2. Literature review

Production rule is an important element in the expert system. By interview with the domain experts, we can induce the rules and store them in a truth maintenance system. An assumption-based TMS has carried out by de Kleer in [16][17][18][19]. After that, the discovery of association rules has been extensively studied with the conception of ATMS in 90's such as [11][22][23][27]. Many extensions were also proposed in recent years, such as: multi-level and generalized rules in [13][24], inter-transaction association rules in [8], Periodic Patterns in [9][10], temporal association rules in [30][33], Mutually Dependent Patterns in [28], and fuzzy rules in [7]. The study in [32] takes that some products may not be on shelf in some stores during some period into considered. Mining association rules with multiple minimum supports was discussed in [3], Applications of association rules were presented in [2][5][29], and the privacy preserving issue has also studied in [1] .

In the section, we will introduce the formal definition and mining process of association rules first, and then we will review some approaches of multi-dimension rules, and a method of mining calendar-based temporal association rules in [33]. Concept hierarchy, which is an important element of our method, will also be discussed at last in this section.

2.1. Association rule

Association rule is one of the major forms of data mining and is perhaps the most common form of knowledge discovery in unsupervised learning systems. The notion of association rule was proposed to capture the co-occurrence of items in transactions. A typical application of association rule is market basket analysis; this process analyzes customer buying habits by finding association between the different items

that customers place in their “shopping baskets” [14] .

2.1.1. Definition of association rules

Suppose we have a transaction database **D** (as Fig. 2.1). A transaction is a tuple in **D** which is consist of a set of items and is identified by its T_ID.

T_ID	Transaction content
001	Bread, Milk
002	Candy, Juice
003	Beer, Diaper
004	Juice, Tomato, Orange

Figure 2.1 transaction database D

Let **I** = $\{i_1, i_2, \ldots, i_n\}$, is the set of all *n* different items in **D**. Each transaction in **D** is a subset of **I**. An itemset is also defined as a subset of **I**. An association rule usually has this form : **A**→**B**, where **A** and **B** are two disjoint itemsets. The association rule **A**→**B** implies that among transactions in **D,** the occurrences of **B** have high correlation with the occurrences of **A.** There are two important factors with association rules. One is *support*, the other is *conference*. *Support* means how often the rule applies. The *support* of an itemset **X** is the fraction of transactions containing **X** in **D**. Given a threshold σ as *minimum support*, **X** is a *large itemset* in **D** if the *support* of **X** in **D** exceed σ. *Confidence* means how often the rule is correct. The *confidence* of **A**→**B** is the fraction of transactions containing **A** and **B** simultaneously in transactions containing **A**. The equations are shown below:

$$Support(X) = \frac{|\text{ Transactions in } \mathbf{D} \text{ containing } X \text{ }|}{|\text{ Transactions in } \mathbf{D} \text{ }|}$$

$$Confidence(A \rightarrow B) = \frac{|\text{ Transactions in } \mathbf{D} \text{ containing both } A \text{ and } B \text{ }|}{|\text{ Transactions in } \mathbf{D} \text{ containing } A \text{ }|}$$

2.1.2. Mining association rules

Mining association rules can be decomposed into two sub-problems: (1) finding all large itemsets, and (2) generating association rules using this large itemsets. The search task in step1 is the crux of discovering association rule, and most algorithms of mining association rules focus on the front one. The simplest search strategy is exponential: Enumerate all the possibilities and try each one until a solution is found [16]. That is, there are n binary selections giving 2^n itemsets which is exponentially with the number of items into consider. For example, if there are 5 items, there will be $2^5 = 32$ candidates as in fig.2.2.

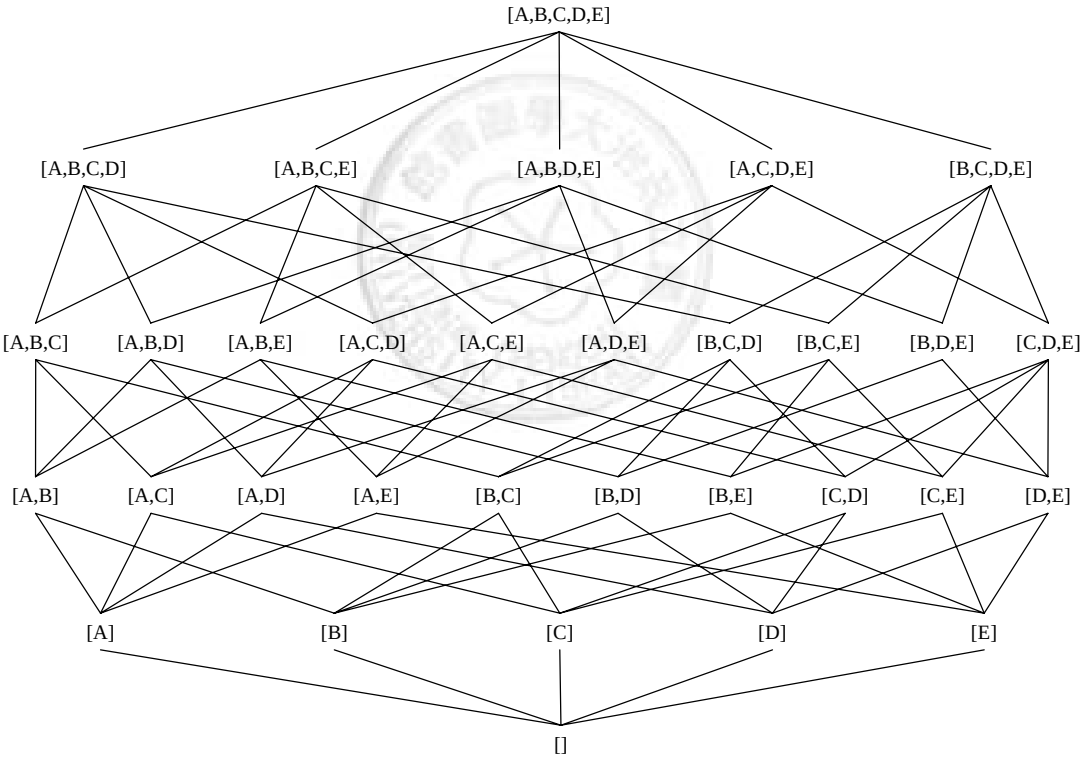

Figure 2.2 Initial candidate space for the circuit example

A concept carried out in [16] can help us with pruning the search space quickly. That is: “If an environment is nogood, then all of its superset environments are nogood as well.”[16]. The definition of large itemsets also indicates a similar property: **X** is a large itemset in **D**, if and only if all subsets of **X** are also large itemsets in **D**.

Thus when an itemset **X** is verified to be a large itemset, all the subsets of **X** can be verified as large itemsets also. On the other hand, if an itemset **X** is verified to be not large, all the superset of **X** can be verified to be not large at the same time. We can reduce the candidate space quickly with this property.

2.1.3. Entropy function & application

In order to reduce the set of remaining candidates efficiently, we help to measure the itemsets that we can gain the most of information after measuring them. By cascade evaluating the consequences of a hypothetical measurement, we could evaluate the consequences of any sequence of measurements to determine the optimal next measurement. The method proposed in [19] uses a one-step lookahead strategy based on Shannon entropy. From decision and information theory we know that a very good cost function is the entropy (H) of the candidate probabilities:

$$H = -\sum p_i \log p_i$$

Where p_i is the probability that candidate $\mathbf{X_i}$ is the actual candidate given the hypothesized measurement outcome, and the cost of locating a candidate of probability p_i is proportional to $\log p_i^{-1}$. The best measurement is the one which minimum the expected entropy of candidate probabilities resulting from the measurement. The expected entropy $H_e(X_i)$ after measuring quantity is given by:

$$H_e(X_i) = \sum (k = 1 \text{ to } m)\, p(x_i = v_{ik})\, H(x_i = v_{ik}).$$

Where $v_{i1}, \ldots, v_{im}$ are all possible values for x_i, and $H(x_i = v_{ik})$ is the entropy resulting if x_i is measured to be v_{ik}.

With Shannon entropy, we can deduce that the itemset which contains smaller items we count, the more information we get (a prove is given in Appendix A).

According to above results, we begin the counting from 1-itemsets and develop an efficiency level-wise searching algorithm with downward closure property to discover all large itemsets.

2.1.4. Mining association algorithm

Given a set of transactions **D**, the problem of mining association rules is to generate all association rules that have support and confidence greater than the user-specified minimum support (called *minsup*) and minimum confidence (called *minconf*) respectively.[23]

According to the results deduced above, we can develop an efficiency level-wise searching algorithm with downward closure property to discover all large itemsets. The outline of this algorithm is shown in Fig. 2.3.

1) L_1 = {large 1-itemsets};
2) **for** (k = 2; $L_{k-1} \neq \phi$; k++) **do begin**
3) C_k = Apriori-gen(L_{k-1}); // New candidates
4) **forall** transactions t in ***D*** **do begin**
5) C_t = subset(C_k, t); // Candidate contained in t
6) **forall** candidates c in C_t **do**
7) c.count++;
8) **end**
9) L_k = {c in C_k | c.count$\geqq$minsup}
10) **end**
11) Answer = $\cup_k L_k$;

Figure 2.3 Association rules mining algorithm

In Fig. 2.3, the first pass of the algorithm simply counts item occurrences to discover the frequent 1-itemsets. A subsequent pass, pass *k*, consists of two phases. First, use the frequent itemset L_{k-1} found in the $(k-1)^{th}$ pass to generate the candidate itemsets C_k by using the candidate generation procedure(described below). Next, scan the database to count the support of candidate in C_k.

The intuition behind the candidate generation procedure is that if an itemset ***X***

has minimum support, so do all subsets of ***X***. Given L_{k-1}, the set of all frequent (k-1)-itemsets, assume the items in each itemset are in lexicographic order for simplicity, candidate generation takes two steps. First, in the join step, join L_{k-1} with L_{k-1} as below:

insert into C_k
select $p.\text{item}_1, p.\text{item}_2, \ldots, p.\text{item}_{k-1}, q.\text{item}_{k-1}$
from $L_{k-1}\ p, L_{k-1}\ q$
where $p.\text{item}_1 = q.\text{item}_1, \ldots, p.\text{item}_{k-2} = q.\text{item}_{k-2}, p.\text{item}_{k-1} < q.\text{item}_{k-1}$;

Next, in the prune step, delete all itemsets $c \in C_k$ such that some (k-1)-subset of c is not in L_{k-1}. For example, let L_2 be {{ab}, {ac}, {ad}, {bc}, {cd}}. After the join step, C_3 will be {{abc}, {abd}. {acd}}. The prune step will delete the itemset {abd} because the itemset {bd}is not in L_2. We will then be left with {abc} and {acd} in C_3. For fast counting, we need to efficiently determine the candidates in C_k contained in a given transaction t. A hash-tree data structure is used for this purpose.

Many variations of this algorithm have been proposed that focus on improving the efficiency of the original algorithm, such as: “Hash-based technique (hashing itemset counts)”, “Transaction reduction (reducing the number of transactions scanned in future iterations)”, “Partitioning (portioning the data to find candidate itemsets)”, “Sampling (mining on a subset of the given data)” and “Dynamic itemset counting (adding candidate itemsets at different points during a scan).

2.2. Multi-dimension association rule

Association rules that involve two or more dimensions or predicates can be referred to as multi-dimensional association rules [14]. Informally, we refer to the association rules along with other attributes as multi-dimension association rules in this paper. We will introduce some approach: “Quantitative Association Rules”, “Relationship graph” and “ARCS, association rule clustering system” in this section.

2.2.1. Quantitative association rules

The problem of traditional association rule mining can be viewed as finding associations between the “1” values in a relational table where all the attributes are Boolean [25]. However, an attribute in a relational table can be quantitative or categorical. If we want to discover association rules in a relational table, how to handle the quantitative attributes will be a critical problem. We will introduce the method proposed in [25], which apply the concept proposed in [20].

Categorical attributes also called nominal attributes, which have a finite number of possible values and with no ordering among the values, such as occupation, brand and color. Quantitative attributes are numeric and have an implicit ordering among values, such as age, income and price. An example of this table is shown in fig. 2.4. Fig. 2.4 shows a customer table with a primary key customer ID (C_ID). The attribute “Age” and “NumCars” are quantitative attributes and the “Married” is a categorical one. The value of a Boolean field corresponding to <attribute$_1$, value$_1$> would be “1” if attribute$_1$ had value$_1$ in the original record, and “0” otherwise. If the domain of values for a quantitative approach is large, an obvious approach will be to first partition the values into intervals and then map each <attribute, interval> pair to a Boolean attribute.

fig. 2.5 shows this mapping for the non-key attributes of the customer table given in Fig.2.4. The quantitative attribute “age” is partitioned into two intervals: 30~39 and 40~49, while another quantitative “NumCars” with small number of values is not partitioned into intervals. The categorical attribute “Married” has two Boolean attributes “Married = Yes” and “Married = No”. We can now use any algorithm for finding traditional association rules to find quantitative association rules, and the part of result is shown in Fig.2.6.

C_ID	Age	Married	NumCars
001	34	No	1
002	38	Yes	1
003	33	No	0
004	47	Yes	2
005	42	Yes	2

Figure 2.4 Example of a customer table

C_ID	Age=30~39	Age=40~49	Married=Yes	Married=No	NumCars=0	NumCars=1	NumCars=2
001	1	0	0	1	0	1	0
002	1	0	1	0	0	1	0
003	1	0	0	1	1	0	0
004	0	1	1	0	0	0	1
005	0	1	1	0	0	0	1

Figure 2.5 Mapping to Boolean association rules problem

(Minimum support = 40%, minimum confidence = 66.6%)

Rules (Sample)	Support	Confidence
<Age = 40~49> and <Married = Yes> → <NumCars = 2>	40%	100%
<NumCars = 0~1> → <Married = no>	40%	66.6%

Figure 2.6 Part of quantitative association mining result

2.2.2. Relationship graph

Previous researches on mining association rules only focus on discovering the relationships among items in the transaction database. The relationships between items in the transaction database and attribute values in the customer database have not been explored in the literature yet [21]. The method proposing in [21] discovers all large itemsets first, and then assigns them attributes according to the priority of each attribute. An example of such an association rule might be "80% of customers whose degree are PHD buy itemset **X**". Example for the transaction database and the customer database are shown in Fig. 2.7.

CID	itemset
1	005, 001, 015
2	001, 003, 013
3	003, 015
4	003, 006, 027
5	011, 203, 002
.	.
.	.
.	.

CID	name	gender	degree	specialty
1	John	M	BS	data-mining
2	Mary	F	PhD	distributed-database
3	Joe	M	BS	spatial-database
4	Jane	F	MS	data-mining
5	Jill	F	MS	spatial-database
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.

Figure 2.7 Example of transaction and customer database

Let **CDB**(*CID*, CA_1, CA_2, …, CA_n) be the customer database and **TDB**(*CID*, *itemset*) the transaction database where *CID* represents the identification number of a customer. The rules discovered in [16] consist of the antecedent and the consequent. The antecedent is a conjunction of "$CA_i = v_i$," where CA_i is a condition attribute from the customer database and v_i the associated value, the consequent is a large itemset. Let $pr(CA_i)$ be the priority associated with the condition attribute CA_i, $1 \le i \le n$. Fig. 2.8 shows the relationship graph that is constructed according to the priorities of attributes and large itemsets.

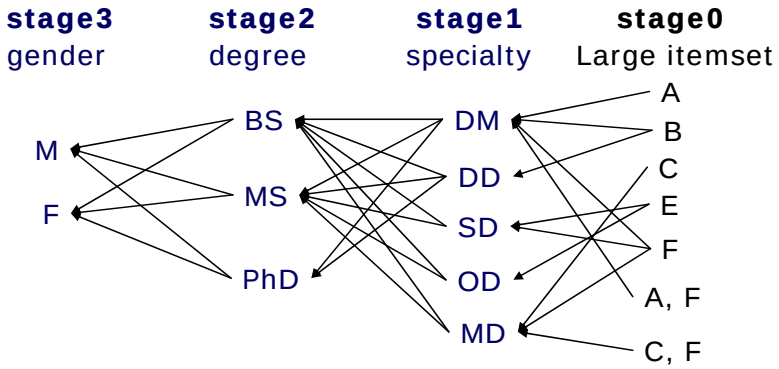

Figure 2.8 Example of relationship graph

Let *stage*(CA_i) be the stage number for attribute CA_i in the relationship graph, where *stage*(large itemset) is 0, and *stage*(CA_i)<*stage*(CA_j) if *pr*(CA_i)<*pr*(CA_j). There is an edge from v_i to v_k, if *stage*(CA_j) = *stage*(CA_k)-1 and there is a tuple with v_j for attribute CA_j and v_k for attribute CA_k. An edge is connected from LS_i to v_{1j} if there is a customer who has v_{1j} for attribute CA_1 and bought all the items in LS_i. The algorithm in [16] is shown in Fig. 2.9. This algorithm first considering the relationship between a large itemset LS_i and a value, v_1, of CA_1, if there is an edge from LS_i to v_1 in the relationship graph. The algorithm will add rule "IF $CA_1 = v_1$ THEN itemset = LS_i" to the rule set, if the confidence = $|L(LS_i,v_1)| / |T(v_1)|$ satisfies the minimum confidence. This algorithm continues to consider the relationship between LS_i and the values v_1 and v_2 for attributes CA_1 and CA_2 if needed.

BEGIN
$S = \phi$
for i = 1 to m do
/* m is the number of large itemsets. */
for all edges <LS_i,v_1> **do**
/* LS_i is one of the large itemsets. */
/* v_1 is a value for CA_1. */
$T' = T(v_1) - L(LS_i,v_1)$;
/* T' represents the set of CIDs for customers who did not buy all the items in the large itemset LS_i but have the value v_1 for condition attribute CA_1. */
if $T' = \phi$ **then**
adding the rule "**IF** $CA_1 = v$ **THEN** *itemset* = LS_i [condidence = 1]" to the rule set S.
else
$LIST$= <v>; /* $LIST$ is an ordered list. */
$TS = L(LS_i,v_1)$;
j = 1; /* j represents the present considered stage number. */
$conf = |TS| / |T(v_1)|$;
if $conf \geq$ *minimum confidence* **then**
adding the rule "**IF** $CA_1 = v_1$ **THEN** *itemset* = LS_i [confidence = $conf$]" to the rule set S.
end if
Next Stage (TS, v_1, j, $LIST$, T', $T(v)$, LS_i);
end if
end for
end for
END

Figure 2.9 Algorithm of relationship graph

This method may lose some important rules hold in parts of database, since it discovers all large itemsets based on whole database first. The different priority of each condition attribute will induce different rule produced. That is, except the priority of each attribute is definite, user may need to try every possible priority to discover all possible rules.

2.2.3. Association rule clustering system (ARCS)

ARCS, association rule clustering system, was proposed in [4]. The method in [4] focus specifically on how to mine quantitative association rules having two quantitative attributes on the left-hand side (LHS) of the rule, and one categorical attributes on the right-hand side (RHS) of the rule. This method considers association

rule clustering in a two-dimensional space, where each axis represents one attribute from the database used on the RHS of a rule. For example, if we want to discover the relationship between salary, age of a person and if owning a house, we will partition the two quantitative “salary” and “age” into intervals to form a BinArray such as Fig. 2.10 first.

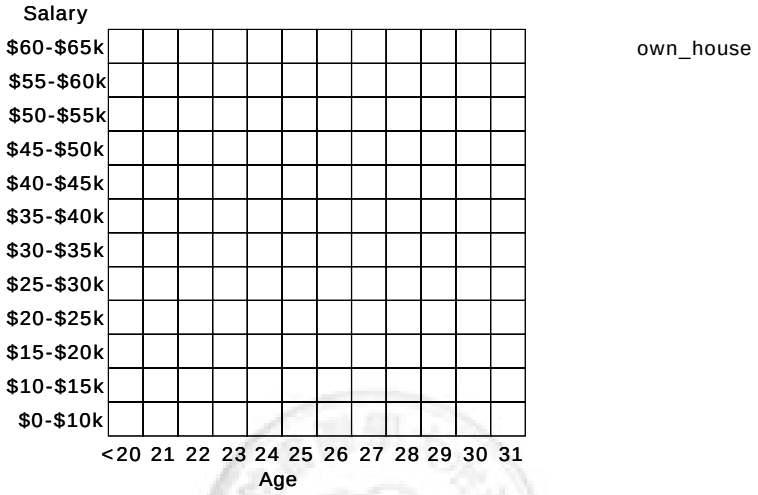

Figure 2.10 An example of BinArray

Every cell in Fig. 2.10 can be represented by an association rule: ($X = i$) and ($Y = j$) $\rightarrow$ G_k, where X is “age”, Y is “salary”, G_k is the Boolean attribute “own a house = yes” or “own a house = No”. The support and the confidence of this rule are defined as follow:

$$\text{Support: } \frac{|(i, j, G_k)|}{N} \quad \text{confidence: } \frac{|(i, j, G_k)|}{|(i, j)|}$$

Where N is the total number of tuples in the source data, $|(i, j)|$is the total number of tuples mapped into the BinArray at location (i, j), and $|(i, j, G_k)|$ is the number of tuples mapped into the BinArray at location (i, j) with criterion attribute value G_k. We check each of the occupied cells in the BinArray to see if the above conditions hold. If the thresholds are met, we give the cells a mark to locate clusters of association rules. The marked BinArray is shown in Fig. 2.11.

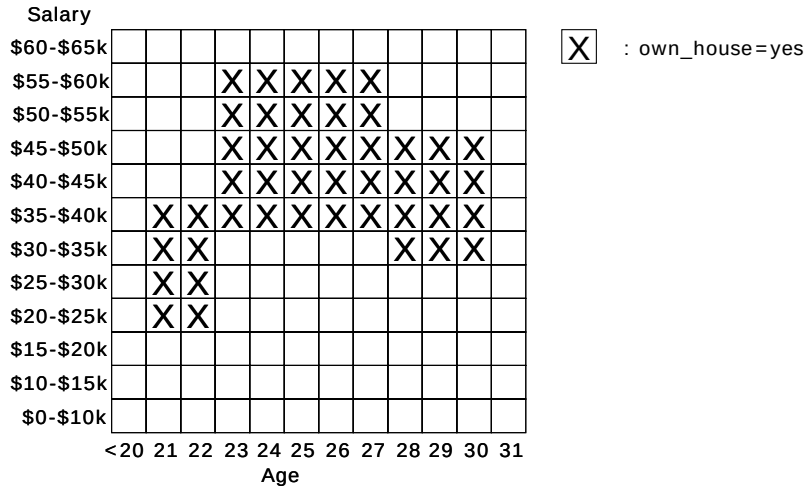

Figure 2.11 An example of marked BinArray

At last, we cluster the marked cells. The result of cluster and the output rules are shown in Fig. 2.12.

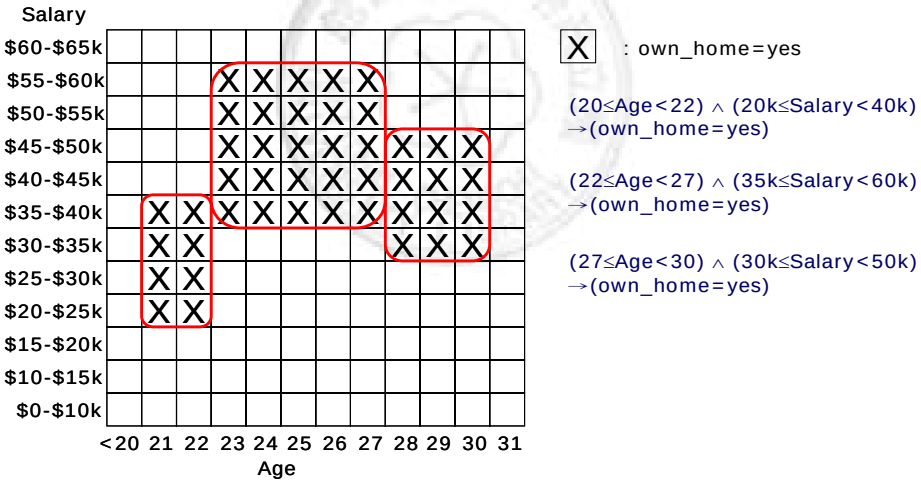

Figure 2.12 Result and output rules

The restriction of ARCS is that the categorical attributes may not be suitable to cluster, only quantitative attributes can be the left-hand side of the rule. On the other hand, only one kind of rules which have the same right-hand side can produce in once cluster, thus we need mass redundancy database scan or huge memory and a lot of cluster process to discover all the rules.

2.3. Calendar-based temporal association rules

The calendar-based temporal association rules was proposed in [33]. The works in [33] develop a calendar schema to produce calendar-based patterns in different time granularities. The method finds all large itemsets for each element pattern first, and then uses the output of every element pattern to update other calendar-based patterns which cover them.

2.3.1. Calendar schema and calendar pattern

A calendar schema is a relational schema $\boldsymbol{R} = (f_n{:}D_n, f_{n-1}{:}D_{n-1}, \ldots, f_1{:}D_1)$. Each attribute f_i is a time granularity name like year, month, and week etc. and domain D_i is a finite subset of the positive integers. For example, $\boldsymbol{R}$ = (year:{1995,1996},month:{1, 2, …, 12}, day:{1, 2, …, 31}).

A simple calendar-based pattern on the calendar schema $\boldsymbol{R}$ is a tuple on $\boldsymbol{R}$ of the form $<d_n, d_{n-1}, \ldots, d_1>$, where each d_i is in D_i or the wild-card symbol '*'. For example, with the above calendar schema $\boldsymbol{R}$, a calendar-based pattern <2005, 1, *> means every day of January of 2005, and <*, 8, 16> means every 16^{th} day of August of every year.

A calendar pattern with exactly k wild-card symbols is a k-star calendar pattern (denoted e_k), and a calendar pattern with at least one wild-card symbol is a star calendar pattern. In addition, a calendar pattern with no wild-card symbol is a basic time interval under the calendar schema. A simplified example of the calendar schema and calendar patterns is shown in Fig. 2.13.

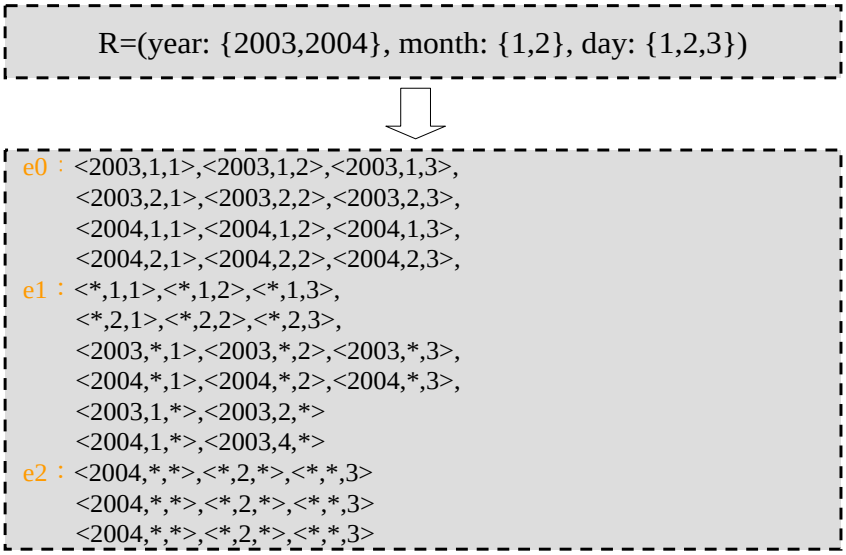

Figure 2.13 An example of calendar schema and calendar patterns

2.3.2. Calendar-based temporal association rules

Given a basic time interval t (or a calendar pattern e) under a given calendar schema, the set of transactions whose time-stamps are covered by t (or e) are denoted as $T[t]$ (or $T[e]$). A temporal association rule over a calendar schema $\boldsymbol{R}$ is a pair (r, e), where r is an association rule and e is a calendar pattern on $\boldsymbol{R}$. Given a calendar schema $\boldsymbol{R}$, a set T of time-stamped transactions, a temporal association rule (r, e) holds in T if and only if the association rule r holds in $T[t]$ for enough basic time interval t covered by e.

The algorithm proposed in [33] is shown in Fig. 2.14. This algorithm works in pass as in [23]. In each pass, the basic time intervals in the calendar schema are processed one by one. During the processing of basic time interval e_0 in pass k, the set of large k-itemsets $L_k(e_0)$ is first computed and then $L_k(e_0)$ is used to update the large k-itemsets for all the calendar patterns that cover e_0.

1) **forall** basic time intervals e_0 **do**
2) $L_1(e_0)$ = { large 1-itemsets in $T[e_0]$ }
3) **forall** star patterns e that cover e_0 **do**
4) update $L_1(e)$ using $L_1(e_0)$;
5) **end**
6) **for** (k = 2; ∃ a star calendar pattern $\boldsymbol{e}$ such that $\boldsymbol{L_{k-1}(e) \neq \phi}$; k ++) **do**
7) **forall** basic time intervals e_0 **do**
8) // **Phase I** : generate candidates
9) generate candidates $C_k(e_0)$;
10) // **Phase II** : scan the transactions
11) **forall** transactions $T \in T[e0]$ **do**
12) subset ($C_k(e_0)$, T);
13) // $c.count$ ++ if c $\in C_k(e_0)$ is contained in T
14) $L_k(e_0)$ = { $c \in C_k(e_0)$ | $c.count \geqq minsupport$};
15) // **Phase III** : update for star calendar patterns
16) **forall** star patterns e that cover e_0 **do**
17) update $L_k(e)$ using $L_k(e_0)$;
18) **end**
19) output <$L_k(e)$,e> for all star calendar pattern e.
20) **end**

Figure 2.14 Outline of algorithm in calendar-based temporal association rules.

The method in [33] proposing an appropriate strategy to discover association rules in every part of database enclosed by calendar-based patterns, but the method only addresses the time dimension to discover temporal association rules. The star pattern is meaningful in temporal dimension, but may be not suitable in other dimensions. In this paper, we use a concept hierarchy to represent the intervals at varying levels of abstraction in each dimension and the belonging relationship between them.

2.4. Concept hierarchy

Concept hierarchy is a simple yet powerful form of background knowledge, which allows the user to view the data at more meaningful and explicit abstractions, and makes the discovered patterns easier to understand [14]. A concept hierarchy defines a sequence of mappings from a set of low-level concepts to higher-level, more general concepts [14], which helps expressing knowledge and data relationships in databases in concise high level terms, and thus, plays an important role in knowledge

discovery processes [12]. The concept hierarchy is usually partially ordered according to a general-to-specific ordering, and often defines a taxonomy or lattice represented. Consider a concept hierarchy for the dimension temporal in fig. 2.15, the most specific concept is month, where the month values for temporal include form January to December. Each month, however, can be mapped to the season which they belong to. The most general concept is the null description ("ANY").

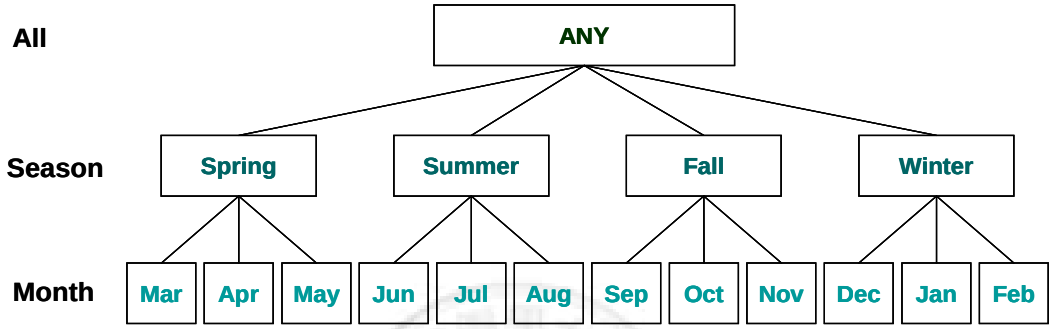

Figure 2.15 An example of temporal hierarchy

Concept hierarchies could be provided manually by users, knowledge engineers, domain experts, embedded in some data relations, or automatic generated based on statistical analysis of the data distribution. Different concept hierarchies can be constructed on the same attribute(s) based on different viewpoints or preferences [12].

There are four major types of concept hierarchies: *schema hierarchies*, *set-grouping hierarchies, operation-derived hierarchies* and *rule-based hierarchies*. A *schema hierarchy* is a total or partial order among attributes in the database schema. Typically, a *schema hierarchy* specifies a data warehouse dimension [14]. An example of hierarchical and lattice structures of attributes in warehouse dimensions is shown in fig. 2.16. A *set-grouping hierarchy* organizes values for a given attribute or dimension into groups of constants or range values [14]. An example of a set-grouping hierarchy for the dimension salary is shown in fig. 2.17. An *operation-derived hierarchy* is based on operations specified by users, experts, or the data mining system, and a

rule-based hierarchy occurs when either a whole concept hierarchy or a portion of it is defined by a set of rules and is evaluated dynamically based on the current database data and the rule definition [14].

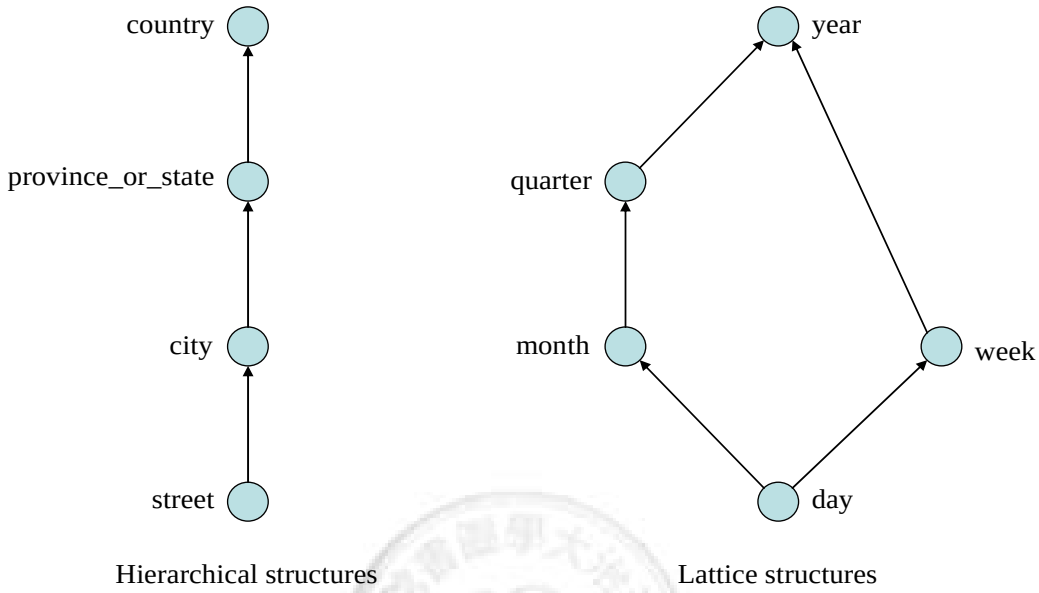

Figure 2.16 Hierarchical and lattice structures of attributes in warehouse dimensions

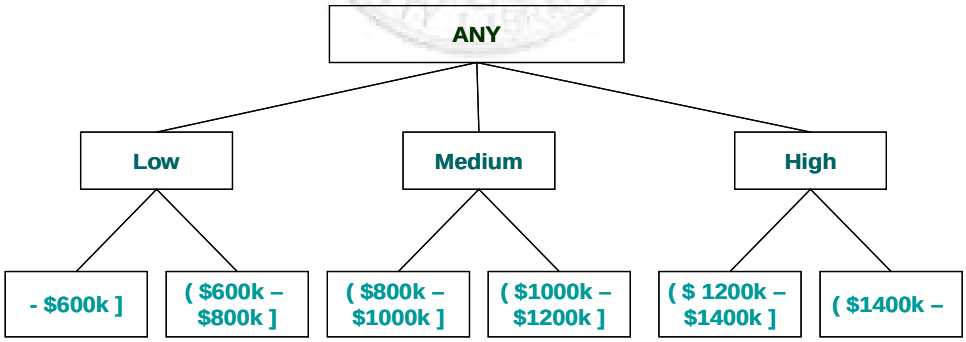

Figure 2.17 An example of salary hierarchy