DEVELOPING ASSOCIATION RULES IN FACILITIES MANAGEMENT DATABASES

G. Goulbourne, F. Coenen, P. Leng
Department of Computer Science, The University of Liverpool,
Chadwick Building, P.O. Box 147, Liverpool L69 3BX, England.
D. Murphy
Royal and Sun Alliance, PO Box 144, New Hall Place,
Liverpool L69 3EN, England.

September 16, 1998

Abstract

Data mining is a recent development in the field of database exploration. A powerful and well investigated topic is the discovery of association rules. This paper examines the use of association rules within the domain of facilities management. Within such a domain the application of association rules offers a way of identifying relationships between sets of data which may previously have been thought to be completely unrelated. We describe algorithms for the development of association graphs and expansion trees to identify such relationships in response to loosely defined user queries. This provides a means to examine many possible relationships prior to exposing an interesting one.
Key words: Association rules, Facilities Management, Directed Graph, Expansion Tree.

1 INTRODUCTION

Recent advances in the fields of mass data storage and network reliability have led to an ever expanding universe of data. From the largest multinational companies to the average small business the collection of data about every aspect of an operation's activity has never been easier or cheaper. As a consequence, the greatest difficulty in management decision making and planning may now arise not from lack of information, but from a surfeit. Knowledge discovery techniques aim to take this universe (or glut), and produce meaningful, interesting and beneficial data from it. Of course, meaningful, interesting and beneficial are all subjective concepts. Usually, the role of knowledge discovery within a business environment involves maximising some aspect of the operation, which may be profit, or some measure of efficiency or service.

In this paper we consider the application of Knowledge Discovery in Databases (KDD) techniques to the domain of Facilities Management (FM). Facilities Management is characterised by a great diversity of function, most of which is only peripherally related to the core objectives of the organisation, and elements of which are essentially unrelated to each other. This diversity of function is reflected in a corresponding diversity of information and data organisation. Many of the information systems thus involved may also be legacy systems which have evolved within the organisation over a long period of time, and tend to be large, developed using old technology, complex and critical to the organisation's day-to-day working.

The overall aim of our research is to investigate which data mining techniques are best suited to the FM operation. The key issue here is the identification of previously unrecognised information from disparate databases. Our starting point is generating *association rules* [1]. We develop the work of Yen and Chen [6] who use association graphs and expansion trees to explore interesting relationships between attributes in databases. We seek to extend and generalise this work, so that it is applicable to a wider range of databases, especially those found within an FM environment, and we present improved algorithms for the derivation of association rules,

The organisation of the paper is as follows. We begin with an outline of the domain of facilities management, followed in Section 3 by an explanation of association rules. In section 4 we present a hypothetical data mining scenario from the facilities management domain, and use this to illustrate the application of the technique. In Section 5 we describe algorithms which formalise the approach, and in Section 6 present our conclusions and proposals for future work.

2 FACILITIES MANAGEMENT

The FM operation of a company can be defined as an integrated approach to operating, maintaining, improving and adapting the buildings and infrastructure of an organisation in order to create an environment that strongly supports the primary objectives of that organisation (Further Education Funding Council [3]). Essentially, therefore, FM attempts to bring together under one organisational heading a number of operations which are largely functionally independent. These operations, for an organisation whose core business is (say) retailing of goods and services, might include: planning and design of premises; estate management; building maintenance; catering; staff travel and accommodation; car fleet management; printing services; and any other activities which are not intrinsic to the principal business function. In practice, within most large organisations, it will be necessary for each such operation to be managed separately. As an example, the Fleet Unit within an FM operation would be responsible for all aspects of vehicle policy within the company including types of vehicle available, delivery times, breakdown and recovery service, and maintenance; i.e. every aspect of running a fleet of perhaps 5000 or more vehicles. The scale and scope of this operation demands in itself a substantial organisational infrastructure which may be quite separate from that of the parent business. This tends to militate against holistic approaches to customer service, financial management, and other issues which are orthogonal to the functional organisation of the business. It is this problem which Facilities Management seeks to address.

An inherent characteristic of the FM domain is a great diversity of information, separate components of which may be developed and managed quite independently. It is this diversity which suggests the potentially important role of KDD in this context, while also posing corresponding problems for its implementation. Knowledge Discovery offers the possibility of identifying associations between information that is held in different repositories and which has no defined relationships. Out of these associations may emerge new possibilities for more efficient and effective management of the overall operation.

3 Association Rules

An association rule is a relation of the form $A \Rightarrow B$ where A and B represent ordered sets (lists) of Boolean valued attributes in the database table. Note, A and B are distinct and the cardinality of each must be greater than or equal to 1. Thus if R is the total set of attributes;

$A = \{x \mid x : X, X \subseteq R, X \neq \{\}\}$ and

$B = \{y \mid y : Y, Y \subseteq R, X \neq \{\}\}$ and

$A \cap B = \{\}$

Given a particular association rule and a set of boolean values for the set A; If all the attributes in A, (the antecedent or tail) evaluate to true, then the existence of the association rule enables us to infer that all the attributes in B (the consequent or head) are likely to be true. Note that we are here dealing with "rules" which support broad and approximate inferences. To enable the effective filtering of rules two measures are applied, *support* and *confidence*. The support for an attribute, or set of attributes A is defined as;

$$\frac{number\ of\ rows\ in\ table\ where\ A = true}{Number\ of\ rows\ in\ table}$$

The confidence in a rule is defined as;

$$\frac{support\ for\ A \Rightarrow B}{Support\ for\ A}$$

The user is free to set any desired level of support and confidence, however, if they are set too low we risk being overwhelmed with rules of little interest, if they are set too hgh, we may miss important rules. The generation of a minimum number of *candidate* sets is also an important factor, these are the sets of attributes from which the rules will be constructed. For reasons of efficiency we wish to minimize the number of these sets.

In this paper we have set support threshold at at 20% and confidence at 80%. When the support for an attribute is $\geq$ the threshold the set is said to be *covering* or *frequent* [1].

The process of determining valid association rules is dependent upon knowledge of the nature of the set $\{A \cup B\}$. We can identify three possible situations here:

Type 1: We know precisely which attributes we are interested in and therefore are only interested in association rules that link these attributes, i.e. the set $\{A \cup B\}$ has a given value.

Type 2: We know a "minimum" set of attributes in which we are interested and wish to find association rules that relate this minimum set including rules that incorporate additional attributes that were not part of the minimum set, i.e. the set $\{A \cup B\}$ is "open ended".

Type 3: We have no idea of what attributes we are interested in and wish to find all association rules that may exist within the database.

This categorisation is identical to that identified by Yen and Chen.

4 Rules in the FM Domain

To illustrate the proposed approach we will present a number of hypothetical FM database tables combined into a single "data warehouse" from which we will derive interesting association rules by considering each of the three cases identified above in turn.

The domain of facilities management is a particularly relevant area for the application of association rules. This is primarily because of the operational and managerial independence of each of the sub divisions. This independence tends to exclude the synergies enjoyed by more tightly coupled sub-divisions. For example, the sales and marketing operations may combine to produce an overall strategy, but the fleet and canteen operations (within FM) have very little common ground and will tend to operate in an autonomous manner. The exploration of the combined fleet/canteen databases may lead to the discovery of hitherto unknown/unforeseen relationships. In this context, association rules will be used to answer "vague" queries such as, *"is there any relationship between the consumption of food in the staff canteen and the allocation of car parking permits"* If the user is to pose such queries against separate databases then the system must be able to reconcile any differences between the data, if this is possible, then present some queries which are possible to the user. Once the query has been validated then the user can select the support and confidence levels to be applied. In table 1 the hypothetical FM databases have been

[1] Sometimes called *large*

S	G	M	P	B	L	N
1	0	1	1	0	0	Sebastian
1	0	1	1	1	1	Sebastian
1	0	1	1	1	1	Sebastian
1	1	1	1	1	1	Sebastian
1	1	1	1	0	0	Sebastian
1	1	1	1	0	0	Sebastian
1	1	1	1	0	0	Sebastian
1	1	1	1	1	1	Sebastian
0	0	0	1	1	1	Prospero
0	0	0	1	1	1	Prospero
0	0	0	1	1	1	Prospero
1	0	1	1	0	0	Trinculo
1	0	1	1	0	0	Trinculo
1	0	1	1	0	0	Trinculo
1	0	1	1	0	0	Trinculo
0	0	0	0	0	0	Antonio
0	0	0	0	0	0	Antonio
0	0	0	0	0	1	Antonio
0	0	0	0	0	0	Antonio
0	0	0	0	1	0	Antonio
1	0	0	1	1	0	Caliban
1	0	0	1	0	0	Caliban
1	0	0	1	0	0	Caliban
1	0	0	1	0	0	Caliban

Table 1: "Data Warehouse"

integrated namely, catering (attributes, B breakfast and L lunch), fleet (attribute P, permit), telephony (attribute M, mobile phone), human resources (attribute G, gym member) and accounts (attribute S bonus scheme). The value 1 in the record indicates that the person has consumed a breakfast, has a parking permit etc. It is likely that the databases to be used will require attribute partitioning to arrive at a boolean representation, however, this issue is beyond the scope of this paper.

An initial inspection of Table 1 indicates support as follows:

S = 67 %, G = 21 %, M = 50 %, P = 79 %, B = 38 %, L = 33 %.

Thus attribute S has a value of "true" (1) in 16 of the 24 rows in the table. Similarly attribute G has a value of "true" (1) in 5 of the 24 rows in the table, and so on. All these individual support values are above the assumed 20% threshold (although G is only just above this value) and thus all these attributes are potentially components of association rules. We refer to this set of attributes, for which there is adequate support, as the "base list". Thus:

$baseList = [S, G, M.P, B, L]$.

Note that the ordering, although arbitrary, is significant for algorithmic purposes; this will be illustrated in sub-sections 5.2 and 5.3. From this base list we wish to identify one or more "total sets" of attributes which may exhibit interesting relationships.

5 Developing Rules

5.1 Type 1 Request

In type 1 scenarios we know precisely what attributes we are interested in. For example let us assume that we are interested in the set {P,B,L}, i.e. we wish to explore the relationships between food and parking permits. From Table 5 the support for the set {P,B,L} is $7/24 = 39\%$. Thus {P,B,L} has enough support to render it interesting with respect to the defined 20% threshold value. Consequently the rules listed below are all candidate association rules. However, to decide whether they are valid or not we must also compute the confidence values for each rule:
{PB} $\Rightarrow L = 88\%$,
$\{PL\} \Rightarrow B = 100\%$,
$\{BL\} \Rightarrow P = 100\%$,
$\{P\} \Rightarrow BL = 37\%$,
$\{B\} \Rightarrow LP = 78\%$,
$\{L\} \Rightarrow BP = 100\%$.

There are four rules which have a confidence value equal to or greater than the threshold confidence value of 80%; namely rules 1, 2, 3 and 6. Thus (from rule 3), if an employee has breakfast and lunch we can predict with a confidence of at least 80% that they would also have a parking permit. To verify these rules they should be tested on an unseen data set.

5.2 Type 2 Request

For a type 2 request we start with a core set of attributes from the total set, the rest of the contents of which are unknown. We therefore commence by identifying all interesting total sets which include the core set. This is achieved by first creating an "association graph" and then using this to develop an "expansion" tree the nodes of which represent all the interesting sets which include the core set.

To create the association graph we commence with the first attribute in baseList. In this case the attribute is S and it has support greater than the threshold, i.e. 67%; this is then the start vertex in the graph. If S did not have the required support the next attribute G would be a candidate for the start vertex. If no attribute has the required support then no rules can be generated. In this case, using the attribute S as a start vertex, we examine the combination of this attribute with every other attribute in the ordered set, starting with the successor to S. From this we identify the couplings $\{SG\}$,$\{SM\}$,$\{SP\}$,$\{SR\}$,$\{SL\}$. The support for each is as follows; {SG} = 21 %, {SM} = 50 %, {SP} = 67 %, {SB} = 21 %, {SL} = 13 %.

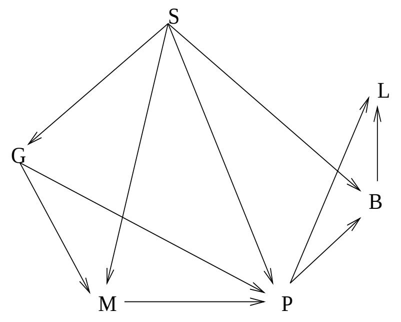

Figure 1: Directed Association Graph

All but the last coupling is above the required threshold, and thus represent edges in the association graph. We then move to the successor to S, (i.e. G) and examine the couplings, GM, GP, $\ldots GL$. This process continues until all the couplings have been evaluated. The result is the association graph shown in figure 1.

The significance of the association graph is that it indicates all the sets of attributes from which association rules can be derived. These are those sets for which all the attributes in the set are connected to each other. Such connectivity can be established by inspection of the graph. For example the set {S,G,M,P} is a candidate set whilst {S,P,B,L} is not (there is no edge between S and L). This feature is expressed using the following lemma:

Lemma 1: *A necessary condition for a set of attributes from which association rules may be derived is that an edge must exist, in the association graph, between all the nodes (attributes) in the graph which are included in the set.*

However, this is only a necessary condition, and not a sufficient condition. At this point our algorithm departs from that of Yen and Chen in that the number of candidate sets generated by our algorithm can be considerably less. This is achieved by the fact that, if and only if, a group of vertices and edges make a fully connected sub-graph are they considered as a candidate set. An example of this improvement can be seen using the association graph in figure 1. If the edge connecting M and P is examined, Yen and Chen would generate the candidate sets, {M,P,B} and {M,P,L}, for which there is insufficient support - there is no edge MB or ML in the graph. The proposed algorithm would not generate either of these sets as candidate sets because the required edges do not appear in the graph. In the example given in figure 1 there is a 50% reduction in the number of candidate sets generated. The reduction in the number of candidate sets is dependent on the graph in question, however, in the worst case thre will be no reduction, (a fully connected graph), in the best case we would achieve a 100% improvement, (a graph where each vertex is only connected to one other).
Having identified a candidate set we must then establish that there is appro-

priate support. The direction of the edges in the graph are not significant with respect to establishing candidate sets, it is only necessary to establish that all attributes in the set are connected to one another, the direction of the edges is not important. However, from a procedural standpoint, the direction of the edges serves to prevent the generation of duplicate sets - this is an important computational efficiency consideration given the likely size of the data warehouses we wish to consider.

Once the association graph has been produced the next stage is to generate the expansion tree using the information contained in the graph. The root node for this tree is the user supplied core set, the expansion then originates from this node so that all possible sets which include the core set are identified. To do this a procedure $P_{expandTree}$ has been defined which recursively manipulates three lists of attributes ordered according to the ordering contained in baseList as follows:

start list: Which at the commencement of the process contains the core set.

end list: In which will the identified sets will be contained, but which at the commencement of the process will be empty.

attribute list: In which all attributes to be considered for possible inclusion in a set are contained. At the commencement of the process this will be equivalent to the base list.

On each iteration of $P_{expnadTree}$ there are five possible outcomes:

1. Both the attribute and start lists are empty, therefore the attributes in end list are a valid set. Output and return.
2. The start list is empty (but not the attribute list) call procedure $P_{xpnasionTreeEnd}$. On completion of this procedure return.
3. The head (i.e. the first element) of the attribute list is the same as the head of the start list. Move the head of the start list to the end of the end list, remove the attribute in question from the attribute list and recur.
4. The head of the start list is before (according to the ordered baseList) the head of the attribute list. Move the head of the start list to the end of the end list and recur.
5. (Default) the head of the attribute list is before (according to the ordering in baseList) the head of the start list. Call $P_{testSet}$ which determines if there are appropriate links in the graph and, if so, whether there is adequate support. If this is the case a new branch in the tree is created and processed in the same manner. On returning from $P_{testSet}$ remove the head of the attribute list and recur again.

The $P_{testSet}$ procedure first tests whether there are edges in the association graph from all attributes in the end set (if any) to the head of the attribute list

```
procedure P_typeTwoExpandTree(startNode)
begin
    P_expandTree([],startNode,baseList)
end

procedure P_expandTree(endList,startList,attributeList)
begin
if (attributeList = []  and startList = [])  output endList and return
else if (endList = []) P_expandTreeEnd(endList,attributeList)
else if (startList head = attributeList head)
    endList :=  endList plus start List head , startList :=  startList minus startList head
    attributeList :=  attributeList minus attribute List head
    P_expandTree(endList,startList,attributeList)
else if (startList head < attributeList head)
    endList :=  endList plus startList head , startList :=  startList minus startList head
    P_expandTree(endList,startList,attributeList)
else
    P_testSet(endList,startList,attributeList)
    attributeList :=  attributeList minus attribute list head
    P_expandTree(endList,startList,attributeList)
end

procedure P_expandTreeEnd(node,attributeList)
begin
if (attributeList = [])  output endList and return
else
     if testSetFrom(endList,attributeList)  and  adequate support
            newEndList :=  endList plus attributeList head
            newAttributeList :=  attributeList minus attribute list head
            P_expandTreeEnd(newEndList,newAttributeList)
attributeList :=  attributeList minus attribute list head
P_expandTreeEnd(endList,startList,attributeList)
end

procedure P_testSet(head,tail,attributeList)
begin
     if F_testSetFrom(head,attributeList[0]) = TRUE  and
             F_testSetTo(attributeList[0],tail) = TRUE  and adequate support
         head :=  head plus attributeList[index]
         attributeList := attributeList[1] ..  attributeList end
         P_expandTree(head,tail,attributeList)
end

function F_testSetFrom(node,attribute)  return Boolean
begin
       Return TRUE if node is empty, or if arcs exists from all elements of
       node to attribute. FALSE otherwise.
end

function F_testSetTo(attribute,node)  return Boolean
begin
       Return TRUE if node is empty, or if arcs exists from attribute to all
       elements of node. FALSE otherwise.
end
```

Table 2: Type 2 Algorithm

using a function $F_{testSetFrom}$ which returns TRUE or FALSE as appropriate. If this returns TRUE similar tests are carried out, using a function $F_{testSetTo}$ to ensure that there are edges from the head of the attribute list to all attributes in the start set (if any). Note the significance of the direction of the edges here - in this manner we ensure that duplicate sets are not generated. If this second test is also successful we test that there is adequate support for the union of the start list, the end list and the head of the attribute list. If so we move the head of the attribute list to the end of the end list and call $P_{expandTree}$ with this new list grouping. If any of the tests fail we simply return from the procedure.

The $P_{expandTreeEnd}$ procedure operates in a similar but simplified manner to $P_{expandTree}$, using only the end and attribute lists (the start list having been exhausted by the calling procedure). On each recursion there are two possible outcomes:

1. The attribute list is empty, therefore the attributes in end list are a valid set. Output and return.
2. Test, using $f_{testSetFrom}$, whether there are edges in the association graph from all attributes in the end set to the head of the attribute list. If so test whether there is adequate support, and if so move the head of the attribute list to the end of the end list and call $P_{expandTree}$ with these new list to start a new branch in the tree. On returning from this call recur with the head of the attribute list removed.

The complete type 2 expansion tree algorithm is presented in Table 2. The generation procedure can best be illustrated by considering an example. Let us assume a core set of {S,P}. At the commencement of the procedure the following instantiation will be made:

$$\begin{aligned} endList &= [] \\ startList &= [S,P] \\ attributeList &= [S,G,M,P,B,L] \end{aligned}$$

For ease of understanding we will group these lists together as one compound list thus [[],[S,P],[S,G,M,P,B,L]]. On the first call to $P_{expandTree}$ the attribute S will be moved from the head of the start list to the end of the end list, and removed from the attribute list to give [[S],[P],[G,M,P,B,L]]. On the second call we discover that G is before P and thus we call the $P_{testSet}$ procedure. Inspection of the association graph (Figure 1) shows that there are edges from S to G and G to P therefore the set {S,P,G} is a candidate set. The support for this set is 21% (5/24) therefore a new branch can be commenced with the

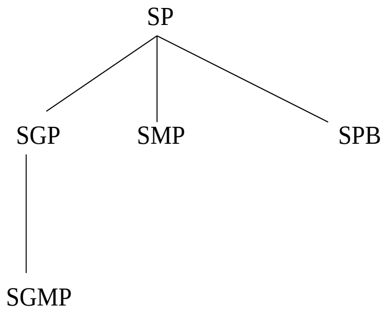

Figure 2: Expansion Tree for Type 2 Query

compound list [[S,G],[P],[M,P,B,L]]. Attribute M is also before P therefore we make a second call to the test Set procedure. There are edges from S and G to M, and from M to P therefore the set {S,P,G,M} is another candidate set. The support is again 21% (5/24) therefore another branch can be created and the $P_{expandTree}$ procedure called with [[S,G,M],[P],[P,B,L]]. The head of the attribute and start lists are now the same, so we recur with [[S,G,M,P],[],[B,L]]. The start list is now exhausted so we go on to invoke $P_{expandTreeEnd}$ with the compound list [[S,G,M,P],[B,L]]. On the first iteration the set {S,G,M,P,B} will be tested as a potential candidate set, however there are no edges from G and M to B so the test returns FALSE and we will recur with [[S,G,M,P],[L]]. There are no edges from S, G and M to L, thus we recur with [[S,G,M,P],[]]. The attribute list is now empty therefore out put the node {S,G,M,P} (Figure 2).

We now back track to where the current branch was commenced and recur with the compound list [[S,G],[P],[P,B,L]]. The next iteration will be with [[S,G,P],[],[B,L]], followed by a call to $P_{expandTreeEnd}$. Here it will be established that neither B or L can form part of a valid set with {S,G,P} (support = 5/24 = 21%), however the set {S,G,P} itself will be output as a valid set in its own right. We will then back track to where this branch was commenced and recur with [[S],[P],[M,P,B,L] and continue the process. In this manner we will establish that {S,M,P} (support = 12/24 = 50%) and {S,B,P} (support = 5/24 = 21%) are also valid sets. The core set {S,P} is also of course a valid set because it appears in the association graph. The sequence of "calls" required to produce the expansion tree for the core set {S,P} given in Figure 2 are listed in Table 3. The line numbering indicates the sequence of calls. The indenting indicates the level in the tree at which the calls are made.

Possible association rules can now be generated from any of the sets in the resulting expansion tree. Recall that a valid association rule is one for which there is appropriate confidence.

1) CALL $P_{expandTree}$([],[S,P],[S,G,M,P,B,L])
2) CALL $P_{expandTree}$([S],[,P],[G,M,P,B,L])
  2.1) CALL $P_{expandTree}$([S,G],[P],[M,P,B,L])
    2.1.1) CALL $P_{expandTree}$([S,G,M,],[P],[P,B,L])
    2.1.2) CALL $P_{expandTree}$([S,G,M,P],[],[B,L])
      2.1.2.1) CALL expandTreeEnd([S,G,M,P],[B,L])
      2.1.2.2) CALL expandTreeEnd([S,G,M,P],[L])
      2.1.2.3) CALL expandTreeEnd([S,G,M,P],[])
      Found set $\{S,G,M,P\}$ *return*
  2.2) CALL $P_{expandTree}$([S,G,],[P],[P,B,L])
  2.3) CALL $P_{expandTree}$([S,G,P],[],[B,L])
    2.3.1) CALL expandTreeEnd([S,G,P],[B,L])
    2.3.2) CALL expandTreeEnd([S,G,P],[L])
    2.3.3) CALL expandTreeEnd([S,G,P],[])
    Found set $\{S,G,P\}$ *return*
3) CALL $P_{expandTree}$([S],[P],[,M,P,B,L])
  3.1) CALL $P_{expandTree}$([S,M],[P],[P,B,L])
  3.1) CALL $P_{expandTree}$([S,M,P],[],[B,L])
    3.3.1) CALL expandTreeEnd([S,M,P],[B,L])
    3.3.2) CALL expandTreeEnd([S,M.P],[L])
    3.3.3) CALL expandTreeEnd([S,M,P],[])
    Found set $\{S,M,P\}$ *return*
4) CALL $P_{expandTree}$([S],[P],[P,B,L])
5) CALL $P_{expandTree}$([S,P],[],[B,L])
  5.1) CALL expandTreeEnd([S,P],[B,L])
    5.1.1) CALL expandTreeEnd([S,P,B],[L])
    5.1.2) CALL expandTreeEnd([S,P,B],[])
    Found set $\{S,P,B\}$ *return*
  5.2) CALL expandTreeEnd([S,P],[L])
  5.3) CALL expandTreeEnd([S,P],[])
  Found set $\{S,P\}$ *return*

Table 3: Sequence of Calls to Produce Expansion tree for Core Set {S,P} Using Type 2 Expand Tree Procedure

```
procedure P_typeThreeExpandTree(arcList)
begin
  index i := 0
   while (i < arcList end)
     attributeList = makeAttributeList(node)
     P_expandTreeEnd(arcList[i],attributeList)
     i++
end

function F_makeAttributeList(node)  return
     attributeList
begin
  attributeList := { baseList[n]..baseList end }
        where baseList[n-1] = node end
end
```

Table 4: Type 3 Algorithm

5.3 Type 3 Request

A type 3 scenario is one where we wish to identify all possible association rulers contained in the data warehouse. To identify sets of attributes from which such rules can be derived we again make use of the association graph concept as described above and derive expansion trees for each node pair in the graph. Again, to prevent the generation of duplicate sets, full use is made of knowledge of the ordering of the attributes in baseList. We commence by defining a variable arc list which is instantiated with a list of all node pairs in the graph. Thus:

$$\begin{aligned} arcList \;:=\; & [[S,G],[S,M],[S,P],[S,B],[G,M], \\ & [G,P],[M,P],[P,B],[P,L],[B,L]] \end{aligned}$$

We then step through this graph using each pair as a start list. However, we cannot use the $P_{expandTree}$ procedures to produce the tree because this would generate duplicates. For example the node pairs [S,G], [S,M] and [G,M] would all produce the set {S,G,M} (amongst others). Instead we use the $P_{expandTreeEnd}$ procedure defined in Sub-section 5.2 but with the endList instantiated with the node pair under consideration (ordered) and the attribute list defined as follows:

$$attributelist = baseList[i+1]..baseListend$$

where baseList[i] is the second attribute of a given node pair. Thus if we are processing the node pair [S,P] the attribute list would be [B,L]. The type three tree expansion algorithm (where it varies from the type two algorithm) is presented in Table 4.

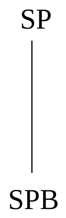

Figure 3: Expansion Tree for Type 3 Query

We will illustrate the procedure by considering the node pair [S,P]. The $P_{expandTreeEnd}$ procedure will initially be called with the compound list [[S,P],[B,L]]. On the first recursion we will establish that {S,P,B} is a valid set, start a new branch in the tree and call type $P_{expandTreeEnd}$ with [[S,P,B],[L]]. We then establish that {S,P,B,L} is not a valid set (there is no edge from S to L in the association graph) and recur with [[S,P,B],[]]. The attribute list is now empty therefore {S,P,B} is a node in the tree (Figure 3). We now back track and recur with [[S,P],[L]]. {S,P,L} is not a valid set (again there is no edge between S and L), therefore we recur with [[S,P],[]]. The attribute list is now empty therefore {S,P} is also a node in the tree.

Comparison between Figures 2 and 3 show that, although in both cases we started with the same start set, the two algorithms produce different trees. The type three algorithm will produce the remaining nodes in Figure 2 (amongst others) when the node pairs [S,G] and [S,P] are expanded.

6 Conclusion and Further Work

We have demonstrated the application of association rules to the domain of facilities management. In particular, we have highlighted how the use of association rules can illuminate relationships between data sets which may appear initially to have little or nothing in common. Our algorithm illustrates how an improvement in candidate set generation can be achieved. The end product of our work has been the generation of rules which could be tested on unseen data. The advantage of rules is that they are fairly easy to understand by a non-specialist and the results from this work could easily be interpreted by a fleet/catering manager who was interested in analysing patterns.

The project will continue to use association rules and develop a rule base for the data sets involved. We will also be reviewing the graph construction and tree building stages of the work to try to increase the efficiency of each stage.

7 Acknowledgements

The work described was carried out as part of the KDD for FM - Knowledge Discovery in Databases for Facilities Management - project (http://www.csc.liv.ac.

uk/ ˜graham_g/kdfm.html) supported by Royal-SunAliance. Thanks also to D. S. Neary for his helpful contributions to this paper.

References

[1] Agrawal, R. Imielinski, T. Swami, A. Mining Association Rules Between Sets of Items in Large Databases. SIGMOD-93, 207-216. May 1993.

[2] Frost, R.A. Introduction to Knowledge Based Systems. Collins. 1986.

[3] Further Education Funding Council Effective Facilities Management: A Good Practice Guide. Her Majesty's Stationery Office. 1997

[4] Manilla, H. Toivonen, H. Verkamo, A.I. Improved Methods for Finding Association Rules. University of Helsinki, Department of Computer Science. Publication C, No. C-1993-65. 1994.

[5] Toivonen, H. Sampling Large Databases for Association Rules. Proceedings of 22nd VLDB Conference pp 134-145. 1996.

[6] Yen, S.J. Chen, A.L.P. An Efficient Data Mining Technique for Discovering Association Rules. DEXA. IEEE Computer Society Press. 1997.