University of Alberta

Multi-Level Association Rule Mining: An Object-Oriented Approach based on Dynamic Hierarchies

by

Scott Fortin   Ling Liu   Randy Goebel

Technical Report TR 96–15
June 1996

DEPARTMENT OF COMPUTING SCIENCE
The University of Alberta
Edmonton, Alberta, Canada

Abstract

Previous studies in data mining have yielded efficient algorithms for discovering association rules. But it is well-known problem that the two controlling measures of *support* and *confidence*, when used as the sole definition of relevant association rules, are too inclusive — interesting rules are included with many uninteresting cases. A typical approach to this problem is to augment the thresholds with domain-specific knowledge, in the form of one or more business-goal-driven classification taxonomies. Most existing proposals use the relational approach to organize and maintain the required multi-level concept hierarchies and their domain-dependent components. We present the motivation for a necessary extension to existing uses of such multi-level hierarchies, and describe an implementation that is better suited than the typical relational paradigm for focusing the search and regulating the mining of association rules both at multi-levels within one concept hierarchy, and across multiple concept hierarchies. Our hierarchy design supports an adaptive encoding scheme for focusing the mining on semantically deeper and more informative knowledge; in essence, it dynamically generates and adjusts concept hierarchies. We demonstrate that the application of an object-oriented implementation of such a design not only provides the advantage of a flexible combination of multiple multi-level concept hierarchies for focusing the data mining task, but also provides smooth integration of multi-level concept hierarchies with legacy relational databases. In addition, by using the adaptive encoding scheme, efficient algorithms developed for discovering interesting association rules can be integrated into the our framework with no or little extra cost.

Keywords: rule mining, mulit-level hierarchies, association rules, data mining, object-oriented

1 Introduction

Data mining (knowledge discovery in databases) is motivated by decision support problems faced by most business organizations. It is a field of increasing interest to researchers in databases, artificial intelligence, and machine learning. Mining for association rules amongst items in a large database of sales transactions has been described as an important database mining problem in decision making, risk management, and the development of customized marketing programs and strategies.

Association rules, introduced by Agrawal, Imielinski, and Swami [1], are a class of simple but powerful abstractions to facilitate the understanding of large amount of data by discovering interesting regularities [8]. The problem can be stated as follows: given a database with many transactions, where each transaction consists of a number of items, find rules which associate the presence of one set of items with the presence of another set. Quite often, we wish to find all rules of form $X \rightarrow Y$ satisfying some given support and confidence threshold parameters, where X and Y are disjoint sets of items, the **support** is defined as the probability that a transaction contains both X and Y, and the **confidence** is the probability that any transaction containing X also contains Y.

Several fast algorithms have been proposed for efficiently discovering association rules, satisfying given thresholds for support and confidence [1, 2, 10, 16, 6, 9, 18]. However, these algorithms often produce a large number of rules, creating an additional knowledge management problem. This arises because in large databases, the rule sets discovered by using a flat relational table and minimum values for support and confidence are often very large and contain many uninteresting rules [17, 10]. In addition, the large sets of discovered association rules are mostly managed in relational systems as internal system data, which makes it difficult to query and further use the rules to find more interesting information with respect to particular business objectives.

Here we propose a new framework for finding interesting association rules in the context of multi-level concept hierarchies. The discovered rules can span a single level in a single hierarchy (as was done in [6]), many levels within a single hierarchy, or many levels within multiple hierarchies. A typical example of a single level association rule is "80% of customers who purchase milk also purchase bread". Not surprisingly, level and hierarchy spanning association rules, such as "50% of customers buy whole-wheat bread if they buy Dairyland 1% milk", or "45% of customers buy Dairyland Cheese if they buy Dairyland milk and a Coupon item", are often more informative and useful. We also extend the semantics of association rules to allow *conjunctive items*; for example "12% of customers buy Lucern Cheese if they buy an item which is *both* whole-wheat bread *and* On Sale". The concept of multi-level association rule mining was originally introduced by Han and Fu [6] as an extension to single concept level association rule mining, with the objective of exploring more interesting rules from large transaction databases. However, they use flat relational data structures to construct the base relations for building multi-level concept hierarchies and a fixed encoding scheme which makes it difficult to dynamically restructure the multiple concept hierarchies to meet users' diverse customization requests.

We demonstrate that the use of an object-oriented implementation for our design of efficient and flexible multi-level association rule mining not only provides advantages in the use of existing data mining algorithms, but also supports improved integration with a user query language, and seamless evolution of concept hierarchies in the presence of database updates. In addition, our framework allows the specification and application of user-defined domain knowledge to focus the mining on more interesting and informative rules and to actively monitor the mining of interesting association rules in the event of database updates and changes in users' customization requirements. More interestingly, with the adaptive construction of a dynamically encoded transaction table all these benefits come with little loss in performance, since the same mining algorithms used with the relational data can be seamlessly adapted to the object-oriented framework.

The rest of the paper is organized as follows. The section immediately following provides a motivating example, which demonstrates the need for the creation of association rules across multiple levels of multiple hierarchies. Subsequently, Section 3 briefly summarizes the challenges inherent in using the relational approach to managing and mining association rules. In Section 4 we describe the requirements for our representation of hierarchies for rule mining, and then provide a description of a suitable design and object-oriented implementation. Our design introduces an adaptive encoding scheme which can be viewed as an extension to the fixed encoding scheme proposed in [6]. We discuss the benefits of using our framework to focusing association rule mining in Section 5, compare our work with related research in Section 6, and conclude the paper in Section 7.

2 Mining rules at multiple hierarchical levels: a motivating example

Assume we are a senior executive working at a large chain of grocery stores, and have access to the database of customer sales transactions. Figure 1 represents a portion of the concept hierarchy for the grocery items. We want to discount the price of a single milk product so that when this milk product sales increase (presumably because of the discount) the sale of bread products will also increase.

Data mining at a single concept level (i.e., the lowest level in the hierarchy only) will not be very useful in this case. This method, used for association rule mining at single concept level, would return rules of the form "Dairyland skim milk $\rightarrow$ Wonder whole wheat bread". Obviously the

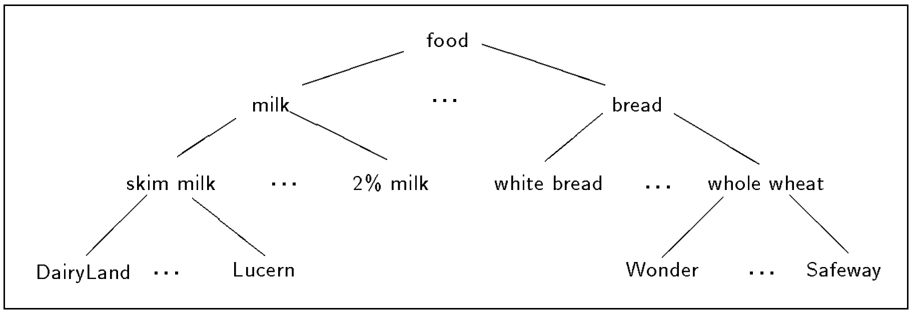

Figure 1: The example concept hierarchy

discovered rules of this form would only give us the milk product which is most likely to cause an increase in *a specific bread product*. We are looking for a milk product which gives us the maximal increase over all types of bread.

Consider mining at multiple concept levels [6]. If each rule is restricted to associate only items at the same level of a concept hierarchy, then even though the mining algorithm [6] would search over each level of the entire hierarchy, we would only be able to obtain rules such as "milk $\rightarrow$ bread" or "skim milk $\rightarrow$ whole-wheat bread". However, what we want is to be able to find a specific milk product to discount, which would best boost the sales of all types of bread.

We, for this type of knowledge discovery, clearly need to mine association rules which span multiple levels. More concretely, we wish to find rules that are strong in association of items between the bottom level concepts in the subtree rooted at milk and the top level concept of the subtree rooted at bread. This would allow us to find rules of the form "Dairyland skim milk $\rightarrow$ bread", and then we could choose the product with the highest probability of causing an increase in bread sales.

Another incentive is to mine rules across many multi-level concept hierarchies. For example, we may realize that we also have access to promotional data from the previous month. That is to say, we can organize the items into *another* hierarchy, not based on the category of product contents but instead on their promotional status. Using both hierarchies we are able to discover rules like "if a customer buys a milk product that is on sale, then the customer will usually also buy Wonder whole wheat bread". These rules would help us more closely predict the outcome of putting a milk product on sale. The executive may even decide to increase the price of "Wonder whole wheat bread" to help recoup the losses.

3 Challenges within the relational paradigm

The relational model has evolved to be the standard for efficient management of tabular extensional data. The interest in automatically (or semi-automatically) creating "interesting" rules requires the efficient representation and manipulation of concept hierarchies, and this need has arisen much more quickly than the relational paradigm has evolved to meet the demand. Previous work on solving the problem of managing the potentially large rule sets discovered through known algorithms [1, 2, 16, 18] includes techniques such as templates [10], metaqueries [19, 5] and mining rules at multiple concept levels [6]. All these techniques are confined to the relational domain, in the sense that the concept hierarchies and/or domain-dependent knowledge are encoded as relations. In addition to the need to invent new relations and attributes to encode the hierarchical structures, there are

barcode	category	brand	content	size
14998	milk	Diaryland	skim	2L
12998	mechanical	MotorCraft	valve 23a	12 in
...	...	...	...	...

Figure 2: An example sales_items relation

other difficulties:

(1) Weak Item Representation

One obvious disadvantage of a relational approach is the semantic straight-jacket the database designer has to wear when deciding how to represent information about the database items. Using the relational approach, every item must fit into a single flat relation to help it be classified by the rule mining software. For example, Figure 2 shows a possible relation called sales_items which might be used as a base relation for association rule mining in a retail organization. By placing each item into such a relation, every item is required to have exactly the same attributes. If we wanted to add an attribute "expiry_date" for milk items, we would have to include that attribute for every item in the database — even though for certain items (such as valves) the expiry date has no real meaning.

(2) Anomalies of Relational Mappings of Concept Hierarchies

As pointed out in [6], the relational tables such as sale_items in Figure 2 can only represent the lower levels in the concept hierarchy. The higher level concepts, such as "milk is a type of food", "apples and pears are a type of fruit", "peanut, pistachios, ..., walnuts are a type of nuts", etc., may not be explicitly or uniformly present in the form of relational tables; these often have to be manually specified by experts as domain-specific knowledge. A table or set of tables is typically created [6] to map high level concepts to actually tuples in a relation like sale_items. These tables are essentially internal data structures created and maintained by the the mining software.

This creates an immediate problem for user queries. Suppose that we want to capture the knowledge that peanut, pistachios, ..., walnuts $\subset$ nuts, and there is no place in the relational tables of the database where the concept of "nuts" is explicitly mentioned. We would thus create a node for "nuts" in the mapping tables so data mining could proceed on this concept. After discovering an interesting rule about "nuts," a user may wish to query the database to get a report on the average price of nuts for the previous year. This operation would be quite difficult to perform. We would have to modify the relevant database programs to use the internal mapping tables of the data mining program. Such modification would almost certainly have to be re-written for different mining software, and maybe even different versions of the same software, due to varying choices in implementation of the internal data structures for mapping tables. The underlying problem is that the domain-specific knowledge, which has to be added for the mining software to work, is not natively supported by the relational architecture.

Further, the insertion of new items is problematic. When the data mining software is first installed, it is relatively easy to examine all items and make sure each one is classified somewhere in the concept hierarchy. However, at some point in the future a new item might be added by a user who has no interest in the data mining portion of the database, and thus is not responsible for

Update 1	Update 2
Move all items in category c_1 to category c_2	
	Add item i to category c_1
Delete category c_1	...
...	

Figure 3: An undesired concurrent update: Item i is lost

including the new item in the hierarchy. To include the new items into the concept hierarchies, the mining algorithms would have to be changed to detect unclassified items at run-time. In addition to the performance hit, it is not clear what could be done upon discovering an error. For example, suppose an executive involved in a data mining task discovers that valve 23a has not been placed in the hierarchy. To examine buying patterns for mechanical items, it is desirable to include all the relevant items that are newly added into the database, such as valve 23a. Unfortunately, it could be very difficult to figure out if the item belongs to the car making or dishwasher products. The creator of the item should have been forced to classify it when it was created, but this constraint is difficult to enforce in a relational setting, where the concept hierarchy is laid on top of relational tuples.

(3) Problems with Concurrent Updates of Concept Hierarchies

In relational systems, there is no support at the system level for taking a set of relations as a unit of concurrency control, in the event of updating concept hierarchies or adding new items. It is user's responsibility to ensure that the concurrent updates to the concept hierarchies and to the database are consistent and correct. For example, when the hierarchy information is being changed, updates to the item information should not be allowed. Figure 3 shows a scenario that could occur when this rule is not followed. **Update 1** is removing a category (where a category corresponds to a high level concept which spans multiple relational tables) and **Update 2** is adding a new item into a hierarchy, time goes downward. When a concept hierarchy is multi-level and thus mapped to multiple relational tables, concurrent updates may result in the loss of a new item because a multi-level concept hierarchy can not be treated as a unit of concurrency control in relational systems. It is left to the users to enforce the complex object level locking protocol that is necessary to guarantee the correctness of concurrent updates of concept hierarchies. This is another exposition that using the relational approach to create and maintain the concept classification hierarchies is more prone to error in a multi-user environment.

(4) Lack of Adequate Support for Multiple Hierarchies

It is non-trivial to mine association rules in the context of many valid methods of organizing the items, when using the relational approach. For example, retail outlets might want to categorize their items based on their *location* in the store. This would find rules of the form "prime location in row 1 $\rightarrow$ prime location in row 2", which would tell the store that shoppers who buy items which are prominently displayed in the first row tend to buy items which are prominently displayed in the second row. As another example, items might be organized based on their *promotional status* of being on sale, or regularly priced, or in a coupon book, and so forth. When supporting multiple views of how the items can be organized in terms of varying concept hierarchies, the relational approach runs into difficulty because the flat relation in which an item is stored implies a single view of classifying the item. It is quite difficult, if not impossible, to use flat relations to support multiple views of organizing the items.

4 An alternative hierarchical representation

Our motivation is to use a more natural implementation for concept hierarchies, that provides a method which helps preserve the efficiency of existing data mining algorithms, and supports the multiple-hierarchy, multiple-level extensions proposed herein. A natural choice is to augment the relational model with an external representation based on an object-oriented model. We first describe the requirements for our rule-mining system, and then provide a description of our design and implementation.

4.1 Hierarchical representation requirements for rule mining

Let I be the set of all literals, called items, and S be a set of transactions over I. Each transaction has two components, a unique transaction identifier and a subset of I which identifies the items involved in the transaction. Items in I may be related differently with respect to different domain requirements and business rules. We describe the relationships between items with respect to some pre-defined domain knowledge in a hierarchical structure, called a concept hierarchy. A concept hierarchy C may have m levels. When $m = 1$, we call C a single-level concept hierarchy. Otherwise C is called multi-level concept hierarchy.

An association rule is an expression of form $X \rightarrow Y$ where $X, Y \subseteq I$ and $X \cap Y = \emptyset$. Given an itemset $X \subseteq I$, the cardinality of X is called the *size* of X. Itemsets of size k are called k-itemsets. Two parameters, namely *support* and *confidence*, have been used to quantify the mining of interesting association rules [1]. We define the **support** of X in S, $\sigma(X, S)$, to be the ratio of the number of transactions in S containing X versus the total size of S. The **confidence** of a rule $X \rightarrow Y$ in S, $\varphi(X \rightarrow Y, S)$, is the ratio of $\sigma(X \cup Y, S)$ versus $\sigma(X, S)$. Intuitively, the support is a measure of the number of transactions in S where a rule possibly could have been satisfied, and the confidence is a measure of how many of the transactions in S actually satisfied the rule.

Given a database $< I, S >$, the process for discovering association rules can be broken into two broad steps, as noted in [2]:

1. Find all sets of items (i.e. subsets of I) which have support greater than a specified minimum support. These itemsets are called large itemsets.

2. For each large itemset, find all the rules which have confidence greater than a specified minimum confidence.

A simple algorithm for solving Step two would be as follows:

Input: LargeItemSet $\{A\}$
Output: ValidRuleSet R
Procedure:
  for each large itemset A
    for every proper subset B of A
      if $(\sigma(A)/\sigma(A - B) >$ min_confidence)
        append $(A - B) \rightarrow B$ to ValidRuleSet R
end

The first step, generating all large itemsets and their supports, is relatively straightforward. However, it is a nontrivial problem when the cardinality of the set I of items is large. If $card(I){=}n$, the number of possible distinct itemsets is 2^n. To reduce the combinatorial search space, all fast algorithms proposed so far exploit the following properties: any subset of a large itemset must also be large, and any superset of a small itemset is small. Applying the same principle to multi-level association rule mining, we say that an item i is **large** at level k of a concept hierarchy if the item i is in the k-th level of the concept hierarchy, the support of the set $\{i\}$ is greater than the pre-defined (either by user or by system) minimum support for level k, and the ancestor of i is large at level $k-1$. For example, if toys does not have the minimum support given at level 3, then we will not consider a level 4 descendent such as teddy bears large even if teddy bears satisfies the minimum support threshold for level 4.

Han and Fu [6] developed a fast algorithm for mining multi-level association rules that associate concepts at the same level of a given concept hierarchy. In order to facilitate the mining of multi-level association rules that associate items spanning several levels of a concept hierarchy, we replace the fixed encoding scheme used in Han and Fu's algorithm by an adaptive encoding scheme (see Section 4.2) which dynamically massages the items being associated to "lie" on the same level. As a result, the problem of mining association rules of form $X \rightarrow Y$, where X, Y are at any level of a concept hierarchy, can be reduced to the problem of discovering all rules $X \rightarrow Y$ where X, Y are at the k-th level of the concept hierarchy, and

- all items in X and Y are large at level k
- $\varphi(X \rightarrow Y, S)$ is greater than the minimum confidence for level k
- $\sigma(X \cup Y, S)$ is greater than the minimum support for level k.

4.2 Using an Adaptive Encoding Scheme

Input: EncodedTransactionTable ET, and support $supp_i$ threshold for each level
Output: all large itemsets for each level
Procedure:

```
for each level i:
    Cand_i ← the candidate large 1-itemsets (descendents of the previous levels' large 1-itemsets)
    scan ET and remove those candidates in Cand_i with support < supp_i
    k ← 2
    loop
        Cand_k ← generate candidates from the (k-1)-itemsets
        for each transaction t in the encoded table
            increment the support for each candidate k-itemset that appears in t
        remove those members of Cand_k who have less than supp_i support
        if Cand_k is empty then break from loop
    endloop
    LargeSets_i ← the union of all non-empty Cand_k
endfor
return the union of all LargeSets_i
```

end

Figure 4: An algorithm for mining association rules at multiple levels

As addressed in the previous section, there are cases where mining association rules which cross levels and which involve multiple hierarchies would be advantageous. In this section we discuss ways of extending previous algorithms proposed by Han and Fu [6] to handle these cases. The main idea is to use an adaptive encoding scheme, rather than the fixed encoding scheme, to encode the data items in the transaction table. This allows existing algorithms [6], for mining rules that associate items at the same level of a concept hierarchy, to be reused for discovering interesting rules that associate items at different levels of a concept hierarchy or across multiple concept hierarchies.

The design of our adaptive encoding scheme consists of two major steps. In the first step, we only consider how to generate the encoded item by scanning through the transaction table in the database, and replacing each item in a transaction with its encoded form. The encoded form for an item is a string which describes its location in the concept hierarchy. For example Food might have an encoding of "1", Bread might have an encoding of "1,2" (the 1 stands for the fact that bread is a food, and the 2 represents that bread is a particular type of food) and whole wheat bread might have an encoding of "1,2,1". The second step transforms the encoding for an item, based upon a user request. We present three different transformations, to support rules which have items from different levels in a hierarchy, from different hierarchies altogether, and from a conjunction of two or more hierarchies.

The first transformation we present supports the discovery of rules which span different levels in a single hierarchy.

> **Input**: the encoded item E, the selecting item S, a target depth $targ$, and the maximum depth max
> **Output**: a new encoded item, or NULL
> **Procedure**:
> if E is not a descendent of S then
> return NULL;
> $E' \leftarrow$ the first $targ$ characters of E
> insert $(max - targ)$ dummy characters into E' starting at position $(targ - 1)$
> return E'
> **end**

To illustrate the use of this transformation algorithm, consider this example: Suppose we want to find a rule relating specific milk products to bread. For each item in the hierarchy of Figure 1 we would call this transformation twice. The first call uses milk as the selecting item with the target depth of 4 and the maximum depth of 4. This would return non-NULL only for items which were specific milk products. A second call would be made with Bread as the selecting item and a target depth of 2 and a maximum depth of 4. This would be non-NULL only for items which descended from Bread, and would return the encoding "1,#,#,2" where # is the dummy character. Thus the encoded table would consist of specific milk products or an item which represents bread. Note that after this transformation, the milk products and bread are on the same level. Thus we can modify the existing algorithms [6] by simply using the encoded transaction table generated by the adaptive encoding scheme (see Figure 4). Obviously this slightly modified algorithm will be able to return rules of the form "Dairyland skim milk $\rightarrow$ bread".

Note that the use of an encoded transaction table makes the algorithm in Figure 4 general in nature since it does not directly access the transaction data and hence does not depend on the storage structure of the transaction data. As long as we can encode and decode an item correctly, we can arbitrarily massage the encoded string to find different types of rules. Although the research reported in [6] suggests alternative ways of handling level crossing rules, the discussion has not led to an effective and well-defined algorithm; for example, the minimum support threshold of a rule which spans levels does not clearly follow from the single level thresholds which are supplied by the user. In contrast, by using our adaptive encoding scheme, no change to the underlying semantics or algorithms is required.

There are several interesting points to note about the adaptive transformation technique we propose. We see that bread must satisfy not just the minimum support threshold for level 2, but

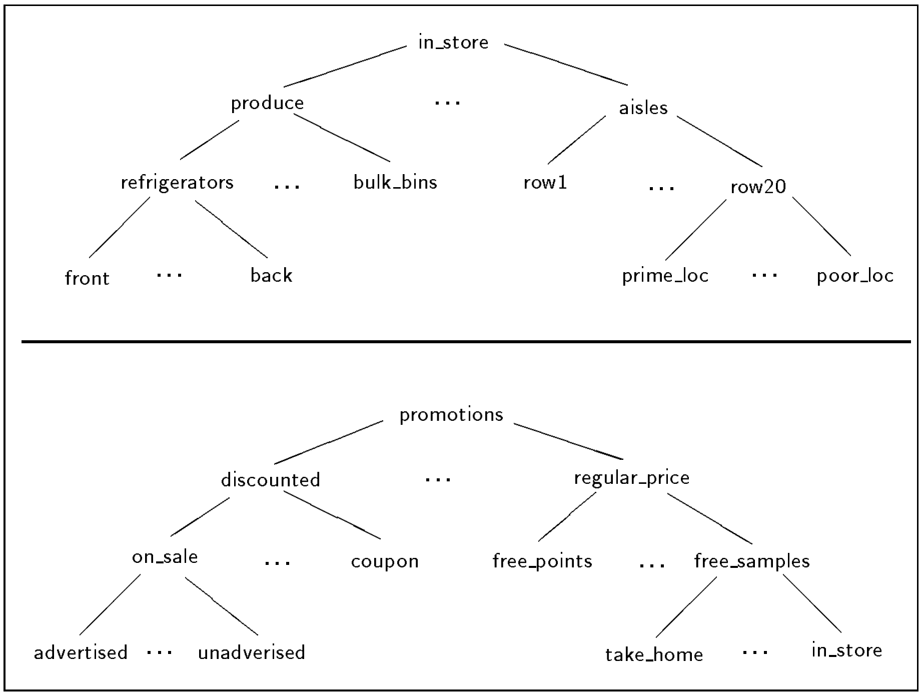

Figure 5: Hierarchies for location in store and promotional offers

also the thresholds for levels 3 and 4. If the thresholds decrease as we go down levels, then if bread survives on level 2 it will always survive to level 4 as well. However, the reverse case is not necessarily true. An item which is large at a lower level may and need not be large if the thresholds at a higher level are used. Interesting to note is that the adaptive transformation has some pruning effect on the generated rules. The level 4 rules generated will involve only specific milk products and possibly bread. The discovered rules which do not include bread, i.e. are all milk products, can be filtered out before presenting the set of interesting rules discovered to the user. Furthermore, the adaptive encoding transformation scheme can be seen as a means for mining rules that involve several types of items, and be applied for many subtrees. For example a user may say that she wanted all rules involving milk at level 3, bread at level 4, cheese at level 4, and snacks at level 2.

Using this idea of transformations, it is relatively straightforward to support multiple ways of organizing the transaction data. For example we may want to organize the item sales not only by their content categories but also by, for example, their locations in the store and/or their promotion categories. These additional organization structures are shown in Figure 5. The second transformation we introduce is very simple, and allows us to integrate the hierarchies. Assume that for every item we get n encodings, $e_1, \ldots, e_n$, corresponding to the item's position in the n different concept hierarchies. Then the transformation to find rules with items from any of the hierarchies is simply stated: prepend an i to every encoding e_i. This (along with the previous transformations) would let us find rules like "customers who buy milk and some item that is on sale usually buy whole wheat bread". These rules, while useful, would not allow us to use the relationships between the items in different hierarchies. In essence, we are finding rules like:

$$\forall x, y, z \in I \quad \mathrm{type}(x, \mathrm{Bread}) \wedge \mathrm{promotion}(y, \mathrm{OnSale}) \rightarrow \mathrm{type}(z, \mathrm{Milk})$$

Thus this rule would gain support if a bread item and *any* item on sale were together with a milk item. We may to constrain this further, and only increment the support if there is an item which is *both* bread and on sale. Thus we would get rules like:

$$\forall x, y \in I \quad \text{type}(x, \text{Bread}) \wedge \text{promotion}(x, \text{OnSale}) \rightarrow \text{type}(y, \text{Milk})$$

The third transformation we present allows us to discover rules of this form, by taking two different encodings of an item and returning a single *conjunctive item* if the item belongs to the two subtrees we are interested in. This transformation for the conjunction of two hierarchies is given below, it generalizes in clear way to a conjunction of n hierarchies.

Input: two different encoding as E_1 and E_2, and selector items S_1 and S_2
Output: a new encoded item, or NULL
Procedure:
  if E_1 is not a descendent of S_1 then
    return NULL;
  if E_2 is not a descendent of S_2 then
    return NULL;
  for $i \leftarrow 1 \ldots \text{size}(E_1)$
    $E'[i] \leftarrow \text{f}(E_1[i], E_2[i])$
    (where $\text{f}(x, y)$ is some function like: $x + 10y$)
  return E'
end

For example, suppose we have an item encoded as "1,2,1,3" and "1,4,5,1", milk is encoded as "1,2" and on sale was encoded as "1,4". Then the transformation would return the conjunctive item "11,24,15,31", where 11 represents food $\wedge$ promotion, 24 represents milk $\wedge$ on sale, and so forth. Users could use this to specify another type of item to be included in their output, which is a conjunction between two or more hierarchies. Thus a user query might look like "find me all rules involving milk items at level 4, bread items and level 2, and items which are both cheese and on sale".

The technique of transformations does not depend directly on any underlying model, it could be implemented on either a relational or object-oriented system. However, the object-oriented model allows the encoding and decoding of items to be encapsulated within the class that contains the item data, making the system easier to implement and maintain.

4.3 Design of the Rule Mining System

4.3.1 A Quick Look at the User Interface

Database mining is a complex, interactive database task that is often performed with the aid of a graphical user interface [10]. We design a user interface based on a SQL-like data mining language to represent the rule discovery queries independently from such an implementation. The following presents the syntax of our data mining user interface language designed for the association rule discovery process:

```
ASSOCIATE     <complex-items>
FROM          <tables and concept-hierarchies>
[WHERE        <conditions>]
HAVING
   [SUPPORT       <support-thresholds | DEFAULT >]
   [CONFIDENCE    <confidence-thresholds | DEFAULT >]
   INTERESTSCOPE  <interested-attributes>
[ORDER BY     <attributes>]
```

Clauses in the rule mining query expression are designed to allow the user to exploit all areas of the data mining framework. The `ASSOCIATE` clause lists those types of items which are allowed to occur in the resulting rules. Elements of this field are node names from a concept hierarchy, such as the one in Figure 1, combined with operators to specify the transformation taking place. Node names can be followed by an optional number, which refers to the level in the hierarchy under the given node which should appear in the output rules. For example, "Bread(0)" means that only Bread and no descendents should appear in the resulting rules, "Bread(1)" means that only Bread's immediate descendents may appear in the rule, and "Bread(MAX)" means only the lowest level Bread items should appear in a rule. As a shorthand, "Bread(*)" or just "Bread" stands for Bread(0), Bread(1), ..., Bread(MAX). As well, items can be joined with an AND which means to only include items which have both of the AND'ed properties.

The `FROM` clause allows the user to specify which concept hierarchies and transaction tables should be used. For example, the user might have tables of transactions based on different store locations, and thus could specify "west-side-transactions, north-end-transactions," and so forth in the from field. The `WHERE` clause is optional. It specifies properties of the individual transactions that must be met before they should be included in the output rule. The meaning of the `SUPPORT` and `CONFIDENCE` clauses are fairly straightforward in the data mining context. They allow the user to specify the values for the thresholds used for each level, or to use the symbol "*" to mean the system default for that level. The `SUPPORT` and `CONFIDENCE` clauses are also optional. When omitted, the system default will be used for all the support and confidence thresholds. The `INTERESTSCOPE` clause does not effect the data mining process directly, but allows the user to specify which properties to include when displaying the discovered rules.

For example, to find the rules that tell the sales trend of "`Bread`" associating with the sale of "`2% Milk`" in 1996, one may write a mining query as follows:

```
ASSOCIATE  2% Milk, Bread
FROM       west_end_sales_transactions w, west_end_concept-hierarchies h
WHERE      w.date.year = 1996 AND n.date.year = 1996
AND        h.nodename='Food'
HAVING
  INTERESTSCOPE category, content-spec, manufacturer
```

Finally the `ORDER BY` clause is optional and can be used to order the mining results based on the attributes involved in the discovered rules. Here is a complete example query:

```
ASSOCIATE   Bread AND On_Sale, Milk(*), Cheese(0)
FROM        all_sales_transactions t, all_concept-hierarchies h
WHERE       t.date.year = 1995 OR t.date.year = 1996
AND         h.nodename='Food' OR h.nodename='Promotions'
HAVING
  SUPPORT       0.8,*,*,*,0.1
  CONFIDENCE    default
  INTERESTSCOPE category, content-spec, brand, manufacturer
ORDER BY    Bread.price, On_Sale.discount
```

This query expression says to search those transactions occurring in 1995 and 1996 to find all association rules, with 80% support at the highest level and 10% support at the lowest level, which associate `Bread` items `on sale` and all types of `Milk` or `Cheese` items. The result of this query is a set of rules discovered from the large transaction database(s). The result is sorted by the price of the bread items and the amount of the discount.

4.3.2 Object-Oriented Implementation

In this section we present a sketch of the object-oriented implementation for the item sales transaction database, including the design of classes and methods.

A class[1] hierarchy is central to an object-oriented system. In an object-oriented database every object is an instance of some class. Associated with all objects of a certain class are a set of properties and behaviors which define the interface used to communicate with the objects of that class. Classes are related to other classes through a subclassing mechanism, where a subclass inherits all of the properties and behaviors (also known as methods) of its superclass.

The object-oriented approach to building an association rule mining system requires the definition of a number of base classes which are used to focus the mining of interesting association rules. A brief overview of the classes for the item sale database is presented as follows:

4.3.3 Item Hierarchy

```
class Item {
   public:
      String encoding_for(String internal_node);
      void add_parent(Organization org);
      void set_name(String);
      String get_name();
        ...
   private:
      List parents;
      String name;
        ...
}
```

```
class Milk : Item {
   public:
      void set_butterfat(String);
      String get_butterfat();
        ...
  private:
     String butterfat_content;
        ...
}
```

The Item class is an abstract superclass for all types of items in the database. Item and its subclasses provide two real services in the system. The first is the storage of information related to a retail store's items; for example the Milk class might contain a property which indicates whether the butterfat content of the milk is "skim", "whole", or "2%". The second function of this part of the class hierarchy is to provide services for generating encoded strings which represent that item. To this end an item maintains a list of its parents, objects which directly contain the item. These container objects are used to implement the different hierarchies and are instances of the class Organization, which is discussed in the next section. The algorithm for generating encodings is straight-forward:

Input: an encoded non-leaf item O (of length k)
Output: an encoding for myself which contains O, or NULL if none do
Procedure:
    for each of my parents p
        $E \leftarrow$ the encoding generated by following links from p up to a root
        if the first k characters of E match those of O
            return E
    endfor
    return NULL
end

Note that while the class hierarchy which is rooted at Item may resemble the top levels of a concept hierarchy such as the one in Figure 1, it is *not* the means by which such a hierarchy is implemented. Instead the hierarchy is explicitly implemented in a tree structure. This allows the inheritance of Items to factored along different lines than that of pure semantic meaning, for

[1] In this paper the terms class and type are used interchangeably when no confusion is incurred.

example we might want to have an abstract superclass "ItemWithExpiryDate" which could have subclasses that were not related in any semantic sense. As well, by encoding all hierarchies explicitly as tree structures we can support multiple hierarchies with a consistent and simple interface.

4.3.4 Organization Class

```
class Organization : Item {
   public:
      void add_child(Item i);
      void set_range(Range r);
      Range get_range();
        ...
   private:
      List children;
      Range range;
}
```

```
class Range {
   public:
      int start, stop;
}
```

Aggregation is the general object-oriented technique for creating complex objects. Basically, objects can have any number of attributes, each of which can be another object. Hence it is possible to create arbitrary nested structures through composition.

Organization is a special subclass of Item that is used to implement a concept hierarchy such as the one in Figure 1; instances of class Organization correspond to the interior nodes of the hierarchy. As such, Organizations are the only items that can have children, and if the concept hierarchy really is a hierarchy (as opposed to a lattice) then they will have exactly one parent.

An Organization's primary function is to implement a tree structure, which would model the hierarchies in Figure 1 or in Figure 5. Thus an Organization has the property "children" which is a List of its child nodes, the reverse link (to the parent node) is inherited from Item and thus is not explicitly mentioned in the Organization class. In addition to implementing a concept hierarchy, an Organization provides other miscellaneous functions. The property `range` allows one to restrict the application of an organization to a subset of all transactions. For example, a store might change its layout once a week, and thus a hierarchy based on the locations in the store would only apply for those transactions in the corresponding week. As well, Organization would contain support routines to help Items generate their encoded strings.

4.3.5 Transformers and Transaction Table

```
class TransactionTable {
   public:
      void add_transaction(Set items);
      void mine(float supp[],
                Transformer trans[]);
        ...
   private:
      void multi_mine(float supp[],
                      Set encodedTable);
      Table transactions;
        ...
}
```

```
class Transformer {
   public:
      virtual String transform(Item);
      Item decode(String);
        ...
}
```

```
class MultiLevelTransformer: Transformer {
   public:
      String transform(Item);
      Organization S;
      int targ, max;
```

```
class ConjunctionTransformer: Transformer {
   public:
     String transform(Item);
     List organizations;
}
```

```
    ...
}
```

The Transformation classes are used to implement the transformation routines described in Section 4.2. The subclasses of Transformer contain the specific properties needed to compute their transformation. For example the transformation to make items on different level "line up" needs to know the subtree that it is to be applied to, the target depth, and the maximum depth; these are all properties of the MultiLevelTransformer class. Similarly the ConjuctionTransformer has to know the list of hierarchies that it is conjuncting together in order to compute its transformation function. The transformers use the algorithms presented in Section 4.2 to implement the "transform" method, with the additional step of first asking the item to generate an initial encoding by using the "encoding_for(Organization)" of Item. The decode method is similarly straightforward, a Transformer simply transverses its component Organization in a top-down manner until it reaches a leaf node.

The class TransactionTable is a class which represents the transaction table found in relation systems. There may be multiple instances of this class, which correspond to different sets of transactions that may be specified in the FROM clause of a data mining query (see Section 4.3.1). The class has a property called "transactions" which is used to record this information.

Besides storing data, the TransactionTable serves as the object which coordinates the data mining process. The private method "multi_mine" is an implementation of the multi-level mining algorithm sketched in Figure 4, and it will not be further discussed in this section. The algorithm for the "mine" method is:

Input: the support threshold array supp[], and a list of n Transformers $t_1, \ldots, t_n$
Output: a set of large itemsets involving items of the types specified by the Transformers
Procedure:

- $ET \leftarrow$ an empty encoded transaction table
- for each transaction t
  - $t' \leftarrow$ an empty encoded transaction
  - for each item e in t
    - for each transformer t_j
      - $e' \leftarrow t_j$.transform(e)
      - if (e' is not NULL) add e' to t'
  - if (t' is not empty) add t' to ET
- $results \leftarrow$ multi_mine(supp,ET)
- return $results$

end

The algorithm simply walks through the transaction table, encoding every item based on the transformers that were given to it. The encoded table that is produced is then passed to base routine, and the results are returned to the user. Even if the encoding step is somewhat slower in the object-oriented version, we can still expect similar results in terms of performance. This is because the dominating step in mining association rules is usually the search for large 2-itemsets [2, 16] and that step is unaffected by the object-oriented implementation.

4.3.6 Example Walkthrough

Recall the query discussed in Section 2: The executive of an store wish to find a specific milk product to discount, which would mostly boost the sales of all types of bread. Using the SQL-like data mining language outlined in Section 4.3.1, this rule discovery query may be expressed as follows:

```
ASSOCIATE       Bread, Milk(4)
FROM            sales_transactions t, sales_items s, concept_hierarchies h
WHERE           t.date='today' AND s.store-location='Edmonton'
AND             h.nodename='food'
HAVING
  SUPPORT       0.5, 0.4, 0.35, 0.3, 0.2
  CONFIDENCE    default
  INTERESTSCOPE category, content-spec, brand, manufacturer
ORDER BY        Milk.manufacturer
```

In this query example, the ASSOCIATE clause says to include bread, in its most general form, and specific milk products. The FROM and WHERE clauses select the transactions and the organizational hierarchies which will be considered. The SUPPORT and CONFIDENCE clauses define the support and confidence thresholds (where "default" means to leave the decision up to the system). The semantics of INTERESTSCOPE and ORDER BY clauses are fairly intuitive.

Assume, before executing this query, the system has already performed some important work and the concept hierarchies have been constructed based on sales-item data and sales-transaction data stored in one or more databases of sales transactions. Each sales-item is uniquely identified by its barcode and has a list of attributes like category, brand, manufacturer, content-spec, price, store-location, and so forth. Each transaction contains a transaction ID and a list of bar-codes. The object-oriented mining system maps the bar-code of relevant sales-item and transaction ID to object identifiers for transactions and item objects. We also assume that the domain knowledge such as "milk is a food", 'bread is a food' is available from the ontology library for the sales business.

This query is first transformed into a standard SQL query to the source sales-transaction database(s) to retrieve all the relevant sales-items with their category belonging to "Food" and their sales-transactions occurring on the same day when this query is issued. For this example, assume that the concept hierarchy of Figure 1 is used, where bread products start with "12" and milk products start with "11".

The first step in the mining task is to convert the object oriented representation of the transaction table to an encoded form. This creates the *base encoded table*, and suppose the result of this step for our example query is given as shown in Figure 6. This encoded table contains sets of strings, which can be more easily manipulated than the original transaction identifiers or sales-item identifiers (e.g., barcodes). This step is performed by taking each transaction in the original table, and then iterating over each object within the transaction and sending it the *encoding_for* message with each concept hierarchy as an argument. Thus each item is converted into k encodings, where k is the number of hierarchies present in the queries; in our example $k = 1$ (only Food is selected).

TranactionOID_1		{ 1112, 1315, 1234, 1113 }
TranactionOID_2		{ 1221, 4314, 1118 }
TranactionOID_3	$\Rightarrow$	{ 1413, 2718, 1814 }
TranactionOID_4		{ 1222, 1111, 1413, 1567 }
TranactionOID_5		{ 1315, 1112, 1111, 1226 }

Figure 6: Base encoding step

If we only wanted to discover association rules which were at the primitive single concept level [2, 6], we could simply use the resultant table in Figure 6 and apply the mining algorithm described in Section 4.2. However, the example query requires that low-level milk products should be associated with the generalized item "Bread". We need to perform another transformation based on our adaptive encoding algorithm. For this step we create Transformer objects, which will massage the encodings so that Bread and low-level milk items "line-up". As well, this step will filter out those items which are not descendants of Bread or Milk. Figure 7 shows this step on our example query and the result table. The filtering process keeps all types of mike items (encoded with '11' as the prefix) and modifies the encoding of the general form of bread (encoded with '12' as the prefix initially) by adding the dummy levels in between '1' for `Food` and '2' for `Bread` as shown in the right side table of Figure 7. Note that empty transactions are not removed in this step, when we use the base mining algorithm by Han and Fu [6] (a variant is described in Section 4.2.), since it needs to have all transactions present to properly count the support of an itemset. It is not difficult to optimize the base algorithm so that it is no longer necessary to keep the empty transactions. This issue is beyond the focus of this paper.

{ 1112, 1315, 1234, 1113 }		{ 1112, 1xx2, 1113 }
{ 1221, 4314, 1118 }		{ 1xx2, 1118 }
{ 1413, 2718, 1814 }	⇒	{ }
{ 1222, 1111, 1413, 1567 }		{ 1xx2, 1111 }
{ 1315, 1112, 1111, 1226 }		{ 1112, 1111, 1xx2 }

Figure 7: Transforming and filtering using level matching

The next step is the discovery of association rules based upon the modified transaction table. Algorithms for solving this step have been studied extensively in the literature [1, 2, 16, 6, 18], and the modular design of object-oriented systems allows any algorithm to be easily used for this step. Finally, once rules have been discovered over the modified transaction table, these have to be decode back into object identifiers so that the results can be displayed to the user. The Transformer objects which we used to create the modified table all have a *decode* method which allows us to recover the object identifiers and thus transaction ids and the barcodes of sales-items, which in turn allows the mining system to assembly the results that will be returned to the end-user. Figure 8 shows the result of this step for our example query. The discovered association rules present the knowledge that a decrease in the price of Dairyland skim milk is most likely to lead to an increase in the bread sales.

{ 1112, 1xx2, 1113 }				
{ 1xx2, 1118 }		1111 → 1xx2 (0.7,0.4)		*Dairyland skim milk → Bread (0.7, 0.4)*
{ }	⇒		⇒	
{ 1xx2, 1111 }		1112 → 1xx2 (0.65,0.3)		*Lucern skim milk → Bread (0.65, 0.3)*
{ 1112, 1111, 1xx2 }				

Figure 8: Discovering and decoding the rules

5 Advantages of the object-oriented design

5.1 Eliminating Anomalies of Using Relational Mappings

There are many advantages to our framework for representing concept hierarchies in non-relational dynamic hierarchies, and its object-oriented implementation. Perhaps most significant is that it encourages the uniform treatment of class hierarchies and relational data through a uniform user query language, which mimics the style of familiar SQL-style syntax while providing user access to the increased functionality. The user knows that any item appearing in a discovered association rule will either be a class, or an instance of a class. For example, in the rule nuts $\rightarrow$ PartyTime unsalted pretzels, "nuts" would correspond to a class in the class hierarchy of the database, while "PartyTime unsalted pretzels" would correspond to all the Pretzel instances where salt_content = 'none' and manufacturer = 'PartyTime'.[2] Any clause included in a rule is also directly queriable.

A second advantage, is that items can be viewed by multiple representations through the class hierarchy. Instead of collecting items of different representations into a single relational table, items can be organized and represented in terms of classes of different number of properties. For example, Milk might have a property named "butterfat_content", but Bread would probably not. Instead of forcing Milk items and Bread into a global sale_items relation, we may simply organize Milk and Bread as two different subclasses in the Food class hierarchy. By design *and* implementation, the only way to create objects is through their classes. Thus every item would have to be assigned to a node in the classification hierarchy when it was created. Moreover, the class hierarchy provides a clear concept taxonomy that helps the creator of new objects to decide to which classification the new items belong.

5.2 Dynamic Adjustment of Concept Hierarchies

Most of mining algorithms represent predefined concept hierarchies in the form of flat relational tables. Unfortunately some portions of the user-defined and domain-specific concept hierarchies, such as milk, ..., bread $\subset$ food, or peanut, pistachios, ..., walnuts $\subset$ nuts, etc., may not be explicitly and uniformly presented in the form of relational tables.

Using our framework, the higher levels of concept hierarchies are captured explicitly and uniformly in the form of classes or objects in classes, and the lower levels of concept hierarchies can be adjusted dynamically upon request. This is because, quite often, the upper levels of the concept hierarchy, such as food, are fixed in their order since other orderings like beverages, fruits, grains $\subset$ milk $\subset$ food make little sense. However lower levels of the hierarchy may be re-ordered. For example both Dairyland skim milk $\subset$ Dairyland milk $\subset$ skim milk $\subset$ milk and Dairyland skim milk $\subset$ skim milk $\subset$ Dairyland milk $\subset$ milk are valid orderings. The hierarchical representation is designed to create the appropriate encoded representation at run-time, so it is straight forward to employ dynamic rule mining schemes. Those schemes allows the user to specify interesting properties, in a preferred order of attributes, and the encoding scheme then uses that information to determine the relevant encoding.

Another interesting advantage of our design is the capability to allow hierarchies to be restricted to ranges of transactions; as previously mentioned, this can be accomplished with a simple attribute called "range" in class Organization. In a typical retail situation, the transactions are ordered by their occurence, and thus a range specification can also be seen as a temporal specification. This

[2]In an object-oriented framework, classes and instances of a class are all treated as first-class objects. Through an object-oriented query interface, such as those provided in OODBMS systems, users may query over classes and instance objects of the classes in the same manner [4, 15, 3].

is significant in that it permits temporal queries, touted by some as a key ability for knowledge discovery systems [11]. Relational systems can also exploit the temporal ordering of transactions, but this adds another piece of information that needs to be stored and maintained for each hierarchy. Again our framework can be more easily extended to handle this additional complexity.

We argue that in order to find more informative and interesting rules from large transaction databases, it is important for the data mining system to support dynamic adjustment of concept hierarchies in order to focus the mining on those rules that are useful with respect to particular business objectives, such as temporal ranges of sale transactions or price distribution of certain types of products.

5.3 Supporting the Adaptive Encoding Scheme

There are many advantages with an adaptive approach to encoding items. One key benefit is that it enables the encoding algorithm to be closely associated with the item being encoded, in contrast to the relational case where the encoding algorithm is one of the decisions that the mining software makes. As a consequence, it allows other programs to make use of the encoding information being available. Further, the kind of dependencies that the encoding scheme follows can be handled in a structured way — if the mining application wishes to change the encoding format, then that change could be handled as a schema update in object-oriented database systems [12].

Most importantly, our adaptive encoding scheme accepts objects that may have different numbers of properties, and rules that associate attributes at different levels of a concept hierarchy. For example, Milk objects might have 5 properties and Bread objects might only have three. This can be handled in the object-oriented solution by simply inserting dummy values into the bread objects' encoded representation so that both objects have the same number of properties in the encoded table. This technique is very similar to the one used to support rule spanning levels (see Section 4.2) and is superior to the techniques presented in [6] for finding rules which span concept levels, because it does not require any change to the underlying rule discovery algorithm. However, it does require a flexible encoding algorithm, something that is easy to provide in our framework framework.

Finally, the adaptive encoding mechanism provides adequate support for mining association rules across multiple concept hierarchies. As mentioned previously, to support a different hierarchy all we need to do is provide routines to encode and decode the items based on that concept organization. This is not difficult to do. It simply amounts to enumerating, for each set which forms the alternate hierarchy, the objects which are contained in that set, and pasting these numbers together to get an encoded string. Moreover, it is also easy to support multiple hierarchies in one rule by prefixing the encoded string with a concept hierarchy identification number. For example to find rules of the type Dairyland Skim Milk $\wedge$ Coupon Item $\rightarrow$ Dairyland Cheese, one would need to use the type hierarchy and the promotions hierarchy. This could be done by prefixing unique identifiers to the front of the encoded strings, which is functionally equivalent to creating a new top level which corresponds to the type of the hierarchy. Items in each transaction would then be encoded k times, where k is the number of hierarchies desired in the final rule, and the union of these encodings would become the transaction in the encoded table. This technique could also be used in relational systems, but is more naturally supported in an object-oriented framework and is thus more likely to be implemented.

6 Related Work

The problem of mining association rules in transaction databases was introduced by Agrawal, Imielinski, and Swami [1]. Subsequent research has resulted in fast algorithms for the generation of these types of rules [2, 16, 18]. This research confines itself to finding association rules in a relational database system, at what is known as a single concept level. Mining at a single concept level means that the items in a rule are always concrete; thus we can find rules like *85% of people who buy* Dairyland skim milk *may also buy* Wonder Whole Wheat Bread, but not conceptually higher rules like *90% of people who buy* milk *may also buy* bread. One primary concern with mining at single concept level is that the number of association rules generated are either so large as to overwhelm the user, or is too small to contain enough "interesting" rules [10] if the thresholds of support and confidence are tightened. It seems that using the two thresholds of support and confidence is not sufficient in many cases to filter out the "weak" or "uninteresting" rules.

Several researchers proposed using domain-specific classification taxonomies to focus the search of interesting association rules from the large set of discovered ones. Klemettien et al [10] propose to reduce the number of generated rules by having the user specify a template, which details the form of the rule (i.e., the number of items in the antecedent and the consequent) and restricts the items that can be used in the rule. This is accomplished by organizing the items into a classification hierarchy (within a relational database), and using this information to guide the rule selection process. However, this technique only deals with mining rules at one concept level — templates are merely used to select a subset of the rules from those that would have been returned by the previous algorithms.

Another approach, proposed by Han and Fu [6], deals with the problem of focusing the search for interesting association rules by mining multi-level association rules. They also use a concept classification hierarchy to capture taxonomy knowledge of the relational data. The novelty of their approach is that they do not use the classification hierarchy simply for filtering of low-level rules, but instead are able to find associations among higher level concepts. Thus their approach allows one not only to find rules generated using the basic mining algorithms, but also to discover rules which associate more generalized items. However, their approach suffers from the limitation of presenting concept hierarchies in the form of relational tables and of the fixed encoding scheme.

Holsheimer et al. [9] report on integrating the data mining process more closely into the database paradigm. They present an algorithm which finds association rules at a single concept level using only union and intersection operations, and show how it can be extended to mine at multiple concept levels. They succeed in incorporating the data structures and operations used in database mining into the relational paradigm, at the loss of some efficiency. To mine at multiple concept levels, the classification hierarchy is utilized although this type of information is not fully integrated into the relational database system.

Shen et. al. [19] introduce the notion of metaqueries for guiding the data mining process. Similar to templates, metaqueries work by specifying an abstract form that the rule must satisfy. However, metaqueries in their most general form are most useful for mining data from different relations in a database, which is not the case for transaction data. Fu and Han [5] show how metaqueries can be used in mining association rules at multiple concept levels, however the type of data they examine is quite different from those used in [19]. Moreover, both papers consider metaqueries solely, and do not specify a complete architecture for data mining.

Knowledge discovery in object oriented databases (OODBs) is examined by Han, Nisho, and Kwanono in [7]. Their focus is mainly on the ways of discovering knowledge *contained* in an arbitrary OODB, rather than using an object-oriented approach to organize the concept hierarchies for focusing the mining of interesting and strong association rules from large relational databases.

7 Conclusion and Future Work

We have presented a framework for the mining of association rules in relational database, based on an explicit representation of dynamic hierarchies which exploit an object-oriented implmentation. The main contributions of this paper are the following. First, we propose an adaptive encoding scheme to support dynamic adjustment of concept hierarchies in order to find more informative and more interesting association rules. Our encoding scheme can be seen as an adaptive extension to the fixed encoding scheme used in [6]. Second, we demonstrate that the application of an object-oriented approach is better suited for focusing the search and regulating the mining of multi-level association rules that span over many levels of a concept hierarchy or multiple concept hierarchies. By using the adaptive encoding scheme, we have shown that the problem of mining association rules of form $X \rightarrow Y$ where X, Y are at any (possibly different) levels of a concept hierarchy can be reduced to the problem of discovering all rules $X \rightarrow Y$ where X, Y are at the same level of the concept hierarchy. Thus, efficient algorithms developed for discovering strong and interesting association rules [1, 2, 6] can be seamlessly integrated into the object-oriented framework with no or little extra cost. Note that we have advanced the notion of directly reusing existing algorithms for expositional clarity. In an actual implementation, optimizations could be made to the base algorithms — for example if we "push" an item down from level 2 to 4, there is no need to calculate its support at level 3 or 4 because it will not have changed from level 2.

We believe that the application of our design (and its object-oriented implementation) to data mining provides us not only the benefits of a flexible combination of multiple multi-level concept hierarchies for focusing the mining on more informative and refined knowledge, but also allows us to enjoy the seamless integration of multi-level concept hierarchies with the legacy relational databases. All these benefits come with very little loss in performance since the same algorithms used with relational data can be trivially adapted to the object-oriented implementation through the use of dynamically encoded transaction table(s).

Our work on multi-level association rule mining continues. On the practical side, we are interested in building a prototype for experimenting with various organizations of the concept hierarchies, and with various methods for focusing the mining on more informative and more interesting association rules. We are also interested in studying the performance of combining our adaptive encoding scheme with existing algorithms for multi-level association rule mining at multi-levels of a single concept hierarchy and across multiple concept hierarchies. On the theoretical side, we are interested in extending the approach to handle databases which are more general than transaction databases, and designing faster algorithms for mining more interesting and informative association rules from large legacy databases. Another important issue would be finding appropriate mechanisms to eliminate redundant or nearly redundant rules. As pointed out in [6, 10], large amount of rules could potentially be pruned if there were adequate ways to remove redundancy. We are also interested in applying data mining techniques to resource discovery in global and interoperable information systems [14, 13].

References

[1] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In *Proceedings of the 1993 ACM-SIGMOD International Conference on the Management of Data*, pages 207–216, Washington, D.C., May 1993.

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

[3] C.Lamb, G.Landis, J.Orenstein, and D.Weinreb. The object-store database system. *Communications of the ACM*, 34(10):51–63, October 1991.

[4] E. Bertino et. al. Object-oriented query languages: the notion and the issues. *IEEE Transactions on Knowledge and Data Engineering*, 4(3):223–237, 1992.

[5] Y. Fu and J. Han. Meta-rule-guided mining of association rules in relational databases. In *Proceedings of the 1995 International Workshop on Knowledge Discovery and Deductive and Object-Oriented Databases*, Singapore, December 1995.

[6] J. Han and Y. Fu. Discovery of multiple-level association rules from large databases. In *Proceedings of the 21st VLDB Conference*, Zurich, Switzerland, 1995.

[7] J. Han, S. Nishio, and H. Kawan. Knowledge discovery in object-oriented and active databases. In *Proceedings of the 1993 International Conference on Building and Sharing of Very Large Scale Knowledge Bases*, pages 205–214, Toyko, Japan, December 1993.

[8] Jiawei Han. Mining knowledge at multiple concept levels. In *ACM International Conference on Information and Knowledge Management (CIKM'95)*, Baltimore, Maryland, USA, November 1995.

[9] M. Holsheimer, M. Kersten, H. Mannila, and H. Toivonen. A perspective on databases and data mining. Technical Report CS-R9531, University of Helsinki, 1995.

[10] M. Klemettinen, H. Mannila, P. Ronkainen, H. Toivonen, and A.I. Verkamo. Finding interesting rules from large sets of discovered rules. In *Proceedings of the 3rd International Conference on Information and Knowledge Management*, pages 401–408, Gaithersburg, Maryland, Nov 1994.

[11] R. Krishnamurthy and T. Imielinki. Practioner problems in need of database research: Research directions in knowledge discovery. *ACM SIGMOD Record*, 20:76–78, September 1991.

[12] L. Liu. Database consistency in the presence of schema evolution. In *Proceedings of IFIP Data Semantics conference*, Atlanta, 1995.

[13] L. Liu, C. Pu, and Y. Lee. An adaptive approach to query mediation across heterogeneous databases. In *Proceedings of the International Conference on Coopertive Information Systems*, Brussels, June 19-21 1996.

[14] Ling Liu and Calton Pu. The distributed interoperable object model and its application to large-scale interoperable database systems. In *ACM International Conference on Information and Knowledge Management (CIKM'95)*, Baltimore, Maryland, USA, November 1995.

[15] D. Maier and J. Stein. Development and implementation of an object-oriented DBMS. In S.Zdonik and D.Maier, editors, *Readings in OODB Systems*. 1990.

[16] J.S. Park, M.S. Chen, and P.S. Yu. An effective hash based algorithm for mining association rules. In *Proceedings of the 1995 ACM-SIGMOD International Conference on the Management of Data*, San Jose, CA, May 1995.

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

[18] A. Savasere, E. Omiecinski, and S. Navathe. An efficient algorithm for mining association rules in large databases. In *the 22nd International Conference on Very Large Databases*, Zurich, Swizerland, November 1995.

[19] W. Shen, K. Ong, B. Mitbander, and C. Zaniolo. Metaqueries for data mining. In U.M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*. AAAI/MIT Press, 1995.