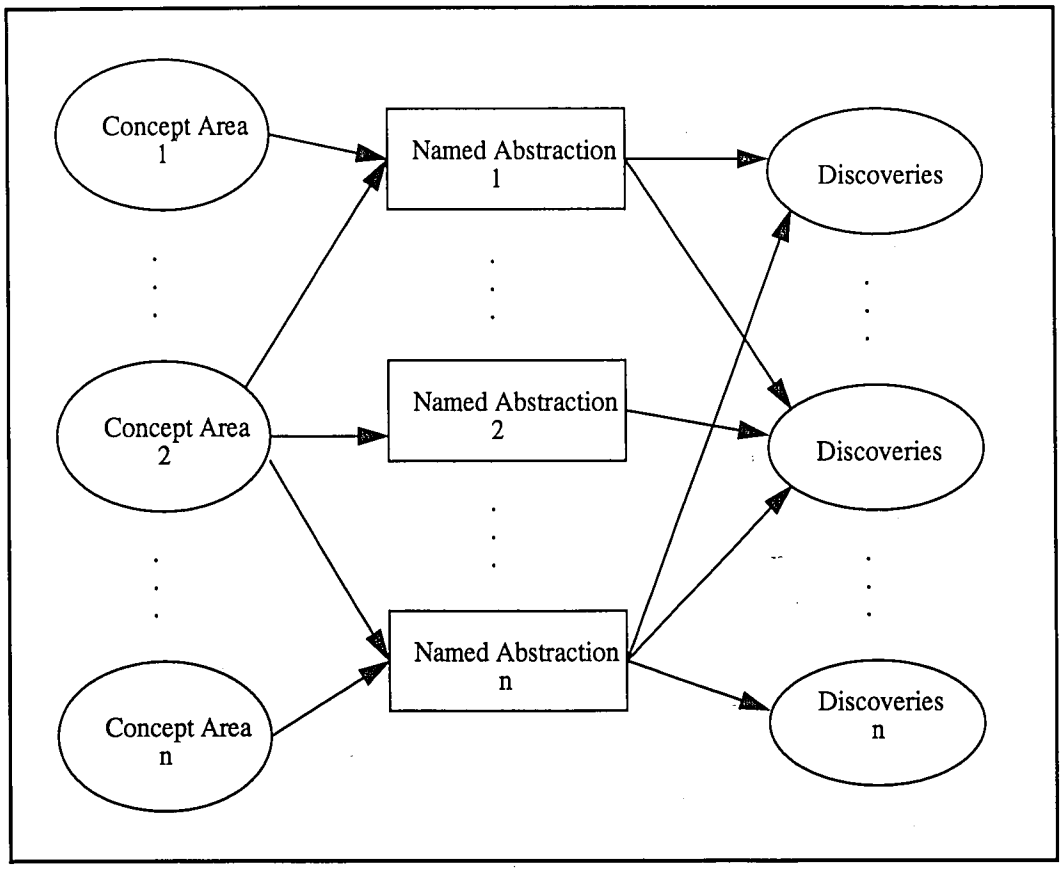

Figure 13: Model for relating discoveries to named abstractions and concept areas.

The model in figure 13 indicates that one or more concept areas are related to one or more named abstractions which may potentially produce one ore more discoveries. The relationship between the concept areas, named abstractions, and potential discoveries is one to many; each concept area can encompass one or more named abstractions, and a named abstraction could be related to one or more concept areas. Similarly, each named abstraction may potentially produce one or more discoveries, and each potential discovery may be related to one or more named abstractions. Additionally, recall that each named abstraction may encompass one or more user input database objects and derived database objects.

This relationship model allows more accurate linkage between the discovered knowledge, and the individual user's subjective interest areas and beliefs about those areas. In addition, it allows this linkage to occur on a per user basis.

3.4.6 Storage and Feedback of Discovered Knowledge

This paper has attempted to equate user-oriented subjective interest to discovered knowledge and the users' beliefs and belief systems of a given database domain. One final component to be discussed with semi-autonomous knowledge discovery and reference tracing is the storage of these discoveries and the potential feedback of this domain knowledge. One of the open areas of research in terms of KDD/DM is the storage of discovered knowledge and it's retrieval and use in subsequent KDD/DM sessions. While there are many issues and problems associated with this, two of the most critical are:

1. What discovered knowledge should be stored as domain knowledge.
2. If discovered knowledge is stored as domain knowledge, what, if any, should be used in later knowledge discovery sessions?

This paper does not have a definitive answer to either of these two issues. However, semi-autonomous knowledge discovery and reference tracing may provide a framework via which these two issues may more easily be resolved. What follows is a brief discussion on this subject, on the intuitive level.

Under semi-autonomous knowledge discovery and reference tracing, all discoveries are considered relevant and interesting to the user, but within a given set of discoveries, there will be varying degrees of interest. Potential discoveries are displayed to the user in decreasing order of their unexpectedness, but this is only to aid the user in terms of subjective interestingness. Recall the modified KDD/DM model depicted in figure 6. This model shows all potential discoveries passing through the user prior to being stored as user-dependent domain knowledge. In this manner, the user themselves are the final "filter" of interestingness, relevancy, and accuracy. If the user either does not believe discoveries, or the discovery is not of sufficient interest to them to warrant storage as domain knowledge, then the discovery is simply displayed and disregarded.

Since discoveries are displayed in terms of their unexpectedness by concept area, then it follows that the discoveries will either be new knowledge to the user, or will either confirm or contradict existing beliefs that the user had about the given concept area. In [2], Silberschatz and Tuzhilin present a matrix of

"actions to be taken with beliefs when a new pattern is discovered" [2]. Figure 14 depicts this matrix:

Belief	Contradictory	Non-Contradictory
HARD	change data	accept data
SOFT	check data	accept data

Actions to be taken with Data when a new pattern is discovered.

Belief	Contradictory	Non-Contradictory
HARD	do nothing	do nothing
SOFT	depends on the data	update degree of belief

Actions to be taken with Beliefs when a new pattern is discovered.

Figure 14: Decision matrix for discovered patterns relative to beliefs [2].

The matrix covers two areas, actions upon data, and actions upon beliefs. On an intuitive level, it is reasonable to expect that new patterns that are contradictory and non-contradictory in terms of soft beliefs for a given concept area have the highest probability of being accepted and stored as domain knowledge, after review by the user. Patterns that are non-contradictory for hard beliefs are typically intuitively obvious to the user, and are not interesting, even though they may be unexpected and displayed early on in the reference tracing ordering. Patterns that are contradictory to hard beliefs may cause the user to inspect the data, named abstractions, etc., but are doubtful to the user as valid discoveries, and would probably not be stored as domain knowledge.

All discoveries that are accepted by the user and marked for storage as domain knowledge would be stored with references to the concept areas and named abstractions that they were derived from, along with the specific user identifier. This provides a mechanism for identifying which, if any, domain knowledge should be used during subsequent KDD/DM sessions, on a per-user basis. Consider the following

example. Suppose that three discoveries, D1, D2, and D3 have been extracted during KDD/DM session 1 by user A. User B determines that these discoveries are worthwhile storing as domain knowledge. Further assume that these three discoveries were derived from the SUPPLIER_PART_INFO named abstraction, and the THE_SUPPLIERS concept area, as described in section 3.4. After KDD/DM session A has completed, and the three discoveries are stored along with the associated concept areas, named abstractions, and user identifier, a subsequent KDD/DM session is begun, session 2. As part of the initial user input database object set, user A includes the concept area THE_SUPPIERS. Since discoveries D1, D2, and D3 were stored with their relationships to the THE_SUPPLIERS concept area and identified with user A, then these three discoveries could immediately be identified as potential domain knowledge to be used during this session (session 2), and any future discoveries.

While this mechanism permits, at a high level, the identification of domain knowledge to be "feedback" into the KDD/DM system based upon logical concepts, it does not resolve the problem of which individual discoveries could be used for particular sessions. In the previous example, suppose that discoveries D1 and D2 were the only relevant domain knowledge to be fedback into the second KDD/DM session. the above method does not include mechanisms to exclude discovery D3 in this particular scenario. This is an open research problem. In addition, this mechanism has problems and issues of it's own. For example, as stored domain knowledge ages, it may become irrelevant and possibly false. The problem of updating the stored domain knowledge, which specific rules to update, etc., is also an open research problem. However, at a high-level, it is an initial mechanism that may be employed as a result of semi-autonomous knowledge discovery and reference tracing to begin to identify which stored domain knowledge may be applied during subsequent KDD/DM sessions.

4. Summary

This paper has discussed a series of techniques in order to provide a framework for knowledge discovery based upon user-oriented, subjective interests. To accomplish this, methods of semi-autonomy and reference tracing were discussed to embody the user's interests, yet expand the scope of these interests to include relevant and potentially interesting discoveries in terms of their unexpectedness. In this manner,

the user's interests are captured prior to patterns being discovered, and thusly providing a more efficient search space, and higher degree of relevancy of the discovered patterns for the user. Methods of storing these discoveries based upon named abstractions and concept areas were discussed, to provide a rudimentary means for referencing discovered patterns to the original concepts. Concept areas were discussed as representing belief areas that the user may have regarding the database domain. Finally, methods for feeding back discovered knowledge in subsequent knowledge discovery sessions were discussed. It is hoped that this paper will initiate further research in the areas of subjective interest-based knowledge discovery systems.

5. References

1. W.J.Frawley, G. Piatetsky-Shapiro, C.J. Matheus, "Knowledge Discovery in Databases: An Overview", *AI Magazine,* vol. 13, no. 3, 1992.
2. A.Silberschatz, A.Tuzhilin, "On Subjective Measures of Interestingness in Knowledge Discovery", Proceedings, *First International Conference on Knowledge Discovery and Data Mining,* AAAI Press, 1995, pg. 275-281.
3. W.J.Frawley, G. Piatetsky-Shapiro, eds., *Knowledge Discovery in Databases,* Cambridge, MA, AAAI/MIT, 1991.
4. U.M.Fayyad, R. Uthurusamy, eds., Proceedings, *First International Conference on Knowledge Discovery and Data Mining,* AAAI Press, 1995.
5. Brachman, et al., Proceedings, *AT&T TIES Conference on Knowledge Discovery in Databases,* June, 1995.
6. *IEEE Transactions on Knowledge and Data Engineering*, vol. 5, no. 6, December 1993.
7. Y. Cai, N. Cercone, J. Han, "Attribute-Oriented Induction in Relational Databases", *Knowledge Discovery in Databases,* AAAI/MIT, pg. 214-228.

8. G. Piatetsky-Shapiro, "Discovery, Analysis, and Presentation of Strong Rules", *Knowledge Discovery in Databases,* AAAI/MIT, pg. 229-248.

9. J. Hong, C. Mao, "Incremental Discovery of Rules and Structure by Hierarchical and Parallel Clustering", *Knowledge Discovery in Databases,* AAAI/MIT, pg. 177-194.

10. V. Dhar, A. Tuzhilin, "Abstract-Driven Pattern Discovery in Databases", *IEEE Transactions on Knowledge and Data Engineering,* vol. 5, no. 6, December 1993, pg. 926-938.

11. C.J. Matheus, P.K. Chan, G. Piatetsky-Shapiro, "Systems for Knowledge Discovery in Databases", *IEEE Transactions on Knowledge and Data Engineering,* vol. 5, no. 6, December 1993, pg. 903-913.

12. G. Gardarin, P. Valduriez, *Relational and Knowledge Bases,* Addison-Wesley, 1989.

13. J. Allen, *Natural Language Understanding,* Benjamin/Cummings, 1988.

14. E.X.DeJesus, S.R. Hedberg, K. Watterson, C.D. Krivda, "Data Mining", *BYTE Magazine,* vol. 20, no. 10, October 1995, pg. 81-103.

15. J.P. Yoon, L. Kerschberg, "A Framework for Knowledge Discovery and Evolution in Databases", *IEEE Transactions on Knowledge and Data Engineering,* vol. 5, no. 6, December 1993, pg. 973-979.

16. F. Bergadano, "Inductive Database Relations", *IEEE Transactions on Knowledge and Data Engineering,* vol. 5, no. 6, December 1993, pg. 969-972.

17. D.A. Bell, "From Data Properties to Evidence", *IEEE Transactions on Knowledge and Data Engineering,* vol. 5, no. 6, December 1993, pg. 965-969.

18. B.R. Gaines, "The Trade-Off Between Knowledge and Data in Knowledge Acquisition", *Knowledge Discovery in Databases,* AAAI/MIT, pg. 491-505.

19. R. Agrawal, T. Imielinski, A. Swami, "Database Mining: A Performance Perspective", *IEEE Transactions on Knowledge and Data Engineering,* vol. 5, no. 6, December 1993, pg. 914-925.

20. J. Giarratano, G. Riley, *Expert Systems,* Boston, MA, PWS Publishing, 1994.

6. Vita

David M. Grube graduated from Lehigh County Community College in 1980 with the Associate in Applied Science degree, and also graduated from Muhlenberg College in 1990 with the Bachelor of Science Degree in Computer Science. David was born September 20, 1960, in Allentown, Pennsylvania, to David W. and Theresa R. Grube. David currently is a Member of Technical Staff at AT&T Bell Laboratories in Warren, New Jersey.

END OF TITLE