# 4. Algorithm

The outline of our algorithm is shown in Fig. 4.1. The input of our algorithm consists of a multi-dimension transaction database **MD**, a set of concept hierarchies for each dimension $\mathbf{CH_x}$(x = 1 to n), a minimal support *minsup*, a minimal confidence *minconf*, and a match ratio *m* (in relaxed match case). The output of our algorithm is all the *multi-dimension rules w.r.t full (or relaxed match)* in **MD**. Instead of modifying the algorithm in [33] intuitively, we decompose our algorithm into two independent steps: (1) finding all association rules in each element segmentation, and (2) updating all combination segmentations by the output of step1. We can adopt any available algorithms (such as [11][23][27]) in step1, and segregating the two steps will make our mining task more flexible in a distribution environment.

```
1)  Input:
2)      Multi-dimension transaction database: MD;
3)      concept hierarchies for each dimension: $\mathbf{CH_x}$ (x = 1 to n);
4)      user define threshold: minsup, minconf, match ratio m;
5)  Procedure:
6)      Phase0:
7)          generate all $E_i$ and $G_j$ by $\mathbf{CH_x}$ (x = 1 to n);
8)          build the pattern table;
9)      Phase1:
10)         for all Ei
11)             discover all association rules r in $T[E_i]$ as $R_{Ei}$;
12)     Phase2:
13)         for all $E_i$
14)             for all $G_j$ that $E_i \subset Gj$
15)                 update $R_{Gj}$ using $R_{Ei}$;
16)     Phase3:
17)         for all $G_j$
18)             for all r (which satisfy m) in $R_{Gj}$
19)                 output ($G_j$, r);
20) Output:
21)     All multi-dimension association rules (p, r);
```

**Figure 4.1 Outline of our algorithm.**

In our algorithm, we discover all association rules $R_{Ei}$ in *element segmentation* $T[E_i]$ for each *element pattern* $E_i$, and then use $R_{Ei}$ to update $R_{Gj}$, a set of *multi-dimension association rules*, for every *generalized pattern* $G_j$ which covers $E_i$. The tasks of each *element pattern* are mining large itemsets within them and tell their super *generalized patterns* "I support these rules to be multi-dimension association rules". The task of each *generalized pattern* is deciding which rules hold within it according to the opinion coming from the *element patterns* which belong to them.

Note that, instead of mining association rules in every segmentation, we only implement mining process in each *element segmentation* and use the output of them to determine which rules hold in the *combination segmentations* they belong to. Consider two *multi-dimension patterns*: <2004 spring, Branch01, Student> and <2004 spring, Branch01, Male>, where the two patterns have an overlapping *element pattern* <2004 spring, Branch01, Student, Male>. In our algorithm, we use the association rules in segmentation enclosed by <2004 spring, Branch01, Student, Male> (and by other related *element patterns*) to derive the multi-dimension association rules in segmentations enclosed by <2004 spring, Branch01, Student> and by <2004 spring, Branch01, Male> to avoid the duplicate database scans. Following, we will discuss some important elements of our algorithm.

## 4.1. Generate all patterns and pattern table

Before starting the mining process, we generate all element and generalized patterns by the input set of concept hierarchies first, and then build a pattern table to store the belonging relationship between the element and generalized patterns. Given a set of concept hierarchies, we can generate a multi-dimension pattern by choosing a node in each concept hierarchy. The combination of different choices can form all the multi-dimension patterns. For example, given two concept hierarchies as Fig. 4.2, there are six element patterns: (Mar, Male), (Mar, Female), (Apr, Male), (Apr, Female), (May, Male), (May, Female), and six generalized patterns: (Spring), (Spring, Male), (Spring, Female), (Mar), (Apr), (May), twelve multi-dimension patterns produced from them totally.

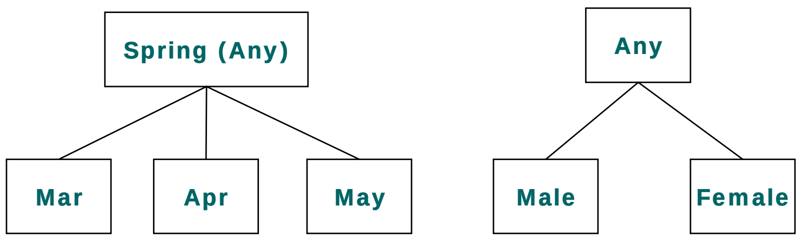

**Figure 4.2 Two given concept hierarchy.**

The belonging relationships between patterns can be presented in a lattice structure as in Fig. 4.3. We store the relationships in a bit map in which there are element patterns and generalized patterns, and "1" indicate that the corresponding element pattern belong to the corresponding generalized pattern, and "0" otherwise. Such bit map which stores the relationships in Fig. 4.3 is shown in Fig. 4.4. This kind of bit map which stores the belonging relationships between patterns called a pattern table in our algorithm.

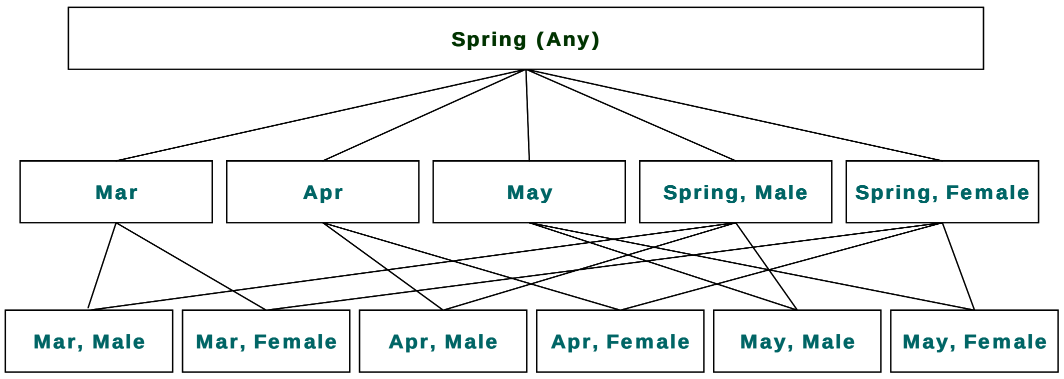

**Figure 4.3 belonging relationships between patterns.**

| | (Mar) | (Apr) | (May) | (Spring, Male) | (Spring, Female) | (Spring) |
|---|---|---|---|---|---|---|
| (Mar, Male) | 1 | 0 | 0 | 1 | 0 | 1 |
| (Mar, Female) | 1 | 0 | 0 | 0 | 1 | 1 |
| (Apr, Male) | 0 | 1 | 0 | 1 | 0 | 1 |
| (Apr, Female) | 0 | 1 | 0 | 0 | 1 | 1 |
| (May, Male) | 0 | 0 | 1 | 1 | 0 | 1 |
| (May, Female) | 0 | 0 | 1 | 0 | 1 | 1 |

**Figure 4.4 A pattern table for concept hierarchies in Fig. 4.2.**

## 4.2. Update process

After all the patterns and pattern table be generated, we begin to read the transactions in each element segmentation to discover all the association rules in phase1. We adopt the algorithm carried out by [23] in this step. The detail of this algorithm has been discussed in section 2 and the outline of this algorithm has shown in Fig. 2.2. The output of phase1 is all the $R_{Ei}$ for element pattern $E_i$, which will be the input in phase2.

In the following phase, we update each $R_{Gj}$ using the $R_{Ei}$ come from phase1. The outline of the update algorithm for full match is shown in Figure 4.5.

```
1)  for all $R_{Ei}$
2)      for all $G_j \supset E_i$
3)          if ($R_{Gj}$ never be updated)
4)              $R_{Gj} = R_{Ei}$;
5)          else
6)              $R_{Gj} = R_{Gj} \cap R_{Ei}$;
```

**Figure 4.5 Update algorithm for full match**

For full match, this is done by intersection the set $R_{Gj}$ with the set $R_{Ei}$ where $E_i$ belongs to $G_j$ (if $R_{Gj}$ is updated for the first time, we let $R_{Gj} = R_{Ei}$). After all the intersection process, the association rule $r$ left in $R_{Gj}$ hold in all element segmentations covered by $T[G_j]$. There is an example shown in Fig. 4.6.

| (Spring, Male) | (Mar, Male) | (Apr, Male) | (May, Male) |
|---|---|---|---|
| | A→B<br>A→C<br>C→A<br>B→C<br>AB→C<br>AC→B<br>BC→A | A→B<br>A→C<br>B→A<br>C→D<br>AC→B<br>BC→A | A→B<br>A→C<br>D→C<br>AB→C<br>AD→C<br>BC→A |

(Spring, Male) **:**

| After update by (Mar, Male) | After update by (Apr, Male) | After update by (May, Male) |
|---|---|---|
| A→B<br>A→C<br>C→A<br>B→C<br>AB→C<br>AC→B<br>BC→A | A→B<br>A→C<br>AC→B<br>BC→A | A→B<br>A→C<br>BC→A |

**Figure 4.6 An example of update for full match**

The outline of the update algorithm for relaxed match is shown in Figure 4.7. For relaxed match, we associate a counter with each rule in $R_{Gj}$. When $R_{Ei}$ is used to update $R_{Gj}$, the counters of the rules that are in both $R_{Gj}$ and $R_{Ei}$ are incremented by 1, and the rules that are in $R_{Ei}$ but not in $R_{Gj}$ are added to $R_{Gj}$ with the counter set to 1. After all the update process, the association rule $r$ in $R_{Gj}$ whose count exceed $m|T[G_j]|$ holds in at least 100$m$% of the *element segmentations* $T[E_i]$ which are covered by $T[G_j]$, and thus ($G_j$, $r$) is a *multi-dimension association rule w.r.t. relaxed match* in **MD**. There is an example shown in Fig. 4.8.

```
1)  for all $R_{Ei}$
2)      for all $G_j \supset E_i$
3)          for all $r$ in $R_{Ei}$
4)              if ($r \notin R_{Gj}$)
5)                  add $r$ to $R_{Gj}$;
6)                  $R_{Gj}.r$.count = 1;
7)              else
8)                  $R_{Gj}.r$.count++;
```

**Figure 4.7 Update algorithm for relaxed match**

| (Spring, Male) | (Mar, Male) | (Apr, Male) | (May, Male) |
|---|---|---|---|
| | A→B | A→B | A→B |
| | A→C | A→C | A→C |
| | C→A | B→A | D→C |
| | B→C | C→D | AB→C |
| | AB→C | AC→B | AD→C |
| | AC→B | BC→A | BC→A |
| | BC→A | | |

(Spring, Male) **:** match ratio m = 0.667

| After update by (Mar, Male) | After update by (Apr, Male) | After update by (May, Male) |
|---|---|---|
| A→B : 1 | A→B : 2 | A→B : 3 |
| A→C : 1 | A→C : 2 | A→C : 3 |
| C→A : 1 | B→A : 1 | ~~B→A : 1~~ |
| B→C : 1 | C→D : 1 | ~~C→D : 1~~ |
| AB→C : 1 | C→A : 1 | ~~C→A : 1~~ |
| AC→B : 1 | B→C : 1 | ~~B→C : 1~~ |
| BC→A : 1 | AB→C : 1 | ~~D→C : 1~~ |
| | AC→B : 2 | AB→C : 2 |
| | BC→A : 2 | AC→B : 2 |
| | | BC→A : 3 |
| | | ~~AD→C : 1~~ |

**Figure 4.8 An example of update for relaxed match.**

## 4.3. Rules output

For full match, we output all the ($G_j$, r) pair for every $r$ left in each $R_{Gj}$. For relaxed match, we output all the ($G_j$, r) pair for every $r$ in each $R_{Gj}$ whose count exceed $m|T[G_j]|$. Our algorithm above can avoid losing interesting rules which only hold in several segmentations, and avoid the output multi-dimension association rules not really hold in all the range of their domain. For example, suppose we have a *generalized pattern* <2004 spring, Branch001, Student, Female, Youth> which covers three *element patterns* <2004 Mar, Branch001, Student, Female, Youth>, <2004 Apr, Branch001, Student, Female, Youth>, and <2004 May, Branch001, Student, Female, Youth>. Suppose that the *minsup* is 10%, and the *minconf* is 70%. For the full match case, we can be sure that the rules which hold in only two months of spring but fail in the other will never be rules with respect to whole spring (under the conditions of other pattern items).