ABSTRACT-DRIVEN PATTERN DISCOVERY IN DATABASES

by

Vasant Dhar
Information Systems Department
Leonard N. Stern School of Business
New York University
New York, NY 10003

and

Alexander Tuzhilin
Information Systems Department
Leonard N. Stern School of Business
New York University
New York, NY 10003

March 1992

Center for Research on Information Systems
Information Systems Department
Leonard N. Stern School of Business
New York University

Working Paper Series

STERN IS-92-11

Center for Digital Economy Research
Stern School of Business
Working Paper IS-92-11

Abstract-Driven Pattern Discovery in Databases

Vasant Dhar

Alexander Tuzhilin

New York University*

Abstract

In this paper, we study the problem of discovering interesting patterns in large volumes of data. Patterns can be expressed in *user-defined* terms and not only in terms of the database schema. The user-defined terminology is stored in a *data dictionary* that maps it into the language of the database schema. We define a pattern as a deductive rule expressed in user-defined terms that has a degree of certainty associated with it. We present methods of discovering interesting patterns based on *abstracts* which are summaries of the data expressed in the language of the user.

1 Introduction

Our interest is in large scale business databases which grow by millions of records daily. While this data is recorded primarily for accounting purposes, executives are interested in leveraging them for other purposes such as analyses of trends in the data. For example, marketing executives can learn about their consumers purchase behavior from credit card and scanner data and can use this knowledge to make decisions with respect to pricing, advertising, and promotions. Likewise, securities trading data can be monitored for patterns that might point to fraud or other irregularities. Clearly, with the massive volumes of data that flow into databases daily, the computer will play an increasingly important role in the analysis. The challenge, however, is for it to generate the "interesting" patterns, which may be hidden deep in the data.

Pattern discovery can be viewed as a generate-and-test problem [Win84], the major challenge being to constrain the generator into generating the interesting hypotheses. In scientific domains, the "interestingness" heuristics [Len77, LBS81] for focusing the generator are theory-based. In the business arena, we can take advantage of the fact that executives are usually interested in trends dealing with changes in aggregate-based functions such as totals and averages for terms in their

*Address: Information Systems Department, Stern School of Business, 40 West 4th Street, Room 624, New York, NY 10003; Internet: vdhar@stern.nyu.edu, atuzhilin@stern.nyu.edu

Center for Digital Economy Research
Stern School of Business
Working Paper IS-92-11

vocabulary that they in fact *want* to specify. This information provides some of the "interestingness" heuristics for focusing the hypothesis generator.

We present a method for extracting patterns from very large databases where the novelty lies in exploiting two types of knowledge. The first is a user-defined vocabulary that provides relational *views* of the data and is used to express generalization relationships among different data types. For example, a credit card company can define Yuppie as a person whose age is less than 35 and who makes more than $80000, or who has a Gold card. Also, we can define Wall-Street-Yuppie and Madison-Avenue-Yuppie as specializations of Yuppie. The second novelty is in how we use *abstracts*, which are summaries of the data expressed in terms of this vocabulary. The vocabulary and abstracts endow the system with the ability to search for patterns in terms of sets that are meaningful to the user, in effect, focusing the generator.

The idea of using an abstracted database was first proposed by Walker [Wal80]. His approach made use of the fact that the domain of an attribute can be abstracted, i.e. for the PET attribute, dogs and cats are mammals, snakes and turtles are reptiles and so on. In Walker's abstracted database, attribute values are replaced by the set to which they belong. Cai et.al [CCH91] use the same technique and search for dependencies among the abstracted attribute values. Our approach generalizes on Walker's and Cai's in that attribute values in an abstracted database can also be predicates or views of the original database, depending on multiple attributes. This makes it possible to derive patterns in user defined terms that would be very difficult to derive with attribute-oriented generalization alone.

In order to describe pattern discovery, we first need a precise definition of a pattern. Certainly, there is no standard definition of the term in the literature. In trying to draw a common thread through a recent collection of papers on "Knowledge Discovery in Databases," Frawley et.al. [FPSM91] define patterns as follows:

> Given a set of facts (data) F, a language L, and some measure of certainty C, a pattern S is a statement S in L that describes relationships among a subset F_S of F with certainty C, such that S is simpler (in some sense) than the enumeration of all facts in F_S.

This definition is made intentionally vague to cover a wide variety of approaches. For example, a linear regression [Cli87] qualifies as a pattern with the above definition as does a set of statistical parameters such as the mean and standard deviation for a collection of numerical values. In fact, any abstraction that in some sense summarizes the data would satisfy the above definition of pattern. In contrast to this, we define a pattern in a more restricted sense, as a rule that has associated with it a degree of certainty. The precise form of the rule will be described in Section 3.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-92-11

CUSTOMER (name, addr, income, profession, age, card_type)
TRANSACTION (name, merchant, type, amount, date)

Figure 1: CREDIT_CARD Database

The rest of the paper is organized as follows. In Section 2, we describe the terminology of the user that he or she employs to express patterns. In Section 3, we define a pattern. In Section 4, we describe an abstract as a summary of the data expressed in user-defined terms. In Section 5, we present a method to derive patterns from the data using abstracts. In Section 6, we define patterns with aggregates and present a method for discovering these types of patterns. In Section 7, we compare our approach with the previous work on pattern discovery.

2 Data Dictionary

Consider the following application where a user may be interested in patterns in the data that are expressed not in terms of the schema of the database but in other terms:

Example 1 Assume a credit card agency stores its data in the CREDIT_CARD database that contains CUSTOMER and TRANSACTION files. The schemas of these files are shown in Fig. 1. The CUSTOMER file stores all the information about the credit card customers, and the TRANSACTION file stores all the information about the transactions performed by these customers, such as customer name, merchant's name, merchant's type, and the amount and the date of a transaction.

□

An executive at the credit card agency may be interested in the spending patterns of various types of customers. In particular, he or she may be interested in the spending patterns of yuppies in expensive restaurants during a recession. Note that the terms "yuppie," "expensive restaurants," and "recession" do not directly come from the data but *can* be derived from it. For example for a credit card company, a yuppie can be a person whose age is less than 35 and who makes more than $80000, or who has a Gold card. This means that "yuppie" is a *view* on the relation CUSTOMER.

To generalize this observation, patterns of interest to the user should be defined in *user-defined* terms, or, in database terms, as *relational views* of the original data. This user-defined terminology, such as "yuppies" and "expensive restaurants," should be stored in a *data dictionary* that defines this terminology in terms of the database schema. A data dictionary incorporates the following:

Center for Digital Economy Research
Stern School of Business
Working Paper IS-92-11

		Views for Relation CUSTOMER
yuppie(name)	::=	age < 35 and income > 80000 or card_type = 'Gold'
Wall_street_yuppie(name)	::=	yuppie(name) and profession = "Investment Banking"
Madison_Av_yuppie(name)	::=	yuppie(name) and profession = "advertising"
senior_citizen(name)	::=	age > 65
student(name)	::=	profession = 'student'
customer_type(name)	::=	CUSTOMER(name,addr,income,profession,age,card_type)
		Views for Relation TRANSACTION
merchant(type)	::=	TRANSACTION(name,merchant,type,amount,date)
department_store(type)	::=	type = "department_store"
restaurant(type)	::=	type = "restaurant"
expensive_restaurant(merchant)	::=	restaurant(type) and amount > 150
moderate_restaurant(merchant)	::=	restaurant(type) and 40 < amount and amount < 150
inexpensive_restaurant(merchant)	::=	restaurant(type) and amount < 40
eco_condition(date)	::=	TRANSACTION(name,merchant,type,amount,date)
yearly_purchas(date)	::=	SUM(amount) grouped_by year(date)
recession(date)	::=	yearly_purchas(date) < yearly_purchas(date - 1yr) and yearly_purchas(date - 1yr) < yearly_purchas(date - 2yr)
boom(date)	::=	yearly_purchas(date) > yearly_purchas(date - 1yr) and yearly_purchas(date - 1yr) > yearly_purchas(date - 2yr)

Figure 2: Vocabulary for CREDIT_CARD Application.

the *vocabulary* containing a set of user-defined predicates or views, a classification hierarchy, and a set of abstraction functions. We now define each of these concepts and their specific roles.

2.1 Vocabulary

The vocabulary consists of a set of *user-defined predicates.* A user-defined predicate over a database is defined as a disjunction of conjunctive clauses, where each atomic formula is either a database relation, or a condition involving attributes from database relations or another previously introduced user-defined predicate. For instance, the vocabulary for the credit card application from Example 1 can have the user-defined predicates presented in Fig. 2[1].

Several comments about the predicates defined in Fig. 2 are in order. First, in most of the user-defined predicates we omitted the relation(s) to which various attributes belong. For example, `yuppie(name)` should really be defined as

[1] In this simplified example, we assume that a recession occurs when the yearly purchases decline over two consecutive years, and a boom occurs when yearly purchases increase over two consecutive years.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-92-11

```
CUSTOMER(name,addr,income,profession,age,card_type) and ( age < 35 and
income > 80000 or card_type = 'Gold' )
```

We omitted these relations to avoid excessive cluttering of the figure and assume that these relations can be identified from the attributes used in the specification of user-defined predicates. Second, note that the predicate `yuppie` is defined as a disjunction of conjunctive clauses. Third, the predicate `Wall_street_yuppie` is defined in terms of another user defined predicate, `yuppie`. However, definitions of user-defined predicates should be acyclic. Fourth, predicate `yearly_purchas` is defined as an aggregation of individual purchases over a year by using the *aggregation function* SUM.

It follows from the definition that a user-defined predicate is similar to a relational view. This means that all the predicates from Fig. 2 can be defined as views on relations CUSTOMER and TRANSACTION.

2.2 Classification Hierarchy

The user-defined predicates introduced in Section 2.1 are grouped into a classification hierarchy. We can impose a partial order on all the predicates in the vocabulary based on the logical implication, i.e. $P < P'$ if P logically implies P'. For example, `Wall_street_yuppie` < `yuppie`.

Based on this partial order, we can build a classification hierarchy of user-defined predicates. The classification hierarchy for the vocabulary from Fig. 2 is shown in Fig. 3. Notice that siblings may not be mutually exclusive in a hierarchy. For example, a senior citizen can (sometimes) be a student.

However, we assume that the children of a user-defined predicate in the hierarchy form a collectively exhaustive set for the parent. This can be achieved by *implicitly* assuming an extra predicate *other* for each node in the hierarchy as the "catch all" condition. For example, we can implicitly define *other_yuppies*, *other_merchants*, etc.

The hierarchy enables the user to specify the level of analysis at which the system should focus. For example, a marketing manager can be interested in patterns at a national level, whereas a branch manager might be interested in a specific region. Anand and Kahn [AK92] provide examples of the varied types of analyses of scanner data from supermarkets that are of interest to different managers and sales personnel of consumer product companies.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-92-11

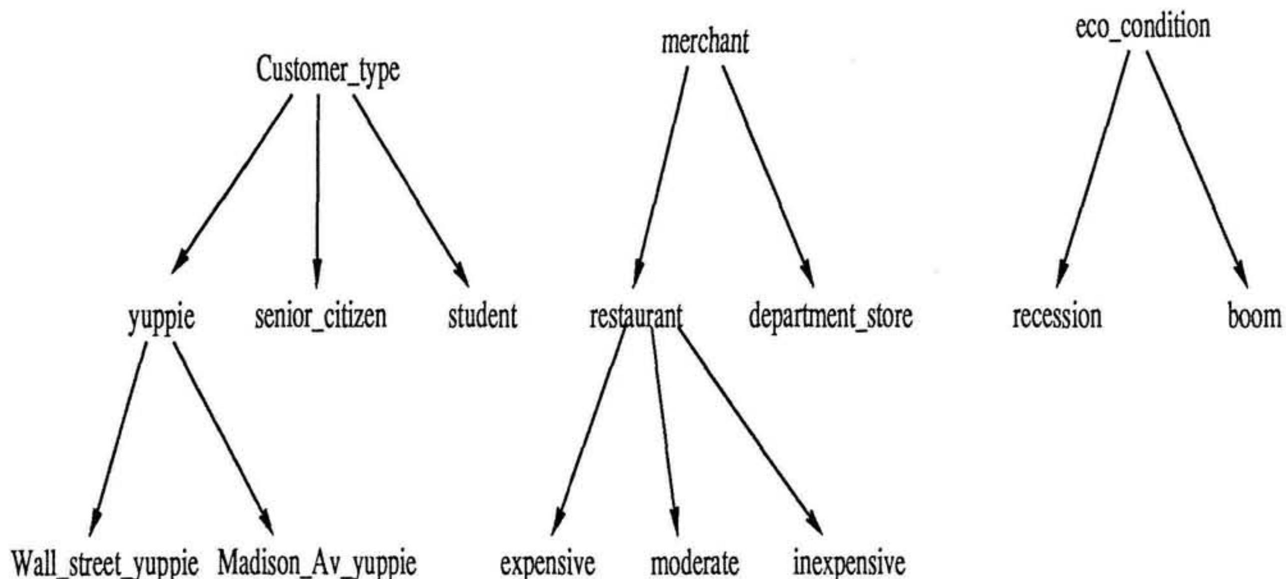

Figure 3: Classification Hierarchy for CREDIT_CARD Application.

2.3 Abstraction Functions

The third component of the data dictionary is the set of user-defined abstraction functions. An *abstraction function* of an attribute maps the domain values of the attribute into some other domain. For example, the abstraction function *year* maps a date into a year by "extracting" the year from the date. Similarly, *city* function extracts the name of the city from a street address.

Furthermore, abstraction functions can be grouped into *abstraction hierarchies* by forming composition of abstraction functions. Examples of some of the abstraction hierarchies are presented in Fig. 4.

As a conclusion, a data dictionary for searching patterns contains the vocabulary consisting of user-defined predicates, a classification hierarchy based on the vocabulary, and a set of abstraction functions grouped into abstraction hierarchies. In the next section, we show how patterns are defined based on the data dictionary.

3 Patterns

As mentioned in the introduction, [FPSM91] provide a very general definition of a pattern as some sort of a "data compression." According to this definition, a mean and a variance of a sample can be a pattern. In this paper, we adopt a more specific definition of a pattern, namely, as a rule

Center for Digital Economy Research
Stern School of Business
Working Paper IS-92-11

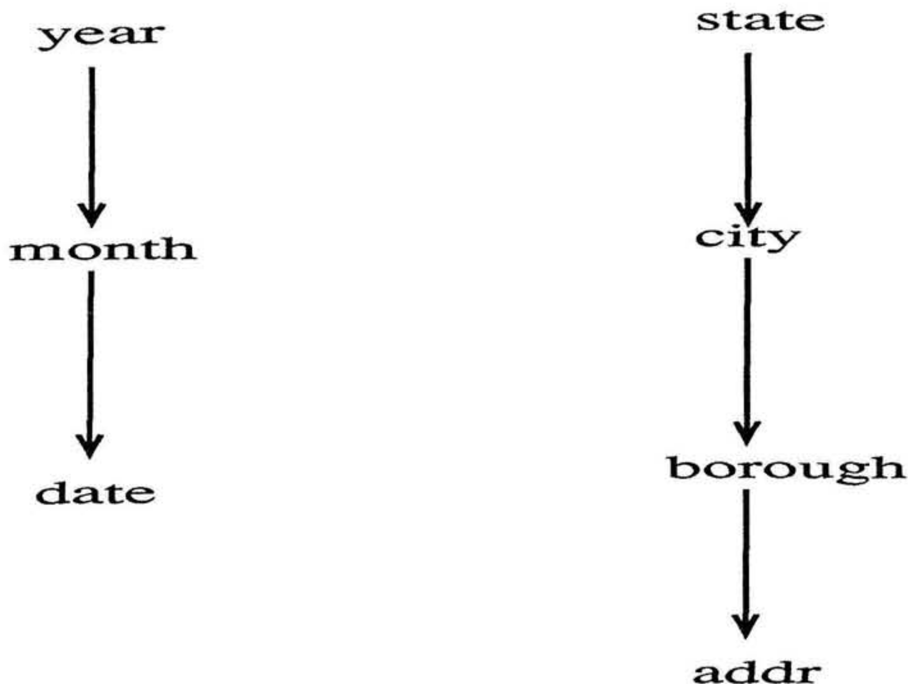

Figure 4: Abstraction Hierarchy for *Date* and *Addr* Attributes of the CREDIT_CARD Database.

expressed in terms of the vocabulary with some degree of likelihood attached to it. In Section 6, we extend this definition to patterns with aggregates.

A pattern is defined as

$$P_1 \text{ and } \ldots \text{ and } P_n \rightarrow Q \qquad \text{(with likelihood } p\text{)} \tag{1}$$

where P_i, $i = 1, \ldots, n$ are database relations, or user-defined predicates, or their negations, and Q is a relation, or a user-defined predicate, or a relational operator $=$, $<$, $\leq$, etc. Finally, p is a "measure of *likelihood*" that a certain pattern holds. We will define it precisely after we provide some examples of patterns.

Example 2 The pattern "`New York Yuppies most likely live in Manhattan`" can be expressed as

```
CUSTOMER(name,addr,income,profession,age,card_type) and
yuppie(name) and city(addr) = ''New York''
→
borough(addr) = ''Manhattan''     (with likelihood 95%)
```


Center for Digital Economy Research
Stern School of Business
Working Paper IS-92-11

Intuitively, "likelihood 95%" means that 95% of New York Yuppies live in Manhattan. We provide a precise definition of likelihood below. Also notice that the right-hand side of the rule contains a relational operator (equality in this case).

□

Example 3 The pattern "`expensive restaurants in the Greenwich Village are mostly visited by yuppies`" can be expressed as

```
CUSTOMER(name,addr,income,profession,age,card_type) and
TRANSACTION(name,merchant,type,amount,data) and
expensive_restaurant(merchant) and city(merchant) = ''New York'' and
school_district(merchant) = ''Greenwich Village''
→
yuppie(name)     (with likelihood 60%)
```

In this example, we also used the relation CUSTOMER besides the relation TRANSACTION because `yuppie` is defined in terms of that relation. Also, notice that the right-hand side of the rule contains a relational predicate.

□

The *likelihood* of a pattern can be defined in several ways. One way is to define it as a *conditional probability* that the head of a rule is true given that the body of the rule is true. In other words, the likelihood of the rule $p \rightarrow q$ is

$$\begin{aligned} P(q \text{ is true} \mid p \text{ is true}) &= \frac{P(q \text{ is true and } p \text{ is true})}{P(p \text{ is true})} \\ &= \frac{\text{number of tuples satisfying conditions } p \text{ and } q}{\text{number of tuples satisfying condition } p} \end{aligned} \tag{2}$$

For example, if there are 100,000 yuppies who live in New York and 95,000 of them live in Manhattan then the likelihood of the rule from Example 2 is 95%.

It is also possible to do more sophisticated statistical analyses of patterns in the database by assuming that its data is a *sample* of the "real" population. In this case, the proportion of yuppies becomes an *estimator* [Cli87] of the true mean. This makes it necessary to compute the *sampling error* [Cli87] associated with the samples. For example, we can say that the likelihood that New York yuppies live in Manhattan is 95% ± 2%, where 2% is the sample error. The sampling error

Center for Digital Economy Research
Stern School of Business
Working Paper IS-92-11

can be computed using standard statistical methods [Cli87] which are beyond the scope of this paper.

Since user-defined predicates in patterns are defined in terms of the database relations, it means that the patterns can be converted into rules defined in terms of database relations and abstraction functions. For instance, the pattern from Example 2 can be converted to

```
CUSTOMER(name,addr,income,profession,age,card_type) and
(age < 35 and income > 80000 or card_type = 'Gold') and city(addr) = ''New York''
→
borough(addr) = ''Manhattan''      (with likelihood 95%)
```

However, this type of a rule is usually less useful to the executive who is more interested in patterns defined in his or her terms, such as `yuppies`, `expensive restaurants`, and `recession`. In fact, much of the statistical analyses that organizations apply to data (such as cluster and factor analyses) are directed at imposing such labels on the data in order to interpret them parsimoniously.

Since the data can contain billions of different patterns in general, it is important to provide methods that limit the search for patterns. Our approach is to let the user first specify the "objects of interest". For example, the user can indicate to the system that he or she is interested in the patterns concerning eating habits of `yuppies` at `expensive restaurants` measured in terms of average or total spending, frequency, and so forth. After the user examines the patterns discovered by the system, he or she may ask the system to find other types of patterns, e.g. concerning eating habits of `yuppies` at `expensive restaurants` during `recessions`. We shall describe the pattern extraction procedure in Section 5.

4 Abstracts

As mentioned above, one way to limit the search for interesting patterns is to let the user specify, in broad terms, the types of patterns of interest. In particular, the user has to specify three types of information:

- the list of relational attributes and/or user-defined predicates the pattern should contain
- the list of abstraction functions the pattern should contain
- *aggregation principle* (or aggregation function).

User-defined predicates and abstraction functions were defined in Section 2. An *aggregation principle* specifies how observations in patterns of interest should be aggregated. The user can provide

Center for Digital Economy Research
Stern School of Business
Working Paper IS-92-11

	LOCATION		
CUSTOMER_TYPE	*CITY*	*BOROUGH*	*COUNT*
yuppie	New York	Manhattan	95,000
yuppie	New York	Queens	1,000
yuppie	New York	Brooklyn	3,500
yuppie	New York	Bronx	500
senior_citizen	New York	Manhattan	450,000
yuppie	Boston	Brookline	9,000
student	Los Angeles	Hollywood	2,000

Figure 5: Example of an Abstract.

only one aggregation principle at a time. For example, the user may specify `yuppies` as a user-defined predicate, `city` (restricted to New York), `borough` as abstraction functions, and *count* as an aggregation principle. This specification means that the user is interested in the living patterns of New York yuppies expressed in terms of counts (e.g. most of New York yuppies live in Manhattan, i.e. the total *count* of Manhattan yuppies is greater than that of yuppies living in other boroughs).

In this example, we considered counting as an aggregation principle. Examples of other aggregation principles are *summation*, *averaging*, *maximizing*, and *minimizing*. They will be discussed further in Section 6.

One way to extract patterns from the data is to group them based on the user provided terms from the data dictionary, and then compare different groups of data. For example, if the user is interested in the living patterns of New York yuppies, we could first group the data based on CUSTOMER_TYPE, CITY, and BOROUGH as is shown in Fig. 5. Then we could extract patterns from this table by comparing patterns of living for New York yuppies with patterns of living of other customer types living in other cities.

We call this kind of a table an *abstract* because it summarizes and abstracts the data in terms of high-level categories. For example, the first row in Fig. 5 says that there are 95,000 yuppies living in the borough of Manhattan in New York. Note that the atomic entities stored in an abstract in Fig. 5 are the user-defined *predicates* `yuppies` and `students`. Note that an abstract can also be considered as a report [PS91]. It aggregates data in a form that can be useful to an executive. In fact, much of the consolidation of data involved in management reporting systems involves the generation of abstract-like tables which provide useful summaries of the data.

An abstract is obtained from the data and the user-specified inputs consisting of predicates, abstraction, and aggregation functions as follows. Initially, database relations that were referenced

Center for Digital Economy Research
Stern School of Business
Working Paper IS-92-11

in the user-specified inputs have to be joined together into a single relation which we will call an *underlying* relation. This can be achieved either by making a universal relation assumption [KKF+84] or by explicitly joining relations based on the joins specified in user-defined predicates. For example, relations CUSTOMER and TRANSACTION should be joined on the field `name` for the CREDIT_CARD database to obtain the underlying relation. An abstract is constructed for the underlying relation and for the user-provided inputs as follows:

1. For each user-defined predicate and each abstraction function, create a column in the abstract. Furthermore, create an aggregation column based on the user-specified aggregation principle.

   For example, if the user specified `yuppies` as a user-defined predicate, `city` and `borough` as abstraction functions, and *count* as an aggregation principle, then the abstract has four columns, one for each of the inputs.

2. For each user-defined predicate selected by the user, consider all of its siblings in the classification hierarchy described in Section 2.2. For each aggregation function, determine its range. Form the Cartesian product of the sets of siblings for the user-defined predicates and the ranges of all the aggregation functions.

   For example, as is shown in Fig. 3, the siblings of the predicate `yuppies` are `senior_citizens` and `students`, and their parent is `customer_type`. The abstraction function `city` defines the set of all the cities in the USA, and `borough` the set of all boroughs in these cities. Take the Cartesian product of all the combinations of all the customer types in all the boroughs in all the cities.

3. For each tuple t from the Cartesian product obtained in Step 2, retrieve all the tuples from the underlying relation satisfying the conditions of tuple t[2].

   For example, consider the tuple (`yuppie, New York, Manhattan`) from the Cartesian product obtained in Step 2. Based on this tuple, select from the underlying relation all the yuppies that live in the borough of Manhattan in New York.

4. Aggregate the values of the tuples selected in Step 3 based on the aggregation principle specified by the user. If the value is 0, then the corresponding tuple is removed from the abstract.

   For example, if the aggregation principle is counting, then the aggregation field contains the number of customers belonging to a certain customer type who live in a certain borough in

[2]Notice that we consider two *different* types of tuples. Tuple t comes from the abstract, and the other type of tuple from the underlying relation.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-92-11

a certain US city. For example, if there are 95,000 yuppies living on Manhattan then we get a tuple (`yuppies, New York, Manhattan, 95000`). Furthermore, if no yuppies live in Bismarck, North Dakota then the tuple (`yuppie, Bismarck,` x`, 0`) will not appear in the abstract for any borough x in Bismarck.

The method described above only illustrates the principle of constructing an abstract and does not address any efficiency considerations. It is highly inefficient because it builds the Cartesian product of all the columns and then selects tuples with the non-zero aggregation column. The discussion of efficient implementations is beyond the scope of this paper.

5 Deriving Patterns From Abstracts

Since an abstract is just a relational table containing aggregated data, it can be used for discovering statistical patterns using standard statistical methods, such as regression, cluster, and discriminant analysis [Cli87]. In this section, we present one method, and mention others in the next section.

Patterns can be derived by "fixing" all the attributes in the abstract except one and comparing aggregated values across the "unfixed" attribute. We call the first type of an attribute *fixed attribute*, and the second type *free attribute*. For instance in Fig. 5, we can fix attributes CUSTOMER_TYPE to be "yuppie" and CITY to be "New York." We then compute the *conditional probabilities* of yuppies living in different boroughs. If `NY_yuppie` is defined as

$CUSTOMER_TYPE = \texttt{yuppie}$ and $CITY = \texttt{New York}$

then the conditional probability that a yuppie lives in borough x of New York is

$$P(x) \;=\; P\{\texttt{NY_yuppie} \text{ and } \texttt{BOROUGH} = x \mid \texttt{NY_yuppie}\}$$

Substituting the numbers from Fig. 5, we obtain the following conditional probabilities:

P(Manhattan) = 95%
P(Queens) = 1%
P(Brooklyn) = 3.5%
P(Bronx) = 0.5%

In general, if r is a condition describing the values of fixed attributes and q is the condition describing a free attribute and if p is the conditional probability, i.e.,

$$P\{q \text{ is true} \mid r \text{ is true}\} \;=\; p$$

then we can obtain the rule

Center for Digital Economy Research
Stern School of Business
Working Paper IS-92-11

$r \rightarrow q$ (with likelihood p)

For example, if r is the condition `NY_yuppie` and q is `BOROUGH` $= x$ then, since P(Manhattan) = 95%, we obtain the pattern

`NY_yuppie` $\rightarrow$ `BOROUGH = Manhattan (with likelihood 95%)`

or in words:

New York yuppies most likely live in Manhattan (with likelihood 95%)

If we expand `NY_yuppie` in the previous rule then we obtain

```
CUSTOMER(name,addr,income,profession,age,card_type) and
yuppie(name) and city(addr) = ''New York''
→
borough(addr) = ''Manhattan'' (with likelihood 95%)
```

An example of another pattern obtained in a similar way is

```
CUSTOMER(name,addr,income,profession,age,card_type) and
yuppie(name) and city(addr) = ''New York''
→
borough(addr) = ''Bronx'' (with likelihood 0.5%)
```

or in words:

New York yuppies most unlikely live in the Bronx (with likelihood 0.5%)

It follows from this discussion that we associate the *likelihood* of a pattern with the conditional probability that the pattern holds.

Notice that in this pattern the head of the rule contains the equality predicate because the free attribute is based on the aggregation function BOROUGH. If it were based on a user-defined predicate, then the derived pattern would have a predicate in its head.

Patterns derived in this section are based only on the counting aggregation principle because counting is used to compute the likelihood of a pattern. In the next section, we describe other types of patterns that use other aggregation principles besides counting.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-92-11

6 Patterns with Aggregates

In Section 3, we defined patterns of the form (1). These expressions define patterns in terms of the original, non-aggregated data. In Section 4, we used abstracts only as a *technique* to derive these patterns, since patterns do not "depend" on abstracts (there is nothing in the pattern that forces it to use abstracts). In fact, these patterns can be derived with other statistical techniques, such as discriminant and cluster analysis [Cli87].

In this section, we consider other types of patterns, which require aggregation of the original data. For example, the pattern

> `Yuppies tend to spend more money on expensive restaurants during boom periods than during recessions.`

has aggregation "built" into it (total amount of money per year).

To discover aggregated patterns, we solicit inputs from the user as for non-aggregated patterns. For example, the user may be interested in cumulative spending patterns of yuppies over the years. In this case, he can indicate an interest in user-defined predicates `yuppies`, `expensive restaurants`, and `recessions`, the abstraction function `year`, and the aggregation principle *summation* for the field `amount` in the relation TRANSACTION.

Based on the user inputs, an abstract is generated as described in Section 4. For the user inputs provided above, an abstract specifying how much a group of customers spends in different types of restaurants during different types of economic conditions is shown in Fig. 6.

Once an abstract is generated, we can extract patterns from it. However, unlike patterns described in Section 3, patterns with aggregates are expressed *in terms of an abstract.* For example, the last pattern can be defined as

```
SPENDING(yuppie,expensive-restaurant,recession,X,N1) and
SPENDING(yuppie,expensive-restaurant,boom,NEXT_BOOM(X),N2)
→
N2 > N1      (with likelihood 95%)
```

where `SPENDING` is the name of the abstract from Fig. 6, X is a variable over the attribute YEAR, NEXT_BOOM is a function specifying the next boom year, and N1 and N2 are the cumulative spendings for yuppies in expensive restaurants during the corresponding years.

Notice that SPENDING in this pattern is the name of an *abstract*, and the pattern is expressed in terms of properties of this abstract. Therefore, abstracts play a crucial role in expressing ag-

Center for Digital Economy Research
Stern School of Business
Working Paper IS-92-11

SPENDING				
CUST_TYPE	*REST_TYPE*	*ECO_CONDITION*	*YEAR*	*TOTAL_AMOUNT*
senior citizens	expensive	recession	1980	1M
senior citizens	moderate	recession	1980	1.8M
senior citizens	inexpensive	recession	1980	0.9M
students	expensive	recession	1980	40K
students	moderate	recession	1980	180K
students	inexpensive	recession	1980	420K
senior citizens	moderate	boom	1986	2.4M
students	moderate	boom	1986	560K
yuppies	expensive	boom	1981	7.3M
yuppies	expensive	recession	1983	6.7M
yuppies	expensive	boom	1987	9.5M
yuppies	expensive	recession	1991	7.4M
yuppies	moderate	recession	1980	4.5M
yuppies	inexpensive	recession	1980	3M

Figure 6: An Abstract for Extracting Patterns with Aggregates.

gregated patterns. Also, as was pointed out in Section 4, user-defined predicates become atomic values in abstracts. Therefore, they are also treated as atomic values in aggregate patterns based on abstracts. For example, `yuppie` is an atomic value in the pattern presented above, as opposed to the user-defined predicate in the patterns presented in Examples 2 and 3.

The general structure of an aggregated pattern is

$$P_1 \text{ and } \ldots \text{ and } P_n \rightarrow Q \qquad \text{(with likelihood } p\text{)}$$

where Q, P_i, for $i = 1, \ldots, n$ are either abstract predicates or relational operators, or their negations. However, we will restrict our attention to the following important special case of an aggregated pattern

$$P(a_1, \ldots, a_m, b_1, \ldots, b_k, x_1, \ldots, x_n, N1) \text{ and } P(a_1, \ldots, a_m, c_1, \ldots, c_k, x_1, \ldots, x_n, N2) \rightarrow N1\ \theta\ N2 \quad \text{(with likelihood } p\text{)} \tag{3}$$

where P is an abstract, $a_1, \ldots, a_m, b_1, \ldots, b_k, c_1, \ldots, c_k$ are constants from the abstract P, $x_1, \ldots, x_n$ are variables, $N1$ and $N2$ are values for the aggregation column in the abstract, and θ is a comparison operator $=$, $<$, $\leq$, etc., and p is the likelihood with which the pattern holds. This pattern means that for all $x_1, \ldots, x_n$, the rule holds with likelihood p, where likelihood is defined as in Section 3 either in terms of conditional probabilities or with any other statistical method.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-92-11

In the previous example, `yuppie`, `expensive-restaurant`, `recession`, and `boom` are constants, X is a variable, and $N1$ and $N2$ are values for the total amount spent per year. The likelihood of the SPENDING pattern can be defined as follows. Let n_1 be the number of business cycles when yuppies spent more money on expensive restaurants during the boom period of the cycle than during the recession period. Let n_2 be the total number of business cycles. Note that n_1 and n_2 can easily be computed from the data in the abstract in Fig. 6. Then the likelihood of the SPENDING pattern can be defined as a conditional probability $p = n_1/n_2$.

It should be noted that there could be many ways to derive the likelihood in the SPENDING rule above besides using conditional probabilities. In particular, one could apply the standard statistical methods, such as regression analysis and ANOVA [Cli87], to show the relationship between the attributes of an abstract. The spending rule above could be derived by running ANOVA on the abstract from Fig. 6. ANOVA will show how much of the variance in the dependent variable TOTAL_AMOUNT is *explained* [Cli87] by keeping some of the attributes in the abstract fixed (e.g. `yuppie` and `expensive restaurant`) and letting the remaining ones vary.

To summarize, we defined aggregated patterns in this section as rules expressed in terms of an abstract. We also discussed how the likelihood of a rule can be defined either in terms of conditional probabilities or using existing statistical methods.

7 Related Work

A recent book by Piatetsky-Shapiro and Frawley [PSF91] contains a collection of articles on pattern discovery. It presents various approaches to this important problem ranging from purely statistical approaches to the knowledge-based methods. We compare some of these methods and other related work to our approach to pattern discovery in this section.

There has been much work done in the area of pattern discovery in the scientific arena. However, there are some fundamental differences between commercial and scientific data, in the types of patterns that one is trying to discover, and in the methods for doing so. First, much of the business data is qualitative or categorical, not numeric. It is not collected in a controlled manner, but is a by product of decisions about what data is necessary for business functions. Secondly, the patterns in a large business database tend to be inherently fuzzy, not precise mathematical relationships as in the natural sciences. It therefore makes more sense to use statistical techniques to test the hypotheses instead of state space search or explicit enumeration techniques. Finally, the criteria for deciding what is "interesting" in scientific domains, which is the generator part of the generate-and-test, tend to be theory-based as in AM [Len77] and BACON [LBS81]. In the busi-

Center for Digital Economy Research
Stern School of Business
Working Paper IS-92-11

ness arena, as illustrated in the examples, executives are usually interested in trends dealing with changes in aggregate-based functions, such as totals and averages, for the terms in their vocabulary that they are interested in investigating. This information provides the "interestingness" heuristics for focusing the hypothesis generator.

There are also other techniques in the machine learning literature referred to as "learning from examples" techniques such as Winston's learning program [Win75], Mitchell's LEX system that works with "version spaces" [MKKC86], and Quinlan's ID3 algorithm [Qui86]. Of these, the ID3 is most closely related to our work. It tries to generate a decision tree that explains the data. For example, given a table with information on yuppies, expensive restaurants, and spending amounts, it would generate a decision tree where each node would involve a test on a particular attribute, in effect partitioning the data. The leaves of the tree contain the classification, such a low spenders and high spenders.

One of the limitations of ID3 is that it does not deal well with noisy data. Specifically, the tree becomes overly complicated in order to account for the noisy instances. A related problem is that it cannot deal with inconclusive data, that it, there are no rules that classify all possible examples correctly using only the available attributes. Uthurusamy et.al. [UFS91] propose that the solution to this problem is to use probabilistic rather than categorical rules. This essentially makes it a statistical approach, in the same spirit as our method which makes use of likelihoods.

As we mentioned at the outset, our use of abstracts builds on the work of Walker [Wal80]. The concept of an abstract as described by Walker was independently rediscovered by Cai, Cercone and Han [CCH91] who term this method as "attribute oriented generalization". As is apparent from the examples, our approach generalizes on the above in that attribute values in our abstracts can also be predicates or views of the original database that depend on multiple attributes. This makes it possible to derive patterns in terms of the user defined vocabulary that are difficult to discover based on attribute-oriented generalization alone.

More recently, Krishnamurthy and Imielinsky [KI91] have proposed a procedure for discovering patterns based on iterative querying of a database until the user feels that an interesting pattern is discovered. In contrast, our approach makes use of the abstracts based on user input as a basis for discovering interesting patterns.

The work of Shum and Muntz [SM88] is also related, although in a more peripheral way. They make use of a generalization hierarchy in order to determine how responses to queries can include aggregate level concepts (from the hierarchy) such that the amount of information conveyed to the user is maximized (using entropy measures).

Center for Digital Economy Research
Stern School of Business
Working Paper IS-92-11

8 Conclusion and Future Work

In this paper, we presented a method to discover a certain class of patterns based on the concept of an abstract. This method assumes that the user will guide the search for interesting patterns by specifying the types of patterns he or she is interested in. Therefore, it is the *user's* responsibility to specify what an "interesting pattern" means. Nevertheless, the system can still discover numerous uninteresting patterns such as "yuppies living in Manhattan live in New York."

We are currently working on extending our approach to incorporate the concept of interestingness in our method. The system should somehow differentiate between "interesting" and "uninteresting" patterns it discovers. In particular, the pattern discovery procedure must be able to take care of functional dependencies and avoid generating patterns that involve them. For example, the pattern presented above is not interesting because it contains the functional dependency BOROUGH → CITY (we assume that there are now two boroughs in different cities with the same name). Therefore, the pattern does not provide any interesting information beyond the known fact that a borough uniquely determines the city. Likewise, there could be other types of patterns that are not worth generating, such as the wealthy spending more than the poor. Thus the system must be able to generate additional constraints to focus the search.

We are also working on the methods that *explain* the discovered patterns. This is a problem of practical concern because executives are often aware that certain trends exist and would like explanations for them. While reasons for trends might sometimes be inexplainable using the data in the database alone, it is worthwhile determining whether the data do in fact account even partially for the observations. One approach to this is to use the notion of approximate functional dependencies [PSM91]. For example, the observation that vasectomies in a hospital are down 50% compared to last year might be explainable by the functional dependency between doctor's specialty (vasectomies) and patient's sex (the explanation could be that the doctor that performs the majority of vasectomies is currently on leave).

Finally, we still need to address the issue of how patterns should ultimately be presented to the user. Trends are probably best conveyed using line graphs and stacked bar graphs, whereas cross-sectional comparisons might be better presented using pie charts. The problem becomes more complicated where multidimensional data are involved. Some of the existing statistical tools make use of color. More sophisticated envisioning techniques have also been described by Tufte [Tuf90]. Finally, approaches such as linguistic summaries based on fuzzy sets [Yag91] could also be useful.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-92-11

References

[AK92] T. Anand and G. Kahn. SPOTLGHT: A data explanation system. In *Proceedings of 8th IEEE Conference on Applications of AI*, Monterey, CA, March 1992.

[CCH91] Y. Cai, N. Cercone, and J. Han. Attribute-oriented induction in relational databases. In G. Piatetsky-Shapiro and W.J. Frawley, editors, *Knowledge Discovery in Databases*. AAAI / MIT Press, 1991.

[Cli87] N. Cliff. *Analyzing multivariate data*. Harcourt Brace Jonanovich, 1987.

[FPSM91] W.J. Frawley, G. Piatetsky-Shapiro, and C.J. Matheus. Knowledge discovery in databases: an overview. In G. Piatetsky-Shapiro and W.J. Frawley, editors, *Knowledge Discovery in Databases*. AAAI / MIT Press, 1991.

[KI91] R. Krishnamurthy and T. Imielinski. Research directions in knowledge discovery. *SIGMOD Record*, 20(3):76–78, 1991.

[KKF+84] H.F. Korth, G.M. Kuper, J. Feigenbaum, A. Van Gelder, and J.D. Ullman. System/U: A database system based on the universal relation assumption. *ACM Transactions on Database Systems*, 9(3):331–347, 1984.

[LBS81] P. Langeley, G.L. Bradshaw, and H. Simon. BACON.5: The discovery of scientific laws. In *Proceedings of IJCAI Conference*, 1981.

[Len77] D. Lenat. Automated theory formation in mathematics. In *Proceedings of IJCAI Conference*, 1977.

[MKKC86] T. Mitchell, R.M. Keller, and Kedar-Cabelli. Explanation-based generalization: A unifying view. *Machine Learning*, 1(1), 1986.

[PS91] G. Piatetsky-Shapiro, November 1991. Personal communication.

[PSF91] G. Piatetsky-Shapiro and W. J. Frawley. *Knowledge Discovery in Databases*. AAAI / MIT Press, 1991.

[PSM91] G. Piatetsky-Shapiro and C.J. Matheus. Knowledge discovery workbench: An exploratory environment for discovery in business databases. In *Proceedings of AAAI Workshop on Knowledge Discovery in Databases*, pages 11–24, 1991.

[Qui86] J.R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-92-11

[SM88] C.D. Shum and R. Muntz. An information theoretic study of aggregate responses. In *Proceedings of VLDB Conference*, 1988.

[Tuf90] E.R. Tufte. *Envisioning Information.* Graphics Press, Cheshire, Conn, 1990.

[UFS91] R. Uthurusamy, U.M. Fayyad, and S. Spangler. Learning useful rules from inconclusive data. In G. Piatetsky-Shapiro and W.J. Frawley, editors, *Knowledge Discovery in Databases.* AAAI / MIT Press, 1991.

[Wal80] A. Walker. On retrieval from a small version of a large database. In *Proceedings of VLDB Conference*, 1980.

[Win75] P.H. Winston. Learning structural descriptions from examples. In P.H. Winston, editor, *The Psychology of Computer Vision.* McGraw-Hill, 1975.

[Win84] P. Winston. *Artificial Intelligence.* Addisson-Wesley, 1984.

[Yag91] Y.Y. Yager. On linguistic summaries of data. In G. Piatetsky-Shapiro and W.J. Frawley, editors, *Knowledge Discovery in Databases.* AAAI / MIT Press, 1991.

Center for Digital Economy Research
Stern School of Business
Working Paper IS-92-11