A NOVEL PC-TREE FRAMEWORK FOR ASSOCIATION RULE MINING
Shruthi Prabhakara & Mikhil Masli
Advisor: Prof. Santhi Tilagam
Department of Computer Science
National Institute of Technology Karnataka

Abstract

ARM activity is one of the data mining activity has been recently attracted the attention of the database community. In this paper, we propose a scheme which uses both knowledge and a collection of databases as input for discovering associations. Finding of association rules across databases is called inter-association rules (IAR). Association Rule Mining activity is both computationally and I/O intensive. Association Rule Mining is a process of discovering items, which tend to occur together in transactions. Data mining across multi-databases is becoming increasingly important in support of this trend; we investigate the problem of mining association rules between multiple attributes with some minimum specified confidence from multi-databases and present an efficient algorithm for this purpose. In this paper, we address the multi-source and multi-destination mining problem across multi-databases: to identify relevant databases and relations and to integrate them to keep track of relationships of varying multiplicity. Our approach exploits the existing domain knowledge and the semantic relationship to link the set of source and target attributes via multiple navigation paths. We evaluate the strength of the links between the set of source attributes and target attributes based on several factors. A pattern-count tree is used as an intermediate structure to semantically partition the database. The links between the nodes are analyzed to establish the association rules.

1 Introduction to the problem:

Most of today's structured data is stored in relational databases. Thus, the task of mining from relational data has begun to receive significant attention in the literature. Since data to be handled by the data mining activity is very huge, methods to handle this activity should be efficient in terms of usage of computational resources like memory and processing time. There are many categories of data mining activity [Fayyad et al.]. One of the important activities is association rule mining (ARM). Association rules, first introduced in 1993 [ ], are used to identify relationships among a set of items in a database. These relationships are not based on inherent properties of the data themselves (as with functional dependencies), but rather based on co-occurrence of the data items. Original motivation for mining association rules is market basket analysis to improve the service provided by a supermarket, this is followed by other applications like fraud detection, selective marketing, financial forecast, optical character recognition (OCR), customer relationship management, attached mailing, medical diagnosis, text (web page) analysis and other data rich areas like digital libraries and biological databases.

The problem of finding association rules is partitioned into two sub-problems: (1) the identification of frequent sets of items and (2) the generation of association rules from those sets. It is well understood that the first sub-problem is computationally and I/O intensive and is directly related to the size of the database being mined. The mining of frequent patterns has significance not only in itself but also for other data mining tasks.
such as mining of association rules, correlations, sequences, episodes, classifiers and clusters.

We can categorize the ongoing work in Association Rule Mining (ARM) on the basis of number of databases being mined area as follows:

**Single Database based:** This category of algorithms is further subdivided based on the number of relations mined from.

i. Most of the ARM algorithms mine for associations in a transaction database and any transaction database may be viewed as a single relation. Such ARM algorithms are called transaction database based or **single database based algorithms**. These association rule algorithms generally differ on the generation of the candidates, counting of the support of a candidate itemset, number of scans over the database, and the data structures employed. Typically, these approaches concentrate on reducing the disk accesses while constructing large itemsets [4][5][8][9].

ii. Most real-world structured data are stored in multiple relations. If the mining is performed on more than one relation of a single database then corresponding ARM algorithms are called **multiple relations from a single database based mining algorithms**. These association rule algorithms aim at discovering interesting knowledge directly from multiple tables without joining data of multiple tables into a single table explicitly. Here finding the associations across relations is done by using the user defined attributes ExKeys, or using Foreign Keys across the relations.

**Multiple databases based:** Multiple databases may be homogeneous or heterogeneous in nature. Mining these multiple databases is therefore categorized as follows:

i. **Distributed Association Rule Mining (DARM)** works on a well-formatted data table stored at different sites. DARM algorithm’s main goal is finding a good data decomposition among the nodes and minimizing communication so that generating global association rules costs less than combining the participating sites' datasets into one centralized site.

ii. **Multi-database Association Rule Mining (MDARM)** works on multiple heterogeneous databases, each containing a set of interconnected relations. MDARM infers interesting, relevant information from a set of relational databases by considering the patterns of connections between individual data elements. The key difficulty is that no piece of information is significant in isolation; rather, it is the combination of many related pieces of data that provide indications of significance. The databases might be both heterogeneous and distributed, but the information in them might still be related to and add value to other pieces of information stored elsewhere. The key challenges of Multiple database association rule mining algorithms are identifying the relevant databases, identifying which relations are related to which others and what are the types of these relationships, and efficient way of mining association rules (in terms of space and time) from these databases.

This paper focuses on mining from multiple databases. In real life, we may have a large collection of data that is stored in several databases in an organized manner. There could be hidden associations among various parts of this data even when it is stored in an organized manner. For example Health service department maintains a
database corresponding to medical history of persons; while person’s purchase information is available in transaction database maintained by supermarkets. An Interesting Question is “is it possible to find knowledge that relates a person’s purchase habits with his health status?” Answer to this question might merge by – mining across multiple databases. This activity of generating association rules between sets of values of attributes needs a scheme that helps in navigating across databases. We call the association rules thus generated inter-database association rules. The activity of mining inter-database association rules is not reported in the literature. This observation motivated us to investigate this important problem. In this paper, we propose a scheme for inter-database association rules.

The traditional way of mining multiple databases is to first integrate the databases [], then apply data mining algorithms. However, it is often hard to integrate heterogeneous databases or to migrate one whole database to another site, because of both efficiency and privacy concerns. Some of the problems encountered with this approach are as follows:

i. Database integration itself is still a problematic area, especially where the source domains differ.

ii. All Databases need to be joined together to produce a single table. However, when multiple tables are joined, the resulting table will increase in size many folds. In large applications, often the join of all related tables cannot be realistically computed because the many-to-many relationship blow up, large relational tables, and the distributed nature of data. Even if the join can be computed, the multifold increase in both size and dimensionality presents a huge overhead to the already expensive frequent itemset mining step: (1) The number of columns will be close to the sum of the number of columns in the individual tables. As the performance of association rule mining is very sensitive to the number of columns the mining on the resulting table can take much longer computation time compared to mining on the individual tables. (2) If the join result is stored on disk, the I/O cost will increase significantly for multiple scanning steps in data mining. (3) For mining frequent itemsets of small sizes, a large portion of the I/O cost is wasted on reading the full records containing irrelevant relationships. (4) another challenge is how to keep track of the many-to-many relationship

iii. If databases are joined and data mining algorithms are applied, the users face the problem of identifying interesting patterns from a large number of discovered rules. In practice, it is too easy to discover a huge number of patterns in a database. However, it is difficult for users to search in all the discovered patterns for useful ones.

iv. The next fundamental problem is efficient algorithms for identifying potentially useful patterns in multi-databases. Thus in multi-database mining we need efficient approaches that can produce good mining results with significant reduction in time and space complexity.

v. An important aspect of multi-database data mining algorithms is that they should scale well to large databases.

A number of challenges must be addressed to provide effective and efficient solutions to the overall problem of inter-database mining. One is how to define the relationship between the tables within the database as well across the databases so that both semantic relationship and existing domain knowledge can be easily exploited, while
avoiding unnecessary joins. Another challenge is that how many such relationships between the tables across the databases must be considered in order to mine all possible inter-database association rules. Since this is a combinatorially complex issue, there is a need to reduce the space and time complexity. In this work, an efficient framework has been developed to generate association rules between sets of values which are subsets of the domains of attributes occurring in relations present in different databases by considering the above issues.

In our Approach, we investigate a mechanism to generate association rules between tuples which are instantiations of Cartesian product of attributes under consideration occurring in relations present in different databases. This process is guided by domain knowledge in the form of semantic network [Findler 79]. The semantic network is used to identify and order the relevant databases. Once the databases are identified, attribute mapping information present in the data dictionary [Elmasri et.al 00] is used to generate the navigation paths. Using these navigation paths, a structure is constructed which semantically partitions the resultant relation. In order to discover frequent tuples, we need to scan this resultant relation only once. Since each frequent table is a consequent of a potential association rule, there is no more processing required for the generation of association rules.

In this paper, we use a tree structure called Pattern Count (PC) tree which can be built using a single database scan for mining frequent itemsets [Ananthanarayana et al. 00]. Later we show that how a collection of PC trees can be used for mining multiple databases.

The major contributions of this paper are:

i. Investigation of multiple navigation paths between the source and destination attributes to capture all the information that is available to come to correct conclusions.

ii. Support for all types of relationships between entities, viz., one-to-one, one-to-many, many-to-one and many-to-many.

iii. Support for mining association rules from multiple source attributes to multiple destination attributes.

2 Problem:

Consider the following situation. We are given a set of source attributes (S₁, S₂, S₃, S₄) and a set of destination attributes (D₁, D₂, D₃, D₄). Our aim is to find counts corresponding to the destination attributes for all input combinations of the source attributes. This means, for a instance of S₁, we have to find the number of occurrences of a given instance of D₁. We shall understand this better with an example.

Consider an employee database with a table containing age and employee name. The age can be generalized into generic categories or age groups. We shall then have names corresponding to various age groups. Similarly, consider a customer database with a table containing customer name and the items he/she purchased. We wish to find the number of televisions purchased by people of age group 3, say. Such scenarios are the motivation for this framework.

3 Outline of the solution:
Figure 1 depicts a graphical representation of our framework. We have, as resources, a number of heterogeneous databases, which can be used to link the source and destination attributes. The relations used to link sources to destinations are internal to the framework and are shown in figure 1 (a) inside the central block. The various links possible between sources and destinations are found and evaluated based on a number of criteria to obtain destination counts. The result of this process is shown in figure 1 (b), where the framework evaluates to direct links. Each of the link has a count which is calculated by appropriate weighted consideration of all the paths connecting the particular source to the particular destination.

**Figure 1:** (a) The framework

**Figure 1:** (b) Direct links

The major steps in the solution are:

*Step 1: Identification of internal databases, relations and instantiation of PC trees:*

In this step, we identify the relevant databases and relations using the semantic network and the data dictionary, trim them to eliminate irrelevant fields, and create one Pattern Count tree (PC Tree) for each relation. PC tree is an efficient intermediate data structure used for this representation.
**Step 2: Calculation of link fractions:**

The framework supports one-to-one, one-to-many and many-to-many relationships between the source and the destination attributes. We traverse through the linked PC trees from the source to the destination and as we do this, we calculate fractions (weights) for each link and use these in determining what part of the final destination count is associated with the particular source attribute being considered – the source from where the traversal began.

**Step 3: Construction of back links:**

On reaching a destination node from a source node in a PC tree traversal, we construct a back link from the destination to the source. The motivation behind construction of these back links is to identify, for each destination attribute instance, the source attribute instance from where we started in order to reach it. If the same destination is reached from multiple sources, it will be evident from the back links of that destination. This will help us associate the destination with the combination of the sources connected by back links. Thus, an analysis of these last level nodes of the PC tree network will reveal count values for destination attribute instances corresponding to combination of different source attribute instances.

**Step 4: Consolidation of counts:**

Finally, we consolidate the counts for the many-to-many cases based on a probabilistic model. The counts, in all other cases, do not require this step.

We shall detail these steps in the following sections.

4 **Identification of internal databases, relations and instantiation of PC trees:**

Here, we shall look into the pattern count tree and how it is used to link databases and relations.

4.1 **Pattern Count tree:**

Pattern Count (PC) Tree is a data structure which is used to store all the patterns occurring in the tuples of a transaction database in a compact way where count filed is associated with each item in every pattern. In the PC tree, all the patterns in the transaction database are stored. The count filed is responsible for a compact realization of the database. The structure of a node in the PC-tree is given in Figure 3.

4.1.1 **Structure of a PC tree:**

Each node of the tree consists of the following structure:

- **Item-name:** In the node, the item-name field specifies which item the node represents.
- **Count:** The count field represents the number of transactions represented by a portion of the path reaching this node.
- **Child pointer:** The child-pointer represents the pointer to the following pattern.
- **Sibling pointer:** The sibling pointer points to the node which indicates the subsequent other patterns from the node under consideration.
Preprocessing Framework for Association Rule Mining

**Figure 2:** Structure of a Node in the PC-tree

### 4.1.2 Construction of the PC-Tree:

PC-Tree construction requires a single scan of the database. We take each transaction from the database and put it as a new branch of the PC-Tree if any sub pattern, which is a prefix of the transaction, does not exist in the PC-Tree; else put it into an existing branch $e_b$ by incrementing the corresponding count field value of the nodes in the PC-Tree. We put the remaining sub patterns, if any, of the transaction by appending additional nodes with count field value equal to 1 to the path $e_b$. We preprocess each transaction to put items in a lexicographical order. We give below an example to explain the construction of a PC-tree. Consider the transaction database shown in Figure 4.a The PC-tree for the transaction database, $D$, is shown in Figure 4.b.

![PC tree construction](image)

**Figure 3:** PC tree construction

### 4.1.3 Mining Frequent itemsets:

For generating frequent itemsets, construct the frequent 1-itemset list ($L$) in the descending order of their frequencies in parallel with construction of PC tree. For each entry $a_i$ in $L$, link all the nodes of the PC-tree (through node-links) which are having $a_i$ as their item name. To find all possible frequent patterns that contain $a_i$, follow $a_i$’s node-links and obtain the patterns where $a_i$ is the last element in it. If the

<table>
<thead>
<tr>
<th>Transaction ID</th>
<th>IDs of Items purchased</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>19,40,510,527</td>
</tr>
<tr>
<td>T2</td>
<td>19,40,179</td>
</tr>
<tr>
<td>T3</td>
<td>19,40,125,510</td>
</tr>
<tr>
<td>T4</td>
<td>527,740</td>
</tr>
<tr>
<td>T5</td>
<td>527,740,795</td>
</tr>
<tr>
<td>T6</td>
<td>19</td>
</tr>
</tbody>
</table>
count value of \( a_i \) in all patterns together is greater than or equal to the user defined minimum support value \( (\sigma) \), then pick the sub pattern (which is the pattern excluding \( a_i \)) from each pattern with the count value set to the count value of \( a_i \) in that pattern. Find the intersection of all sub patterns and add the count value of corresponding item in the sub pattern. If each item in the resultant sub pattern has its count value greater than or equal to \( \sigma \), then the resultant pattern is a frequent \( x \)-itemset, where \( x \) is the number of items in the sub pattern.

### 4.1.4 Properties of PC tree:

i. PC-tree is a complete representation \( (T) \) of the transaction database. A representation \( R \) is complete with respect to a transaction database \( DB \), if every pattern in each transaction in \( DB \), is present in \( T \).

ii. PC-tree is a compact representation of the transaction database. Let \( D \) be the database of size \( D_s \). Then, a complete representation \( T \) of \( D \) is compact if \( T_s < D_s \), where \( T_s \) is the size of \( T \).

### 4.1.5 Advantages of PC tree:

i. Uses only one scan of the database for construction.

ii. Compact and Complete representation of the tuples in the database

iii. It can be used for incremental mining.

### 4.2 Identification of relevant databases:

It consists of locating and identifying databases that are relevant, i.e. how to determine which databases are relevant and where they are located. Database selection is an application-dependent technique [Liu-Lu-Yao 2001]. In our approach, a semantic network is used as knowledge in the process of identification of relevant databases in the required order.

**Algorithm: Database Identification (DI)**

| Input: | P, Q and the semantic network S |
| Output: | Concepts \( C_1, C_2, ..., C_n \) linking P and Q in the semantic network. |
| Steps: | i. Get a node (concept) \( C_1 \), from \( S \), where \((C_1, P) \in E \). |
| | ii. Get a node (concept) \( C_n \), from \( S \), where \((C_n, Q) \in E \). |
| | iii. Output all possible paths in \( S \) each of which connects \( C_1 \) to \( C_n \). |
| | iv. For each path, execute components B to C in IDARM. |
| | v. Merge the frequent tuples generated in the semantic partitions due to each output |
| | vi. of Concept identification pertaining to a generalized value of a characteristic |
| | vii. attribute which maps to the characteristic concept P. |

*Example:* Let us consider that we want to find the associations between two characteristic concepts *Salary* and *Goods-Purchased*. Let the given semantic network as shown in Figure 1 and let generalization tree for the characteristic concept *Salary*
be shown in Figure 2. Using these inputs, the algorithm DI finds the primary concepts which connect Salary and Goods-Purchased. They are EMPLOYEE, PERSON, and CUSTOMER. Therefore, as in any effective knowledge discovery process, the first important step in mining multiple databases is indeed to select those databases that are relevant to a specific mining task.

4.3 Identification of relevant relations:

After knowing the databases containing the required source and destination attributes, the next step is to identify the correct relations containing these attributes. Using the data dictionary, the relevant relations (origin and target relations) are identified from the above identified databases.

The next step is to identify other relations such that they link origin and target relations. These relations may be present in different databases, possibly different from those containing the origin and target relations. This pre-processing step is especially critical, because it allows records across databases to be linked. Thus specifying beforehand where joins semantically make sense by means of links between nodes (relations) allows the entire multi-database system to act as if it were just one large database. Links are abstract connections between relations that refer to the co-occurrence of the same value in fields of records of two different relations.

\[
\begin{array}{|c|c|}
\hline
\text{Input:} & \text{Concepts } C_1, C_2, \ldots, C_n. \\
\text{Output:} & \text{Global Relation graph corresponding to databases related to } C_1, C_2, \ldots, C_n. \\
\text{Steps:} & \text{For each } C_i, \text{ find set } D_i \text{ of relevant databases using data dictionary. Find a} \\
& \text{database, call it } D_1 \in D_1, \text{ such that one of it’s relations, call it } R_1^1, \text{ has an} \\
& \text{attribute } P_1^1 \text{ mapping to the characteristic concept } P \text{ and call it } X. \text{ Similarly,} \\
& \text{find a database, call it } D_L \in D_n, \text{ such that one of it’s relations, call it } R_L^L, \text{ has attribute} \\
& Q_L^L \text{ mapping to the characteristic concept } Q \text{ and call it } Y. \\
\end{array}
\]

Example: The algorithm DI finds the database corresponding to the primary concepts EMPLOYEE, PERSON, and CUSTOMER. Let the database identified be EMP and CUSTOMER_TRANSACTION (the PERSON primary concept is shared by these two databases). Then RI algorithm next identifies the relation(s) in such a way that first relation (Call this relation as Origin Relation) in the EMP database should have the attribute corresponding to the characteristic concept Salary; let it be Sal and the last relation (call it as Target Relation) in the CUSTOMER_TRANSACTION database should have the attribute corresponding to the characteristic concept Goods_Purchased; let it be Purchased_Goods.

There are two ways of relating the instances between relations that we want to discover in this step. They are as follows:

i. Relating the instances in the same database (Intra-Database Links): To relate relations from the same database, we can take advantage of predefined relationships in the existing data definitions. Relationships are defined as Primary Key – Foreign Key references. The foreign key can be used to cross-reference tables within a database. In the relational model, different joins have different
ii. Relating instances from different databases (Inter-database Links): To mine information across heterogeneous databases, one must detect inter-database links, which are links between matched attributes and can serve as bridges for mining. No predefined relationships exist between instances of two different databases. Semantically equivalent data elements or Common Concepts defined in these databases have to be identified using data dictionary. These attributes are used to establish relationships between the databases. Determination of common set of attributes (CoAs) between the databases is based on attribute equivalence property discussed in schema integration methodology. We assume data dictionary has all CoAs.

Thus, this phase results in establishing links within the database using foreign key relationships and across the database using the CoA relationships. Each edge describes a relationship between two nodes, and can be of type one-to-one, one-to-many, many-to-one or many-to-many.

4.4 Construction of local and global graphs:

Local graphs are graphs that indicate intra-database relationships. In a local graph, each identified relation within the particular database is represented by a node and two nodes are connected by an edge if and only if the corresponding relations are related via a foreign key relationship.

The global graph represents the entire picture obtained by linking the relations across databases. Thus, local graphs have to be linked via inter-database links to obtain the global graph. We identify the nodes in the global graph that correspond to the origin and target relations and find out navigation paths between these nodes.

Algorithm: CreateLocalGraph (D_x)

<table>
<thead>
<tr>
<th>Input:</th>
<th>Database D_x.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output:</td>
<td>Local Relation graph corresponding to the input database.</td>
</tr>
</tbody>
</table>
| Steps:          | i. Let G_z be the local graph for D_z.    
|                 | ii. Create a node R^z for each relation in D_z. |
|                 | iii. For each pair (R^m_z, R^n_z), if there is k-f or f-k relationship between relations R^m_z and R^n_z then link nodes R^m_z and R^n_z. |
|                 | iv. Return (G_z). |

Algorithm: CreateGlobalGraph (G_1, G_2, …., G_L)

<table>
<thead>
<tr>
<th>Input:</th>
<th>Local graphs G_1, G_2, …., G_L.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output:</td>
<td>Global Relation graph.</td>
</tr>
<tr>
<td>Steps:</td>
<td>i. Let G_global be the global graph.</td>
</tr>
<tr>
<td></td>
<td>ii. For each pair (G_m, G_n) 1 ≤ m ≤ L, 1 ≤ n ≤ L, m ≠ n.</td>
</tr>
<tr>
<td></td>
<td>iii. Identify all possible Common Attributes (CoA) between D_m and D_n. (Using Data Dictionary)</td>
</tr>
<tr>
<td></td>
<td>iv. For every CoA, Link the relations involving common attributes</td>
</tr>
</tbody>
</table>
between $D_m$ and $D_n$.

v. Return ($G_{\text{global}}$).

Figure 2 shows the construction of the Global Graph. Edges within the local graph represent f-k relationships and edges between local graphs are obtained using CoAs. After identifying the relations $R_1$, $R_2$, ..., $R_m$ and possible links between them we can model the result by a graph $G(V,E)$ where $V$ is the set of relations (nodes) and $E$ is the set of edges (joining attributes – CAs and CoAs) between the relations in $V$. A link is present between two attributes A and B, if and only if they are semantically equivalent.

Solid lines indicate the links within a database, i.e., the links from the individual local graphs. Dashed lines are those constructed across databases using CoAs. Local graphs are interconnected using these links to construct the global graph.

![Figure 2: Construction of the Global Graph](image)

**Figure 2:** Construction of the Global Graph

4.5 Selection of navigation paths:

We consider multiple navigation paths between every pair of source and destination attributes. This is because the instances of the source attributes mapped to the instances of the destination attributes by these paths may differ and hence, may require consolidation in the end to produce more accurate results.

**Algorithm: Identification of Navigation paths**

<table>
<thead>
<tr>
<th>Input</th>
<th>Global graph.</th>
</tr>
</thead>
</table>
Output: Navigation paths.

Steps:

i. Start the traversal from the origin node \( R_i \) in \( G_i \) which contains the source Characteristic Concept (CC). Also find the nodes which can be joined with \( R_i \) either using direct links (f-f relationships) or using indirect links (through intermediate relations).

ii. Repeat for all local graphs \( G_i \), which link to the previous sink nodes using CoA.
   a. Determine the source nodes \( R'_i \) in \( G_i \) which contain the CoA.
   b. For each source node \( R'_i \), traverse the local graph \( G_i \) to reach its corresponding sink nodes using direct or indirect joins until the destination node is reached.

Consider two indirect join paths \( P_i \) and \( P_j \) connecting two relations \( R_x \) and \( R_y \). Let \( X_i \) be the set of values of attribute \( X \) in \( R_x \) and \( Y_i \) be the set of values of attribute \( Y \) in \( R_y \) which are mapped in path \( P_i \). Let \( X_j \) be the set of values of attribute \( X \) in \( R_x \) and \( Y_j \) be the set of values of attribute \( Y \) in \( R_y \) which are mapped in path \( P_j \). Assuming that \( P_i \) and \( P_j \) differ in at least one intermediate relation, we may find the following conditions to be true:

i. Total intersection: If \( X_i \) is a subset of \( X_j \) and \( Y_i \) is a subset of \( Y_j \), then only \( P_j \) can be considered for mining. The other way round selects \( P_i \).

ii. Partial intersection: If \( X_i - X_j \neq \emptyset \) or \( X_j - X_i \neq \emptyset \), then it indicates that one of the two paths map some values of attribute \( X \), which are not mapped by other path. Similarly \( Y_i - Y_j \neq \emptyset \) or \( Y_j - Y_i \neq \emptyset \) also indicates that one of the two paths map some values of attribute \( Y \), which are not mapped by other path. Here, relative weights of both paths have to be taken into consideration.

iii. Total disjoint: \( X_i \cap X_j = \emptyset \) or \( Y_i \cap Y_j = \emptyset \) indicates that the two paths map totally disjoint values of attributes \( X \) or \( Y \). In this case, both paths have to be considered.
Figure 5: Overlapping in relations

Example:
Let X be the source CC and Y be the destination CC.
\( X_1 \subset X \), and \( Y_1 \subset Y \) are the set of values that get mapped along the path \( P_1 \).
Similarly \( X_2 \) and \( Y_2 \) get mapped along \( P_2 \).
Preprocessing Framework for Association Rule Mining

Figure 6: Illustration of Partial Intersection

In the above join graph there are two join paths

\( P_1: R_1 \Join R_1 \Join R_2 \)

\( P_2: R_1 \Join R_1 \Join R_2 \)

\[ X_1 = \{1,2\} \quad X_2 = \{2,3,4\} \]

\[ Y_1 = \{2,8\} \quad Y_2 = \{7,8,11\} \]

There are two approaches in selecting the join paths between two relations:

i. *Selecting all possible paths:* Selecting all possible paths is too expensive because infinitely many paths exist if we allow loops in graph. Even if we do not allow loops, there may be exponential number of paths between two nodes, if the number of nodes in the graph and connectivity of nodes is high.

ii. *Selecting most probable paths:* Considering the probable paths that would cover most of the associations, which would otherwise be generated by all possible paths, is sufficient. In order to limit the number of paths that are to be considered, we associate a weight factor to each path. The weight factor indicates the semantic strength of the join path. The weight is determined based on the following properties of the path:

a. *Selectivity:* Selectivity of a join path often indicates the strength of the relationship. In general, the selectivity of a join path is the average number of tuples in \( R_i \) that can be joined with tuples in \( R_j \). The selectivity and semantic strength of a join path are highly correlated. A high selectivity indicates that most of the tuples of \( R_i \) are covered. For a join path \( P = R_1 \Join \ldots \Join R_l \), its selectivity \( \rho (P) \) is the average number of tuples in \( R_i \) that are joinable to a...
tuple in $R_1$ via $P$, $\rho(P)$ is the average selectivity for all tuples in $R_1$.

b. **Path length:** The distance between relations indicates the strength of relationship. We are assuming that affinity between two attributes decreases if the distance is more. Here distance is measured in terms of number of relations involved in join path.

c. **Number of databases:** The inter-database links through CoAs are weaker links as compared to intra-database links via foreign key relationships. Thus, the more the number of databases encountered along the join path, the weaker is the path.

Selectivity of join paths can be estimated by many previous approaches \[7,12\]. A simple approach for estimating selectivity of a join path $P = R_1 \bowtie R_2 \bowtie \ldots \bowtie R_l$ is to use the product of selectivity of every join in the path, i.e., $\rho(P) = \prod_{(1 \leq i \leq l-1)} \rho(R_i \bowtie R_{i+1})$. We assume that join paths are independent.

\[ \rho(R_i \bowtie R_{i+1}) \] is selectivity estimation based on partial participation of relations which is given by the following formula

\[
n(C_i \bowtie D_j) = n(C_i) \times J(C_i) \times \frac{n(D_j)}{dv(D_j,A_j)}
\]

Where $J(C_i)$ is the join participation ratio defined as the ratio of number of distinct join column values of $C_i$ participating in unconditional join to the total number of distinct join column values. $n(D_j)$ is th cardinality of $D_j$ and $dv(D_j,A_j)$ is the number of distinct values of $D_j.A_j$.

The weight information is added into the global graph and $k$ – best paths are found using standard graph algorithms, where $k$ is the user-defined threshold.

**Algorithm: CreateWeightedLocalGraph ($D_z$)**

<table>
<thead>
<tr>
<th>Input:</th>
<th>Database $D_z$.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output:</td>
<td>Weighted Local Relation graph corresponding to the input database.</td>
</tr>
<tr>
<td>Steps:</td>
<td></td>
</tr>
<tr>
<td>i.</td>
<td>Let $G_z$ be the local graph for $D_z$.</td>
</tr>
<tr>
<td>ii.</td>
<td>Create a node $R_i^z$ for each relation in $D_z$.</td>
</tr>
<tr>
<td>iii.</td>
<td>For each pair $(R_m^z, R_n^z)$, if there is k-f or f-k relationship between relations $R_m^z$ and $R_n^z$ then link nodes $R_m^z$ and $R_n^z$.</td>
</tr>
<tr>
<td>iv.</td>
<td>Assign Link weight $W_i(R_m^z, R_n^z) =</td>
</tr>
<tr>
<td>v.</td>
<td>Return ($G_z$).</td>
</tr>
</tbody>
</table>

**Algorithm: CreateWeightedGlobalGraph ($G_1,G_2,\ldots,G_L$)**

<table>
<thead>
<tr>
<th>Input:</th>
<th>Local graphs $G_1, G_2,\ldots,G_n$.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output:</td>
<td>Weighted Global Relation graph.</td>
</tr>
<tr>
<td>Steps:</td>
<td></td>
</tr>
<tr>
<td>i.</td>
<td>Let $G_{global}$ be the global graph.</td>
</tr>
<tr>
<td>ii.</td>
<td>For each pair $(G_m, G_n) \ 1 \leq m \leq L, \ 1 \leq n \leq L, \ m \neq n$.</td>
</tr>
<tr>
<td>iii.</td>
<td>Identify all possible Common Attributes (CoA) between $D_m$ and $D_n$.</td>
</tr>
</tbody>
</table>
(Using Data Dictionary)
iv. For every CoA, Link the relations involving common attributes between $D_m$ and $D_n$.

v. Assign Link weight $W_i(R^+_m, R^+_n) = \mid \log \frac{1}{\rho (R^+_m \bowtie R^+_n)} \mid + \text{constant.}$

vi. Return $(G_{\text{global}})$.

4.6 Trimming:

While linking relations, the only useful attributes are the source attributes in the origin relations, destination attributes in the target relations and all the attributes involved in the intra-database (foreign keys) and inter-database (common attributes) joins. Thus, every relation other than the origin and target relations has only two sets of attributes – one set for linking to previous relation and one set to link to the next relation in the navigation path. An origin relation has the source attribute and a link to next relation, whereas a target relation has a link from the previous relation and the destination attribute. Thus, every relation has only two sets of attributes. Hence, we trim the relevant relations to retain just these two sets.

Algorithm: Trimming

<table>
<thead>
<tr>
<th>Input:</th>
<th>$i^i R$, $i^2 R$, ..., $i^m R$ of path $P_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output:</td>
<td>$i^1 \mathcal{R}$, $i^2 \mathcal{R}$, ..., $i^m \mathcal{R}$</td>
</tr>
<tr>
<td>Steps:</td>
<td>For each $i^j R$, construct trimmed relation $i^j \mathcal{R}$ as follows:</td>
</tr>
<tr>
<td>i.</td>
<td>If ($j = 1$) then</td>
</tr>
<tr>
<td>ii.</td>
<td>$i^j \mathcal{R}$ has $&lt;X, i^j C, \text{Count}&gt;$ attributes.</td>
</tr>
<tr>
<td>iii.</td>
<td>Where $i^j C$ is</td>
</tr>
<tr>
<td>a.</td>
<td>a key if $i^j \mathcal{R}$ is obtained from the same database as $i^1 \mathcal{R}$, in which case navigation is through a key.</td>
</tr>
<tr>
<td>b.</td>
<td>a common set of attributes if $i^j \mathcal{R}$ is obtained from a different database than that of $i^1 \mathcal{R}$, in which case navigation is through a common set of attributes.</td>
</tr>
<tr>
<td>If ($j = m$) then</td>
<td>$i^j \mathcal{R}$ has $&lt;Y, i^{j-1} C, \text{Count}&gt;$ attributes.</td>
</tr>
<tr>
<td>Else</td>
<td>$i^j \mathcal{R}$ has $&lt;i^{j-1} C, i^{j+1} C, \text{Count}&gt;$ attributes.</td>
</tr>
<tr>
<td>Where each $i^{j-1} C$ and $i^{j+1} C$ is either a key or a common set of attributes.</td>
<td></td>
</tr>
</tbody>
</table>

**NOTE:** In the above representation, the attribute Count has a value corresponding to the number of times a tuple, based on other attributes (i.e., excluding Count Field) in relation $i^j \mathcal{R}$ occurred.

For $i^1 \mathcal{R}$, 

i. Generalize the values corresponding to attribute X using generalization
tree $T_A$ and generalization level value $l$.

ii. Update the count field value of the tuple by adding its value with value in Count Field of other tuples if they have same values for other attributes (i.e., excluding Count Field) in relation $\mathcal{R}$. Remove the latter tuples.

iii. Let $CA\_COUNT(f)$, $1 \leq f \leq \rho$ be the total count for each generalized value of $X$ in $\mathcal{R}$. Note that if $CA\_COUNT(f) / |\mathcal{R}| < \sigma$ for $1 \leq f \leq \rho$, then no association rules exist for generalization value $f$ of $X$. So in later steps, processing w.r.t generalization value $f$ is not considered.

### 4.7 Construction of PC trees:

We construct one PC tree per relation in the final global graph. These PC trees will be of two levels, each corresponding to one set of attributes in the trimmed relation from which the tree was constructed. These PC trees are instantiated and stored in the memory.

**Algorithm: PC tree instantiation**

<table>
<thead>
<tr>
<th>Input:</th>
<th>$\mathcal{R}_1, \mathcal{R}_2, \ldots, \mathcal{R}_m$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output:</td>
<td>PC trees $\mathcal{PC}_1, \mathcal{PC}_2, \ldots, \mathcal{PC}_m$</td>
</tr>
<tr>
<td>Steps:</td>
<td></td>
</tr>
<tr>
<td>i. Each $\mathcal{R}_j$ has two sets of attributes. If ($j = 1$), then the first set corresponds to characteristic attributes and the second set corresponds to common set of attributes. If ($j = m$), then the first set corresponds to common set of attributes and the second set corresponds to characteristic attributes. If ($1 &lt; j &lt; m$), then both sets correspond to characteristic attributes.</td>
<td></td>
</tr>
<tr>
<td>ii. For each relation, $\mathcal{R}_j$, where $1 \leq j \leq m$, use the values of first set and second set of attributes in that order and construct PC-trees, $\mathcal{PC}_1, \mathcal{PC}_2, \ldots, \mathcal{PC}_m$ as discussed in section 4.1.</td>
<td></td>
</tr>
</tbody>
</table>

### 4.8 Linking of PC trees:

For each edge in the global graph, we link PC trees corresponding to the end-nodes of that edge. The values corresponding to the second set of attributes in each relation are used to link the PC trees. The trees are linked by connecting second level nodes of one tree with first level nodes of next tree.
Preprocessing Framework for Association Rule Mining

Figure 7: Linking PC trees

Algorithm: PC tree linking

| Input:  | \( \{PC_1, PC_2, \ldots, PC_i\} \) |
| Output: | Linked PC trees of path \( P_i \) |
| Steps:  | Use the values corresponding to the second set of attributes in each relation, \( \{\mathcal{R}_1, \mathcal{R}_2, \ldots, \mathcal{R}_m\} \), to link the PC-trees. Link PC-trees connecting second level nodes of each tree with first level nodes of next tree. The instances of Relations in path \( P_i \) is shown in figure 10 and the corresponding linking of PC-Trees for the path \( P_i \) is shown in figure 11. |

Example:

![Diagram of PC trees linking](image)

Figure 8 (a): Instances of Relations in path \( P_i \)
5 Calculation of link fractions:

As we traverse the PC trees from source attribute to destination attribute, whenever we reach a node $N$, we find out the set of nodes $S_N$ from which $N$ can be directly reached. Then, we compute the relative weight ratio of the node $N_{prev}$ (from $S_N$) from which we reached $N$ in the set $S_N$. This ratio is computed by dividing the count value of $N_{prev}$ by the sum of count values of all nodes in $S_N$. The link fraction associated with the link from $N_{prev}$ and $N$ is assigned to this ratio. Such link fractions are found for each edge traversed from source to destination.

The count associated with the nodes corresponding to the instances of the destination attribute when reached from a particular instance of the source attribute is calculated by multiplying the resident count on that node by all the link fractions encountered while we traverse from the source attribute instance to these nodes.

The understanding of these link fractions and why they are justified is based on the following analysis.
5.1 Many-to-one relationships:

In real scenarios, data may rarely be unique, especially join attribute data. Hence, while linking two relations, we encounter one-to-one, many-to-one, one-to-many and many-to-many relationships. One-to-one relationships are a subset of many-to-one relationships. Many-to-one relationships are handled by our framework in the following way.

5.1.1 Scenario:

Our employee relation is cropped to contain the fields that matter, viz, the Age Group attribute, which is our attribute of origin and the Name attribute which is the common attribute with the customer relation. Similarly, the customer relation is also cropped to include only the common attribute and the destination attribute – Item Purchased.

![Figure 9: PC Tree representation of the relations](image-url)

We explore the case of a situation occurring between the two relations that belong to different databases, in which a many-to-one relationship exists. This scenario is illustrated by the example we have taken up. John is a very common name in organizations and hence, a situation may arise wherein there are 9 Johns of different ages as shown. Understand that they have different IDs and hence, are essentially different people. However, ID is not considered in the usual approach because it is neither the original nor destination attribute nor a common attribute.

Consider the customer relation recording that one John bought 5 items and another John bought 2 items. This relation distinguishes between these Johns on the basis of the ID stored in it. Again, the usual method does not consider this ID for the same aforementioned reasons. The PC Tree drawn would connect all the Johns from the employee relation to the John in the customer. Now, while performing the item count, the entire count for the John in the customer relation is considered although this count
has contributions from all the age groups. This results in wrong results especially if we are investigating the count for a particular age group.

We propose a probabilistic model of calculating the expected number of items bought by a John of age group 1 or 2 or 3 based on the probabilities of each of these Johns to correspond to the Johns in the customer relation. We split up the count of items 8 among the age groups as follows:

Total number of Johns in all age groups: 9  
Number of Johns in age group 1: 3  
Number of Johns in age group 2: 4  
Number of Johns in age group 3: 2

Probability that a given John out of these 9 belongs to a particular age group is:

Group 1: \( \frac{3}{9} \)
Group 2: \( \frac{4}{9} \)
Group 3: \( \frac{2}{9} \)

The item count of 8 is split as follows:

Age group 1: \( \frac{3}{9} \cdot 8 = \frac{24}{9} \)
Age group 2: \( \frac{4}{9} \cdot 8 = \frac{32}{9} \)
Age group 3: \( \frac{2}{9} \cdot 8 = \frac{16}{9} \)

Thus, on an average, based on probability, we consider that the Johns belonging to age groups 1, 2 and 3 have bought \( \frac{24}{9} \), \( \frac{32}{9} \) and \( \frac{16}{9} \) items respectively. These numbers just give an indication and certainly are not the actual accurate values which cannot be predicted with the information available.

This argument seems very intuitive. The following section will demonstrate it based on probability theory.

### 5.1.2 Statement:

Consider the following diagram wherein there is a many-to-one relation between categories 1 to \( n \) indicated in PC Tree A and the item in PC Tree B. The number of items purchased by all these Johns is \( Y \), in total. Let the number of Johns of category \( i \) be \( x_i \) and their sum be \( X \).
The number of items purchased by Johns of category $i$ is:

$$\mathbb{N} = \frac{x_i}{X} Y$$

5.1.3 Proof:

Consider that $Y$ items were purchased as a collection of $m$ transactions. Let the number of items purchased in the $j^\text{th}$ transaction be $y_j$. Thus, the only allowed values for the number of items purchased by a person are 0, $y_1, y_2, \ldots, y_m$, sums of $y_i$'s taken two at a time, three at a time, till all taken at a time.

Let the probability that a John from category $i$ performs transactions with a total of $j$ $y_i$'s taken together be $G(i, j)$. The maximum number of ways of permuting $X$ Johns in $m$ transactions is $P(x, m)$. Out of these, the favourable conditions are when the $x_i$ Johns from category $i$ perform the $j$ fundamental transactions and the remaining Johns perform the other transactions. Thus, the value of $G(i, j)$ becomes:

$$G(i, j) = \frac{P(x, j) \cdot P(X - x_i, m - j)}{P(X, m)}$$

Now, the summation of number of items in a set of transactions in which $j$ fundamental transactions are taken at a time is $C(m - 1, j - 1) \times Y$. But, if $j$ is 0, then the summation is also 0. Thus, the expected number of items purchased by Johns from category $i$ is obtained by taking the expectation of this distribution.
\[ E = \sum_{j=1}^{m} \{ G(i, j) \cdot C(m-1, j-1) \cdot Y \} = \sum_{j=1}^{m} \left\{ \frac{P(x_i, j) \cdot P(X - x_i, m - j)}{P(X, m)} \cdot C(m-1, j-1) \cdot Y \right\} \]

Converting the permutations to combinations, we obtain:

\[ E = Y \cdot \sum_{j=1}^{m} \left\{ \frac{C(x_i, j) \cdot C(X - x_i, m - j)}{C(X, m)} \cdot j \right\} \]

Expanding,

\[ E = Y \cdot \sum_{j=1}^{m} \left\{ \frac{x_i! \cdot (X - x_i)! \cdot (m-1)! \cdot (X - m)! \cdot j}{j! \cdot (x_i - j)! \cdot (m-j)! \cdot (X - x_i - m + j)! \cdot m!} \right\} \]

Simple manipulation yields

\[ E = \frac{Y}{C(X, x_i)} \cdot \sum_{j=1}^{m} \{ C(m-1, j-1) \cdot C(X-m, x_i - j) \} \]

The summation in the numerator can be directly computed from the expansion of the right hand side of the following equation and equating the binomial coefficients of the \((x_i - 1)^{th}\) power of \(z\) on both sides.

\[ (1 + z)^{x_i} = (1 + z)^{X-m} \cdot (1 + z)^{m-1} \]

Thus,

\[ E = \frac{Y \cdot C(X-1, x_i-1)}{C(X, x_i)} = \frac{Y \cdot x_i}{X} \]

5.1.4 Illustration:

The customer relation makes it clear that one John bought 5 items and another bought 3. Hence, the only possible values for number of items bought by Johns from a particular age group are 0, 3, 5 and 8. We calculate the probability of a particular group of Johns buying each one of the above number of items and then compute the expectation of items. We arrive at the exact figures as obtained in the previous section.

Number of possible permutations of Johns that can fit in into the customer relation Johns
\(= (\text{No. of ways of choosing the first John}) \times (\text{No. of ways of choosing the second John})\)
\(= 9 \times 8\)
\(= 72.\)

Thus, the denominator in all the probability values shall be 72.

Let \(H(X, Y)\) be the probability that a John from age group X has an item count of Y. Thus, the allowed values for Y are 0, 3, 5 and 8. The calculation of \(H(X, Y)\) for age group 1 is shown below:
To calculate $H(1, 0)$, we consider that when age group 1 buys 0 items, age groups 2 and 3 buy all 8. Hence, we fit the 6 Johns of groups 2 and 3 into the two slots in the customer relation. This can be done as follows:

(No. of ways of selecting first John) $\times$ (No. of ways of selecting second John)
$= 6 \times 5$
$= 30$ ways.

Thus, $H(1, 0) = 30 / 72$.

Similarly, we obtain
$H(1, 3) = 18 / 72$
$H(1, 5) = 18 / 72$
$H(1, 8) = 6 / 32$

Expectation $= 0 \times (30 / 72) + 3 \times (18 / 72) + 5 \times (18 / 72) + 8 \times (6 / 72) = 24 / 9$.

All other values are obtained similarly.

5.2 Multiple source attributes:

The previous sections explained how the count of the destination instance is split by link fractions in the case of a many-to-one scenario. But so far, we have always been trying to investigate the counts associated with the association rule $S \rightarrow D$, i.e., one source attribute to one destination attribute. However, our framework can handle multiple source attributes to multiple destination attributes. This section will show how multiple source attributes are handled.

5.2.1 Scenario:

Consider this scenario: We have classified people according to age into groups 1, 2 and 3; and according to place of work into Microsoft, Google and Samsung. The same closed domain of people is considered for classification in both the taxonomies. Now, the people from this closed domain do some purchasing at retail stores and buy items like TV, fridge and oven. From the store records, we know the number of each of the items sold and we also know the names of the customers who purchased them. We are now interested in finding out the number of TVs purchased by people from age group 3 and Microsoft, or any instance like this one.
The calculation becomes as trivial as doing a count through the PC tree, when there is a one-to-one or a one-to-many relationship between the age tree and the items tree, as also the company tree and the items tree. In that case, there is just one person John in the domain who is in age group 3 and company Microsoft and has purchased 8 TVs. However, if we have a many-to-one relationship, then we have, say, 9 Johns and they purchase 14 TVs. How do we know how many Johns are from age group 3 and from Microsoft? Consequently, how we know how many TVs were purchased by people from age group 3 and Microsoft?

5.2.2 Problem:

This is a problem of finding the cardinality of the intersection of two sets. The cardinalities of the two sets respectively are known to us, but the cardinality of the union is not known. Hence, a precise value of cardinality of intersection cannot be arrived at. Therefore, we use a probabilistic approach and find out the most expected value of the cardinality of the intersection.

5.2.3 Analysis and solution:

Assume that there are 3 Johns from age group 3 and 4 Johns from Microsoft. There are totally 9 Johns in our closed domain. This is our domain size. The first set is the set of Johns from age group 3 and the second set is set of Johns from Microsoft. The number of Johns in the intersection will be the number of Johns who are from age group 3 and Microsoft. However, these sets can intersect to variable extents. Hence, we have to come up with the most expected extent of intersection. In the following
analysis, we shall refer to this extent of intersection as the *overlap*. The logic proceeds 
as follows.

A configuration is defined as a representation of the number of elements (in this case, 
Johns) in the two individual sets, their intersection and their union. For a given 
overlap, by varying the other parameters, we can get various configurations. We can 
find out the expected value of the overlap by multiplying the probability of each 
configuration occurring and the overlap in each and summing up the products. The 
following mathematics precisely puts these ideas into formulae and arrives at a 
solution to the expected overlap.

Let the following variables be defined:

\[ n = \text{the domain size} \]
\[ n_1 = \text{the number of elements in the first set} \]
\[ n_2 = \text{the number of elements in the second set} \]

Consider the case of an overlap of \( k \) elements. Combination is denoted by \( C(n, r) \) for 
more legibility.

We have to select \( k \) elements from the domain for the overlap. This can be done in \( C(n, k) \) ways. Then, we have to arrange the remaining elements of both sets such that 
they do not produce any additional overlap. This can be done in \( C(n - k, n_1 - K) \times C(n - k - n_1 + k, n_2 - k) \) ways.

Hence, the total number of ways in which an overlap of \( K \) can be produced is:

\[ C(n, k) \times C(n - k, n_1 - K) \times C(n - n_1, n_2 - k) \]

The total number of configurations possible is \( C(n, n_1) \times C(n, n_2) \).

Hence, the probability of a configuration with an overlap of exactly \( K \) is:

\[
p_k = \frac{C(n, k) \times C(n - k, n_1 - k) \times C(n - n_1, n_2 - k)}{C(n, n_1) \times C(n, n_2)} 
\]

The minimum overlap is \( k_{\text{min}} \) where \( k_{\text{min}} = n_1 + n_2 - n \) if \( n_1 + n_2 > n \); 0 otherwise. The 
maximum overlap is \( k_{\text{max}} \) where \( k_{\text{max}} = \text{minimum } (n_1, n_2) \).

Thus, the expected overlap, denoted by \( x \) is given by:

\[
x = \sum_{k = k_{\text{min}}}^{k_{\text{max}}} \frac{C(n, k) \times C(n - k, n_1 - k) \times C(n - n_1, n_2 - k)}{C(n, n_1) \times C(n, n_2)} \times k \]

We then calculate our expected fraction of the destination count as in the previous 
section by

\[
E = \frac{Y \cdot x}{X}
\]

where \( Y \) is the total count of destination node.
5.2.4 Illustrations:

Example 1: \( n = 9; n_1 = 4; n_2 = 3 \).

<table>
<thead>
<tr>
<th>Overlap (K)</th>
<th>Calculation of numerator components</th>
<th>Contribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( C(9, 0) \times C(9, 4) \times C(5, 3) \times 0 )</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>( C(9, 1) \times C(8, 3) \times C(5, 2) \times 1 )</td>
<td>5040</td>
</tr>
<tr>
<td>2</td>
<td>( C(9, 2) \times C(7, 2) \times C(5, 1) \times 2 )</td>
<td>7560</td>
</tr>
<tr>
<td>3</td>
<td>( C(9, 3) \times C(6, 1) \times C(5, 0) \times 3 )</td>
<td>1512</td>
</tr>
</tbody>
</table>

**Total Numerator:** 14112

Denominator = \( C(9, 3) \times C(9, 4) = 10584 \)
Hence, \( N = 1.333 \)

Example 2: \( n = 12; n_1 = 4; n_2 = 3 \).

<table>
<thead>
<tr>
<th>Overlap (K)</th>
<th>Calculation of numerator components</th>
<th>Contribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( C(12, 0) \times C(12, 4) \times C(8, 3) \times 0 )</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>( C(12, 1) \times C(11, 3) \times C(8, 2) \times 1 )</td>
<td>55440</td>
</tr>
<tr>
<td>2</td>
<td>( C(12, 2) \times C(10, 2) \times C(8, 1) \times 2 )</td>
<td>47520</td>
</tr>
<tr>
<td>3</td>
<td>( C(12, 3) \times C(9, 1) \times C(8, 0) \times 3 )</td>
<td>5940</td>
</tr>
</tbody>
</table>

**Total Numerator:** 108900

Denominator = \( C(12, 3) \times C(12, 4) = 108900 \)
Hence, \( N = 1.000 \)

Example 3: \( n = 9; n_1 = 9; n_2 = 9 \).

<table>
<thead>
<tr>
<th>Overlap (K)</th>
<th>Calculation of numerator components</th>
<th>Contribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>( C(9, 9) \times C(0, 0) \times C(0, 0) \times 9 )</td>
<td>9</td>
</tr>
</tbody>
</table>

**Total Numerator:** 9

Denominator = \( C(9, 9) \times C(9, 9) = 1 \)
Hence, \( N = 9.000 \)

We can see from the examples 1 and 2 that as our domain expands keeping the individual sets of the same sizes, the expected overlap reduces. This is clear because the probability of a zero overlap increases thereby decreasing the probabilities of other integral overlaps.

Example 3 clearly shows that when domain size is equal to the size of both sets, then there is no other option but a complete overlap.
6 Construction of back links:

Back links are created at the time of traversal. When we start from a source attribute instance and traverse through the PC tree network till the destination attribute instance, we trace the path from the source instance to the destination instance. After reaching the destination, we setup a back link to the source instance, so that given the destination instance; we can trace the source instance. This is useful in the case of multiple source attributes as shown in figure 12.

![Diagram of PC tree network with back links]

**Figure 12: Back links**

In figure 12, back links are constructed as a result of the traversals to find out number of TVs purchased by age group 3 and Microsoft. Now, if we examine the destination nodes, we see two TV nodes. Now, there are back links from the left TV node to both age group 3 and Microsoft; whereas there is a back link from the right TV node to only age group 3; not to Microsoft. Hence, if we want to count TVs purchased by age group 3 only, then we consider both the TV nodes, but if we want to count TVs purchased by age group 3 and Microsoft then we will restrict ourselves to only the left node.

In the language of the analysis done previously, to count TVs for age group 3, \( Y = 8 \); whereas, for age group 3 and Microsoft, \( Y = 3 \). We multiply link fractions to this total destination count.

An important issue to note here is that the last level PC tree in the network must be derived out of a “transaction identifier” – “transaction” relation. This ensures all
traversals to necessarily converge at the last level. This is necessary because back links associate final transactions with their source instances and for this, the transactions themselves must be defined.

7 Consolidation of counts:

Here, we shall see how the framework works for all the multiplicities and how the count is successfully consolidated. These are relationships between attributes.

7.1 Single source attribute, single destination attribute:

The one-to-one case is a subset of the many-to-one case and will be covered under the link fraction scheme. For one-to-one links, the number of entities mapping on to one entity is 1, hence, $n = 1$ (refer section 3.1.2) and $x_i = X$. Hence, link fraction $x_i / X = 1$, thereby making $E = Y$.

7.2 Multiple source attributes, single destination attribute:

The many-to-one case is handled in section 3.1. In this case, the link fractions will be less than 1. As we traverse through the PC tree network from the source instance to the destination instance, we keep multiplying the link fractions encountered along the way and finally, multiply this product to the total destination instance count to obtain count corresponding to the source instance we started from.

7.3 Single source attribute, multiple destination attributes:

![Diagram](image)

Figure 13: One-to-many scenario
The analysis from section 4.3 can also be used in the one-to-many case. Consider this scenario (figure 6): We know the count of TVs corresponding to age group 3. Along with this information, consider the information about movies watched by people is known, where, we have the movie database containing a table indicating who watched which movie. So, we would have entries like (John, Die Hard) and the like. Hence, we can calculate the count of the watching of Die Hard by age group 3. What is the count of instances where a TV has been purchased and Die Hard has been watched by age group 3? This question is loose, because information about time is not stored in the databases, and the heterogeneous nature of the databases results in no uniformity in the transaction identifiers also. Hence, we can find the most expected number of such instances by an estimation of the overlap.

Here, the domain size is the total number of instances of the source attribute. The first set is the set of TV purchases, and the second set is the set of Die Hard sightings. The overlap, as calculated using analysis from section 4.3 will yield us our answer.

7.4 Multiple source attributes, multiple destination attributes:

This is not an independent case – it is a combination of many-to-one and one-to-many. We first carry out the analysis for case 7.2 to get counts for each destination attribute and then analysis for case 7.3 to arrive at our answers.

8 Correspondence with network model:

The network model is used to depict the data entities in a domain and their relationships. If two data entities are related, they are connected. The network model allows any number of connections leaving an entity and any number of connections entering an entity. Hence, it is a graph based model.

For example, the following scenario is one in which we have many boys and girls and we want to record who is friends with whom.

![Network model](image.png)

**Figure 14:** Network model
Our framework can be mapped to a network model because it is graph based. We connect relations within a database and between databases using graphs (local graphs and global graphs).

9 Future scope:

This framework has been used here to do the preprocessing for Association Rule Mining (ARM). The primary purpose is to compute the counts for instances of various associations, and the mining algorithm will form association rules based on frequency of tuples. As depicted in figure 1 (a), the framework starts with constructing links between the source and destination attributes and then converting to direct links as in figure 1 (b). The weights assigned to the direct links, are in this case, counts; but depending on the application, they can be based on any heuristic. Thus, the framework can be customized for use in any other application.

Thus, this framework can be considered as a contribution to solving the bigger link discovery problem in data mining.

10 Conclusion:

We have presented a framework which uses semantic network, generalization trees, and a collection of databases; builds a navigation paths and a PC tree structure to link the attributes across the databases and generates associations between generalized value set and value set of CAs.

An important contribution of our framework is to generate the inter-databases association rules and getting these rules at a higher degree of efficiency. We showed that that it is possible to efficiently obtain a semantic partition of the transaction databases implicitly using a collection of databases and semantic network. Further, generation of large itemsets needs scanning the databases only once. The performance study shows that effort involved in generating the large itemsets and the number of large itemsets generated on the same transaction databases is significantly larger in the case of syntactically partitioned algorithm than that of the semantically partitioned algorithm. Based on the simulation study it is observed that the number of large itemsets generated by the syntactic partition algorithm is larger by a factor of ten to few hundreds as compared to the semantic partition algorithm. So, the semantic partition method reduces computer time requirements considerably and provides for meaningful selection of candidates.

A novel data structure called PC-Tree can be used to represent the database in a complete and compact form. PC-tree can be constructed using a single database scan.
ARM algorithms based on PC-Tree are scalable. PC tree is ideally suited for Incremental mining also.