A NOVEL ALGORITHM: APRIORI-ANT THE COMBINATION OF APRIORI ALGORITHM AND ANTS COLONY OPTIMIZATION ALGORITHM

Choo Ai Ling

Master of Advanced Information Technology
(Data Mining and Intelligent Systems)
2005

Table of Contents

Acknowledgements

I would like to express my gratitude to the following persons and organization who had been of great help to me when I was compiling this thesis:

Associate Prof. Narayanan Kulathuramaiyer of Unimas, the Faculty Dean of Computer Science and Information Technology, who had taught me in the area of Data-Mining, and who had inspired me to choose this area to make my research.

Mr. Lee from Cognitive Science Faculty of Unimas who taught me in the area of Intelligent System.

Mr. Sylvester Arnab, my supervisor, who had been a great mentor and a spiritual supporter all along during the process of my thesis.

Kuching General Hospital from which I gathered data for my thesis. Special thanks to Mr. Kho and his staff in the Record Unit in helping me to trace the useful patients' case notes from a vast number of records.

Abstract

This thesis explores an innovative technique to extract data from a vast number of patients' records and then generate them into useful rules that can be used in an Expert System. Material used for the study of this thesis is a set of Skin Burn data.

Data-Mining is a process of gathering and analyzing data sets (often large) to find suspected relationship between them and summarize them in novel ways which are understandable and useful to a data owner.

Two well-known algorithms, Apriori algorithm and Ants Colony Optimization algorithm, are used in this thesis.

Apriori algorithm is the best-known rules discovery technique in Data-Mining, but its disadvantage is that it allows only the frequent itemsets to survive and form another level of itemsets and this process may leave out some interesting rules. Besides, the pruning and joining action in this algorithm prolongs the rule generation process.

Ants Colony Optimization algorithm is a probabilistic technique that can be reduced to find a good and shortest path.

In this thesis, the design of Apriori-Ant algorithm is to make use of the strength of Ants Colony Optimization algorithm to shorten the rule generation process in Apriori algorithm. At the same time, the joining process in Apriori algorithm is still retained until Level-2. The count from Level-1 (1-itemset) and Level-2 (2-itemset) is used in the modified pheromone formula in Ants Colony Optimization algorithm.

Rules are generated based on the highest pheromone value after 1000 cycles of Pheromone Update and Pheromone Evaporation process. No interesting rule is removed from the list if it is an infrequent item in Apriori algorithm. Thus all of the items are treated equal in this novel technique.

The experiment is carried out in two main areas: Processing Speed and Number of generated rules. From the finding, pheromone value and the items generated in 1-itemset (Level-1) and 2-itemset (Level-2) do greatly affect the Processing Time.

Chapter 1: Introduction

An Expert System is a system which dispenses expert advices and guidance on certain matters or problems. Thus, a Skin-Burn Expert System has to obtain expert knowledge from skin specialists and convert such knowledge into rules. The required information or medical data can be obtained from the case notes of skin-burn patients.

Medical data from patients' skin-burn case notes are not transactional data used in mining associate rules. Transactional data are data which can be obtained from a collection of items purchased at a grocery store or supermarket. The purchased items that go into each basket are analyzed to generate association rules.

An association rule is an implication of the form $X \Rightarrow Y$, where X and Y are each a set of items which occur together in a significant number of baskets. *Support* S is the *percentage of the total transactions* in which X and Y *occur together*. *Confidence* C is the number of transactions in which X and Y *occur divided by the number of transactions* in which only X *occurs and expressed as a percentage*. An association rule is one where S and C meet some minimum threshold requirements referred to as the *Minimum Support* (*Min Sup*) and *Minimum Confidence* (*Min Conf*) respectively.

This project introduces a novel technique – Apriori-Ant algorithm. It is an experiment to use non-transactional data to mine frequent patterns, which are data from a vast number of patients' case records, and generate them into association rules to be used in an Expert System. The data is different from the transactional data obtainable from the supermarket or grocery store.

Apriori-Ant algorithm is based on Apriori algorithm and Ants Colony Optimization (ACO) algorithm. Apriori-Ant algorithm uses only two I/O scans over the database for the whole process, thus eliminating the need for generating the candidate items.

So far, there is still no research yet on how to generate association rules from non-transactional data, and it is still not discovered as to whether or not non-transactional data can produce any useful rules.

Apriori algorithm [1, 3 & 15] is the best-known rules discovery technique in Data-mining. Data-mining, also known as Knowledge Discovery in Database (KDD), is a process of analyzing data sets (often large) to find suspected relationship between them and summarize them in novel ways that are understandable and useful to the data owner.

Apriori algorithm is an algorithm that finds interesting relationships between data. It mines the frequent itemsets to form rules; which employs an interactive approach known as level-wise search, searching from one itemset to another throughout the whole database.

This algorithm is good as far as finding quality rules are concerned, but if the database is large, then there will be many frequent items and many possible combinations. The reason is that, each time the candidate generation process will make as many scans over the database as the number of combinations of items in the antecedent, which is exponentially large. Due to combination explosion, it may lead to poor performance when frequent pattern sizes are large.

Apriori algorithm needs repetitive I/O disk scans over the database to generate frequent items and this involves huge computation during the candidates generation.

Another concern is that the number of frequent patterns is very much dependent on the support threshold. If the threshold is set to low, the occurrence of frequent patterns would be high. But if the threshold set to high, the occurrence of frequent patterns would be low, and this may reduce the quality of generated rules.

There are two measurements in Apriori algorithm which are called *Support measurement* and *Confidence measurement*. These two measurements are used to determine the threshold used in each joining and pruning process. The threshold is used to determine which items are to be pruned and which are to be joined in the next step. The frequent sets are those that support sets which are greater than minimum support threshold. Items that occur very infrequently or below the minimum support threshold in the data set are pruned although they may produce interesting and potentially valuable rules later.

As these two measurements may result in some interesting rules being left out, they are known as 'selfish' measurements because it allows only the itemsets greater than the minimum support to 'survive' to proceed to the next step.

An ACO algorithm [7 & 15] is a metaheuristic approach based on parameterized probabilistic model or pheromone model. A probabilistic technique is adapted to solve computational problems that can be reduced to find a good and shortest path.

The literature review [8, 9, 16, 18 & 20] shows that ACO algorithm is best for solving computational problems. The features of the main process of ACO algorithm, which are best for finding a good and shortest path in least amount of time, can be used to replace the tedious joining-and-pruning process in Apriori algorithm.

Joining-and-pruning process is the core process in Apriori algorithm, but this process requires repetitive I/O disk scans. So, it seems that the core process in ACO algorithm can be a remedy for the main process in Apriori algorithm.

This research identifies two main problems:

1. Repetitive I/O Disk Scans.

   In an Apriori algorithm, the candidates generation (joining and pruning action) takes up a lot of time and space while processing. Each cycle of candidates generation process requires full scan of database regardless of its size.

   To reduce repetitive I/O disk scans, either the candidates generation is to be reduced or the time-consuming candidates generation process be replaced.

   A lot of research has been done in this area. The proposed Apriori-Ant algorithm method is a technique which is basically intended for solving the problem of mining association rules, the large number of discovered patterns or rules [4], and the candidates generation process.

   Some researchers have used other similar techniques to replace candidates generation process such as the Trie method (similar to hash tree), HybridApriori; the combination of Apriori algorithm and AprioriTid algorithm, MLPFT. The Candidate

and Frequent Pattern Tree technique such as COFI algorithm and generation process is replaced by a tree technique, but there are some limitations in each algorithm (more detail in Chapter 2 – Literature Review).

2. Huge computational cost required to generate frequent items.

   In Apriori algorithm, the candidate generation process involves not only the joining and pruning process, but also two other computational processes of Support measurement and Confidence measurement.

   The purpose of Support and Confidence calculation is for the next candidate generation. These values are to identify which itemsets have values that are above the threshold (Minimum Support value and Minimum Confidence value). Those items below the user-defined threshold are removed from the list which are called an infrequent itemsets.

   User-defined threshold may lead to poor quality rules being generated. There is no specific level to be taken as the best value to be used as the threshold in the process. If the threshold is set to low, then a high number of frequent patterns would be admitted, which thus requires much space for the searching, massive I/O, and high memory dependencies. If the threshold is set to high, it may not generate vast number of patterns, but then quality of generated rules may suffer.

   Each time a change is made on the user-defined threshold, the process needs to start over again. This makes Apriori algorithm technique time-consuming and requires high computational cost.

The pruning process may remove many rules that are useful. This brings us back to the problem of user-defined threshold in which the threshold that is set to high would generate a small amount of frequent patterns and thus may turn to be biased.

Some itemsets that are below the threshold may have good and interesting combination with other itemsets. For example, bread & margarine is combined with sugar & condensed milk. Sugar and condensed milk may not be interesting by themselves or with margarine, but it may be interesting if combined with both bread and margarine as some children like to have bread with some sugar spread on top of margarine or just bread with condensed milk.

The ideal way is to save all the itemsets and let an efficient process decide which itemsets are to be pruned and which are to be saved. This is thus an unbiased process.

The objective of this research is to find out how to improve an Apriori algorithm to solve the problems above and how an infrequent itemsets in Apriori algorithm can produce a series of useful association rules use in Medical Field. The proposed method is one which combines an Apriori algorithm with an ACO algorithm.

The research focuses mainly on the following questions:

1. How does an ACO algorithm replace the candidates generation process in Apriori-Ant algorithm to generate rules? Can this method solve the I/O overhead?

2. How efficient is the modification of measurement method (Support, Confidence and Lift) used in Apriori algorithm? Can it work well with an ACO algorithm? Can the

values generated by the Apriori algorithm measurement method be used in the state transition rule module in ACO algorithm which allows "*ants to move from one item (city) to another item (city)*"?

3. What affects the processing speed in Apriori-Ant algorithm? Is it the number of records or the number of rules generated? Or is there any other reason?

In this proposed algorithm, every itemset is treated equally, which means no itemset is pruned even if it is infrequent or below the user-defined threshold. The idea is not to prune the itemset during the Apriori algorithm but to calculate each of its support and confidence measurement. It is to make sure that the itemset which are below the minimum threshold will not pruned but to generate rules.

This design involves 5 parts in the process: Part one is to convert the data from text file into string data for processing. Part two is to generate the frequent itemsets and frequent counts. Part three is to perform the calculation. Part four is to join the rule based on the pheromone value. Part five is to write the rules into text file. All the processes can be found in Chapter 4 Methodology and Chapter 5 Experimental Result.

In part one, array [5 & 14] is introduced to handle the data read from the text file. All the string data will be stored into a string array for later process.

In part two, Apriori algorithm is introduced. This design adapts the Apriori algorithm method to find frequent itemsets and frequent counts for the respective itemset. But the process stops at the second level of finding, no pruning process is carried in this section.

The Apriori algorithm process is to find and store the frequent itemsets and frequent counts for 1-itemset and 2-itemset into the respective array.

Part three consists of two different computational sections. The first computation section is based on Apriori algorithm. In this section, one modified and two original measurement methods are used. The modified measurement method is Lift measure, and the original measurement methods are Support measure and Confidence measure that are used to assist in Lift measure.

The second computation section, which is based on ACO algorithm, substitutes the values that are generated from Apriori algorithm into the modified pheromone probability formulas. This value is used to decide which item is joined into the rule set and not to start a tour.

In part four, after probability pheromone values are generated, the rules generating process takes place. A tour is started from a randomly generated item in 1-itemset list to be matched with the items in 2-itemset list and only the item with the highest pheromone value would be joint into the rule set. This process is carried out until no more matching is found in 2-item sets. Modified Pheromone update and Pheromone evaporation is carried out in each cycle of process.

The last part is to write out the rules into a readable text file. A *connection* program is required to encode the text file so as to be readable by an Expert System.

The arrangement report is as below:

Chapter 2: Literature Review.

5 materials for Apriori algorithm and 5 materials for ACO algorithm are reviewed in this section.

For Apriori algorithm, the review is on techniques to improve its internal processing part. Research is done to improve efficiency of Apriori algorithm which includes techniques to replace the tedious pruning and joining process.

For ACO algorithm, the review is on techniques and how efficient ACO algorithm can be used. It also includes modified techniques used in ACO algorithm and techniques in ACO algorithm with Data Mining technique.

Chapter 3: Data Collection Analysis.

Data are manually collected from 190 patients' case notes of Kuching General Hospital and then converted into a digital form that can be used by this prototype system. This section shows how the process is carried out from data collected in manual format to digital format and how the digital format is then used in this prototype system.

Chapter 4: Methodology.

Methodology section covers the modification of Apriori algorithm, ACO algorithm and Rules Generation.

In Apriori algorithm, only the joining process is carried out which stops at Level-2 of the joining. No pruning process is performed. The output of Apriori algorithm is the 1-itemset generated at Level-1 with its count and also 2-itemset generated at Level-2 with its count.

In ACO algorithm, Lift-measurement is introduced into Apriori-Ant algorithm. The count at Level-1 and Level-2 is used in Lift measurement method to calculate the Pheromone value. And the Pheromone value is used in Pheromone Update and Pheromone Evaporation process in later part.

Methodology section also explains how rules are generated. Rules are generated based on the pheromone value calculated by the modified formula.

Chapter 5: Experimental Result.

Experiment is done in 4 areas: Processing Time, Rules Generation, Manual-Checking Analysis and From Generated-Text file to Expert System.

Processing Time: The experiment reveals that the influencing factors of processing speed are: *Total Pheromone value* and *Number of records*.

Total Pheromone value: The first finding of the experiment reveals that more pheromone requires more processing time.

Number of records: Apparently, more records should require more processing; but in the experiment, it is to find out how to reduce the process time regardless of number of records involved in the processing.

Rules Generation: The Experiment investigates two areas: What influences the number of rules generated and how are the rules generated?

It is found that the number of items in 1-itemset and 2-itemset influences the number of rules generated. The number of items in 1-itemset influences the number of items in 2-itemset. Items in 2-itemset indicate the number of rules generated by this prototype.

From Generated-Text file to Expert System: The rules generated by this prototype system are stored into a text file. Before the rules can be used in an Expert System, another program is needed to re-arrange the rules into format that is compatible with Expert System.

Output from generated text file can be re-arranged in two areas: *General* and *Burn-Area*.

General Area: Rules are re-arranged in general basis. It is not based on any specific area like Burn-Area-*'Flame'*, or Body-Area-*'Finger'*.

Burn Area: Rules are re-arranged according to specific Burn-Area. For example, in *'Flame'* Burn-Area and *'Hot Oil'* Burn-Area, different treatments and medication are applied to each of them. Thus, in this section, rules are re-arranged based on the Burn-Area.

Manual Checking Analysis: This analysis is done manually in two sections: *In Apriori-algorithm* and *in Apriori-Ant-Internal-Calculation*.

In Apriori-algorithm: The checking is carried out on the rules generated by the prototype with Apriori algorithm in manual form. And the results are compared with those generated by the prototype system.

In Apriori-Ant-Internal-Calculation: Checking is done on the internal calculation in Apriori-Ant algorithm using Microsoft Excel. The results are used to tally those produced by the prototype system.

Chapter 6: Discussion.

This research is to find out how Apriori-Ant algorithm solves the length pruning and joining process in Apriori algorithm; and how ACO algorithm is introduced into the middle part of Apriori-Ant algorithm.

Chapter 7: Conclusion.

Conclusion of this research.

Chapter 2: Literature Review

The aim of data-mining is to find the "***hidden gold***" in a data source; meaning to extract valuable information hidden in that data source. Such hidden valuable information may reveal a new business trend based on purchasing habits of consumers. Data-mining techniques are thus based on data retention and data distillation.

Apriori algorithm is one of the data-mining techniques which seeks to find and generate frequent itemsets by using support measurement to prune itemsets which are not frequent. Eliminating non-useful information in this way would save more space for more memory storage in a computer and thus improves its running time.

The objective of ACO algorithm is to solve Non-Polynomial problem (NP-problems) like route planning, scheduling, creating of timetables, etc. These problems need to be solved by using an approximation technique which is not an optimal solution to a given problem but is a solution that is good enough for that specific application.

2.1 Apriori algorithm

The basic concept of association rule mining is:

First, Apriori algorithm would analyze all the transactions in a dataset for each item support count. Let $J = \{i1, i2 \ldots im\}$ be a set of items. Let D, the task-relevant data, a set of database transactions where each transaction T is a set of items such as $T \subseteq J$. Any item that has a support count of less than the minimum support count threshold is removed from the candidate items list.

The frequently used measure methods in Apriori algorithm are Support and Confidence. To calculate a Support *(1)*, let A be a set of items. A transaction T is said to contain A if $A \subseteq T$. An association rule is an implication of the form $A \Rightarrow B$, where $A \subset J$, $B \subset J$ and $A \cap B = \phi$. The rule $A \Rightarrow B$ holds in the transaction set D with *Confidence* c if $c\%$ of the transactions in D that contain A also contains Y. The rule $A \Rightarrow B$ has supports s in the transaction set D if $s\%$ of transaction in D contains $A \cup B$. (i.e., both A and B).

$$Support(A \Rightarrow B) = P(A \cup B). \qquad (1)$$

For confidence method *(2)*, it is taken to be the probability, $P(A \cup B)$. The rule $A \Rightarrow B$ has confidence c in the transaction set D if c is the percentage of transactions in D containing A that also contain B. This is taken to be the conditional probability, $P(B \mid A)$.

$$Confidence(A \Rightarrow B) = P(B \mid A). \qquad (2)$$

Rule support and confidence are two measures of rule interestingness. Association rules are considered interesting if they satisfy both a minimum support threshold and a minimum confidence threshold. Such thresholds can be set by users or domain experts.

There are two-step processes in Apriori algorithm: Join step and Prune step.

In Join step, for example, in order to find L_k, a set of candidate k-itemsets is generated by joining L_{k-1} with itself. The join $L_{k-1} \bowtie L_{k-1}$ is performed, where members of L_{k-1} are joinable if the first $(k-2)$ items are in common.

In Prune step, a scan of the database to determine the count of each candidate in C_k would result in the determination of L_k. C_k is a superset of L_k that is, its members may or may not be frequent, but all of the frequent k-itemsets are included in C_k.

Any $(k-1)$-itemset that is not frequent cannot be a subset of a frequent k-itemset. Hence, if any $(k-1)$-subset of a candidate k-itemset is not in L_{k-1}, then the candidate cannot be frequent either and so can be removed from C_k.

In the implementation of Apriori algorithm, the item that is less than the minimum support count will be removed from the candidate list. And items whose support count is greater than the minimum support count will be stored in one-candidate-itemsets list.

The remaining itemsets in one-candidate-itemsets list are joined to create two-candidate-itemsets list. The calculation of support count for two-itemsets is performed. Only if the support count is greater than the minimum support count, the remaining two-itemsets are joined to create three-candidate-itemsets. This process is iteratively performed until all item's support counts in the candidate-itemsets list are less than minimum support count.

All the candidate-itemsets generated with a support count greater than the minimum support count form a set of frequent itemsets. Apriori algorithm recursively generates all the subsets of each frequent itemset and creates association rules based on the subsets with a confidence greater than or equal to the minimum confidence. Thus a lot of interesting rules are pruned at this stage if it is an infrequent itemset.

$$confidence(A \Rightarrow B) = P(B \mid A) = \frac{support\_count(A \cup B)}{support\_count(A)} \qquad (3)$$

Where *support_count*$(A \cup B)$ *(3)* is the number of transactions containing the itemsets $A \cup B$, and *support_count*(A) *(3)* is the number of transactions containing the itemset A.

Thus it makes multiple passes over the database before generating a useful rule. It brings the problem to I/O overhead and high memory need.

This research shares some similarities with [1, 2, 6, 17 & 19]. Let us look into Apriori algorithm. Rakesh Agrawal and Ramakrishnan Srikant [1] are the Guru of Apriori algorithm. In [1] they proposed an AprioriHybrid algorithm to mining association rules in a faster way. AprioriHybrid algorithm is the combination of Apriori algorithm and Apriori-Tid algorithm.

Authors [1] focused on the problem of discovering association rules between items in a large database. Problem of mining association rules is to generate all association rules that have support and confidence greater than the user-specified *Minimum-Support* and *Confidence-Support*.

Let $I = \{i_1, i_2, \ldots, i_m\}$ be a set of literals, called items. Let D be a set of transactions, where each transaction T is a set of items such that $T \subseteq I$. In AprioriHybrid algorithm, associated with each transaction is a unique identifier, *TID*. Transaction T contains X, a set of some items in I, if $X \subseteq T$.

An association rule formed in AprioriHybrid algorithm is based on Apriori algorithm pattern. An implication of the form $X \Rightarrow Y$, where $X \subset I$, $Y \subset I$, and $X \cap Y = \phi$.

In AprioriHybrid algorithm, Apriori algorithm is introduced at the first level, it examines every transaction in the database. At this level, Apriori algorithm is used to generate the candidate itemsets, and the itemsets are to be counted in each pass and

only the itemsets found large (frequent) in the database are survived to form the next level of set-of-itemsets.

After the candidate itemsets are generated, Apriori-Tid is introduced. In Apriori-Tid, the database is not used at all for counting the support of candidate itemsets after the first pass done in Apriori algorithm. If a transaction does not contain any candidate k-itemsets, then the set-of-itemsets will not have entry for this transaction thus the number of entries in set-of-itemsets maybe smaller than the number of transaction in the database.

Apriori algorithm is in the initial passes and later switch to AprioriTid when it expects the set-of-itemsets at the end of the pass will fit in memory.

The advantages of AprioriHybrid are that the size of the set-of-itemsets declines in the later passes; it scales linearly with the number of transactions, and the execution time decreases a little as the number of items in the database increases.

But the limitations are that, it incurs switching cost without realizing the benefits, and it suffers from the repetitive I/O disk scans. The candidates generation process still takes place in every cycle until the rules are generated. Although AprioriTid reduces the number of database, it still needs to scan in each candidate generation process.

In [2], the authors focus on algorithm running time and memory need. In this research [2], the problem for frequent itemsets mining is to find all frequent itemsets in a given transaction database, and this requires high running time and memory space as it needs to scan through the database more than one time.

As already mentioned, using Apriori algorithm requires repetitive I/O disk scans and memory space as it scans through database to find all the frequent itemsets. To reduce these requirements, the authors focus on the central data structure. They introduce a Trie Method to store not only candidates but also for frequent itemsets; which works like a hash-tree.

Tries are not only suitable to store and retrieve words but also applicable to any finite ordered sets. In the design, a link is labeled by an element of the set, and the Trie contains a set if there exists a path where the links are labeled by the elements of the set in increasing order.

The authors [2] modify the Support Count Methods with Trie. Support Count Method takes the transactions one-by-one. If a subset of an item is found to be a candidate, it would increase the support count of the respective candidate by one. This method does not generate all the respective subsets of certain transaction, but would perform early quits whenever possible. In this approach, Trie stores not only candidate but also frequent itemsets.

Support Count is done by reading transactions one-by-one to determine which candidates are contained in the actual transaction. Two simple recursive methods are introduced to solve the problem of finding candidates in a given transaction. And it is found that the running time difference is determined by the recursive step.

The Trie is built with frequency codes instead of original codes because we know exactly the frequency order after the first reading of the whole database.

Apriori-Brace is created to keep track of the memory need and stores the amount of the maximal memory need. Apriori-Brace does not carry on with support count, but to check memory need. If the memory need exceeds the maximal memory need, the support count is evoked and maximal memory need counter is updated, if not candidates are generated. The process is repeated until memory need does not reach maximal memory need.

This procedure collects together the candidates of the latter phases and determines their support in one database reading. To avoid generating false candidates, it generates only candidates that are frequent.

It is more useful to store up input data of the same transactions. Thus the approach is to choose only reduced transactions for storage because reduced transactions consist of only the frequent items of the transaction and yet have all information needed to discover frequent itemsets of large sizes.

In [19], the authors focus on the candidates generation process that leads to poor performance when frequent pattern sizes are large. They also focus on calculation of association rules that are from raw itemsets by using Apriori algorithm. The method still needs to scan the entire database on each pass and eliminate the candidate generation process.

The method proposed by [19] is Pattern Repository to read and store patterns from the raw itemsets in a compact form. Information about association rules could be derived from the Pattern Repository and calculated for display. The calculation time is spent only on possible rules which is different as compared to Apriori algorithm whereby the

calculation is carried out on every itemset before pruning process takes place. And candidate generation process in Apriori algorithm is avoided in Pattern Repository approach.

Unlike in Apriori algorithm, Pattern Repository avoids extra database scan. The design of Pattern Repository is based on Frequent Pattern tree which scans the database twice. But the Frequent Pattern tree design requires an in-memory operation and is not updateable. If support levels or new data are present, the entire database must be read twice again in Frequent Pattern Tree approach.

Pattern Repository uses a unique symbol (unique number) in itemsets that would be replaced with a token. The pattern in the database contains sufficient information to reproduce the items in each itemset and produce statistic values about each item. The Pattern Repository approach extracts records that contain the token (frequent) and the counts of the other items in the transactions. The other items that meet the minimum support level are put in numerical order to pair with the original token in a list. Pattern Repository extracts those records that contain the token pairs and counts the tokens that meet the minimum support level.

There are two advantages in Pattern Repository: first advantage is that, adding new record in Pattern Repository Approach would not create any update problem as compared to Frequent-Pattern tree, and each time new records are added or there are changes on minimum support value, the tree needs to be rebuilt. The Second advantage is that the system is scalable by using multiple processors or disk storage.

In Pattern Repository, 3 limitations are discovered. The first limitation is that, in their theory, infrequent items do not contribute to association rules, but actually infrequent items may generate some good association rules. Second limitation is that actual run-time is still very sensitive to the minimum support setting in which lower setting would make the search take longer. And the third limitation is that, the candidates generation still takes place in the first level generation.

In [6], the authors focus on the problem of generating frequent itemsets and how to reduce the candidancy generation.

In this research, the authors [6] proposed *Co-Occurrence Frequent Item* tree algorithm (COFI-tree) to reduce candidacy generation in Apriori algorithm. The design is also based on Frequent Pattern Tree algorithm.

In COFI-tree approach, 2 stages are carried out. In first step, 2 full scans are required to build the Frequent Pattern tree in the First step. The first scan is to identify the frequent 1-itemsets. Items in 1-itemsets will be sorted according to their frequency. Those items below the defined threshold will be removed. And the second scan is only carried out on the frequent itemset (1-itemsets) present in the header table collected and sorted. Header table is to store the 1-itemsets with their frequency higher or equal to the defined threshold. These sorted transaction items later will be used to construct the Frequent Pattern tree.

In second step, the process of building the small data structure is repeating. Approach for computing frequencies relies first on building independent relatively small trees for each frequent item in the header table of the Frequent Pattern Tree.

Pruning is applied to remove all infrequent items with respect to the main frequent item of the tested COFI-tree. Pruning is done in either after building the tree or while building the tree. The tree is discarded as soon as it mined.

In COFI-tree approach, it uses new anti-monotone local infrequent or global frequent property. It is to eliminate the frequent items set at *i-itemset* and sure that they are not joint in (*i+1*) *itemset*. In Apriori algorithm, all non-empty subsets of a frequent itemset must also be frequent but in COFI-tree approach, all non-empty subsets of a frequent item with respect to the item *A* of the COFI-tree must also be frequent with respect to item *A*.

In this design, only globally and locally frequent items will participate in the creation of COFI-tree. It is similar to the conditional Frequent Pattern Tree approach. COFI-tree approach has bi-directional links in the tree thus allowing bottom-up scanning.

During the mining process, the COFI-tree of all frequent items are not constructed together. Each tree is built, mined then discarded before the next COFI-tree is built. The mining process is done for each tree independently with the purpose of finding all frequent *k*-itemset patterns in which the item on the root of the tree participates.

There are two advantages of COFI-tree algorithm: first is, it only involves two scan on database thus reduces the candidate generation process and reduces the number of itemset generate in each process. Second advantage is that the tree is discarded once the frequent itemsets are mined, thus it reduces the memory to store the built tree. But the limitation of this algorithm is that, once the defined threshold is changed, it needs to rebuild the tree. Although the tree being discarded after being built is an advantage to

memory needs, there is a limitation whereby there is no way to trace back the patterns of the tree. Another limitation is that, in COFI-tree algorithm, it requires 2 full scans on database, which thus would slightly increase the I/O disk scans.

In [17], it shares the same focus with [1]. Here the authors focus on the candidates generation process, and they propose a new approach called ***Multiple Local Frequent Pattern Trees*** (MLFPT) which is slightly different from that of [2].

Multiple Local Frequent Pattern Trees is the parallel implementation based on FP-growth algorithm in [13]. ***Multiple Local Frequent Pattern Trees*** it does not require an interactive generation of candidate frequent itemsets and it scans the database only twice to build a special data structure.

Multiple Local Frequent Pattern Trees involves two stages. First is the construction of parallel frequent pattern trees and each parallel frequent pattern trees allocates one processor each for the process.

The construction is done in 2 phases: In Phase 1, the initial scan of the database is to identify the frequent 1-itemset. The purpose is to generate an ordered list of frequent items and the tree is built in Phase 2. Database used in this construction is divided equally among the available processors.

Construction of frequent pattern tree for each available processor is done in Phase 2. In this phase, it requires a second complete I/O scan of the database, each processor also reads the same number of transactions in Phase 1. After the reading, each processor starts to build its own frequent pattern tree.

Mining is carried out in the second stage. In this stage, the actual mining for these data structures much like in the FP-growth algorithm. It starts with a bottom-up traversal of the nodes on the Multiple Local Frequent Tree structures. Total sum calculation of global support is carried for all items and divides them equally by the number of processors to find the average number of occurrences that ought to be traversed by each processor. The purpose is to let all FP-tree shared by all processors thus reduces the I/O cost and utilize most of the processing time.

The advantage of this algorithm is that, it reduces the candidate generation and at the same time it uses processors to achieve better load balancing with the goal of distributing the work fairly among processors for the mining process. But, the limitation is that, optimum balance between processors is difficult to obtain.

2.1.1 Summary

Apriori algorithm is famous for its candidates generation process that generates rules from transactional dataset. The two main steps involved in candidates generation process are joining and pruning.

But this candidates generation process is found to be time-consuming because each time the process takes place, the system go through full and repetitive scans of the database. Several researches have been carried to try to reduce the number of candidates generation process cycles and as well as to reduce repetitive I/O scans over the database.

In [1], the authors focus on the problem of discovering association rules between items in a large database. AprioriHybrid algorithm is designed based on Apriori algorithm and Apriori-Tid algorithm. Apriori algorithm is used to generate candidate itemset, the concept design of which is shared by Apriori-Ant algorithm. And rules are formed based on Apriori algorithm

In [2], the authors focus on algorithm running time and memory need. Trie method, which works like a hash-tree, is introduced, and its function is store candidates and frequent itemsets. The authors modify Support Count method with Trie, which takes the transactions one-by-one. The Trie is built with frequent codes.

The authors in [19] focus on the candidates generation process, which is also one of the focuses in Apriori-Ant algorithm research. The authors propose 'Pattern Repository' to read and store patterns from raw materials. Pattern Repository avoids extra database scans. The design of Pattern Repository is based on Frequent Pattern Tree which scans

the database twice. Apriori-Ant algorithm shares a common method which scans the database only twice and thus reduces the candidates generation to two cycles only.

The author [6] shares the same focus as in [19]. They focus on the problem of generation frequent itemsets and how to reduce the candidates generation. The author proposes Co-Occurrence Frequent Item Tree algorithm (COFI-Tree) and the design is based on Frequent Pattern tree. Two steps are involved in COFI algorithm: First step, 2 full scans are required to build the Frequent Pattern tree. The first scan is to identify the frequent 1-itemset and the second scan is on the frequent 1-itemset to construct the Frequent Pattern tree. In the 2nd step, the process of building small data into FP-tree is repeating. The tree is discarded as soon as it is mined.

In [17], the authors share the same focus with those in [1] but they [17] propose a new approach called Multiple Local Frequent Pattern Trees (MLFPT) which is slightly different from that of [2]. MLFPT does not require interactive generation of candidate frequent itemsets and it scans the database only twice to build a special data structure. MLFPT involves 2 stages: First is the construction of Parallel Frequent Pattern trees and second involves mining. The actual mining for the data structures is very much like that in FP-growth algorithm.

2.2 Ants Colony Optimization algorithm

Ants Colony Optimization (ACO) algorithm is a paradigm for designing metaheuristic algorithm for combinatorial optimization problems (NP-problems) which is inspired by the pheromone trail laying behavior of real ant colonies. ACO algorithm has been applied to many types of NP-problems, ranging from the classical traveling salesman problem to routing in telecommunication networks.

ACO algorithm solves their problem by multi-agent cooperation using indirect communication through modification in the environment known as Stigmergy. ACO algorithm is multi-agent systems; each single agent is a sort of artificial ant. The artificial ants would leave a trail of substance called pheromone as it crawls along. Each time it crawls along the same trail, more pheromone would be accumulated on that trail and thus makes the pheromone trail stronger.

The artificial ants are able to identify the pheromone leftover by the previous ants and would therefore choose the path with the highest pheromone concentration with higher probability. This is the characteristic of ACO algorithm, in which they make explicit use of elements of previous solution. It is a positive feedback called autocatalytic behavior.

The negative feedback is the evaporation of the pheromone trail. Evaporation is carried out to decrease the pheromone trails in each iteration of the algorithm.

2.2.1 Pheromone trail.

Ants communicate among themselves through pheromone. Pheromone is a substance deposited by an ant when it crawls along and thus forms an ant pheromone trail. The

more pheromone is deposited on the trail, the more attractive it becomes to the other ants to follow. The pheromone on the shorter path will therefore be more strongly reinforced and will eventually become the preferred route for the stream of ants. The pheromone trail is a very useful commuting device for the ants because it helps them to establish the shortest routes to their food source.

In ACO algorithm, the pheromone trail is the probability value between *city i* and *city j*. Each ant would build a tour at a starting city based on the pheromone trail. The smaller the value the better it is.

2.2.2 Transition rule.

ACO algorithm is famous for solving NP-class problem - Traveling Salesman Problem. In the algorithm, each ant is placed to different city in the system.

A brief explanation of an ant travel from *city i* to *city j* at iteration:

- Has the city been visited? Each ant will have its own Tabu memory. Tabu memory is a list used to store the cities that have been visited, and forbids the ant to visit them again until a cycle has been completed.
- The probability for a single ant to travel from *city i* to *city j*. The Ant will only choose the small probability of the city to travel.
- After the completion of the tour for the single ant, it would deposit the pheromone on each edge (*city i*, *city j*) that it has used.
- The pheromone trail evaporates a little after every iteration, and is reinforced by good solution generate by next ant.
- Trail updating is done after a complete tour cycle.

In [8 & 9], they use ACO algorithm to solve NP-problems such as route planning (symmetric and asymmetric Traveling Salesman Problem) and industrial scheduling.

In [9], the authors apply ACO algorithm to solve Symmetric and Asymmetric Traveling Salesman Problem. Traveling Salesman problem (TSP) is the problem of finding a minimal length closed tour that he visits each city once. In the design, cities $v_i \in V$ are given by their coordinates value (x_i, y_i) and d_{rs} is the Euclidean distance between two different cities r and s.

For Asymmetric Traveling Salesman Problem (ATSP), if distance from r to s is not equal to the distance from s to r, $d_{rs} \neq d_{sr}$, then TSP becomes an ATSP.

In ACO algorithm, the first applied module is state transition rule module. An ant situation in city r moves to city s by using state transition rule. This state transition rules would favor transitions towards nodes connected by short edges with a high amount of trail.

Local-updating rule helps to update ants each time they visit edges and change their trail. Once all ants have completed their solution, edges belonging to shortest tour made by the ants have their trail changed by applying Global-updating rule. They focus on the global-updating rule that can be used to solve the problem. Global trail updating provides a higher amount of trail to shorten the tours.

They have two different ways to choose the ant that allows the performance of the global updating: iteration-best updating and global-best updating. Their research adapts the global-best updating method, which selects the selected agent that does the shortest tour since the beginning of the computation.

From their research, ATSP problem is the heuristic function that represents the inverse distance between node r and node s. In ATSP, it only uses the accumulated trail to generate new tours.

They have also found out that a good value for parameter t used in updating rule is $(n \cdot L_{nn})^{-1}$, where L_{nn} is the tour length produced by the nearest neighbor heuristic and n is the number of cities.

The authors have categorized three families of edges. *BE* (Best edges) - those belonging to the last best tour, *TE* (Testable edges) - those that do not belong to the last best tour but recently did, *UE* (Uninteresting edges) - those that either have never or have not for a long time belonged to a best tour.

From the finding, Ants Colony Systems (ACS) favors exploitation of edges in *BE* and exploration of edges in *TE*. An interesting aspect is, while edges are visited by ants, the application of local updating rule makes their trail diminish, making them less and less attractive. From what they observe in their experiment., edges in *BE* when ACS achieves a good performance will be approximately downgraded to *TE* after an iteration of the algorithm and that edges in *TE* will soon be downgraded to *UE*.

From their research, they also find out that with the global-best updating method, the system performs best in terms of average time per trail and in terms of the quality of the generated tours. That is to say, time and quality of the tour is important.

The authors have slightly modified ACS algorithm which incorporate a more advanced data structure known as candidate list. Candidate list is a data structure normally used to solve big TSP problems. It contains a list of preferred cities to be visited.

In [8], the author has proposed an augmented ACO algorithm for industrial scheduling. A number of additions to the basic ACO algorithm to solve the industrial scheduling problem have been proposed.

First, the authors propose to use multiple distance matrices to define terms in the transition rule, which gives better adjustment of the exponent parameters and leads to increase control of the algorithm. The research finds that the use of 3 separate matrices allows for better adjustment of the exponent parameters. The produced solutions prove to have better quality as compared to using single distance matrix approaches.

The authors have also proposed a look-ahead feature to provide look-ahead information about the potential of the current partial solution. The look-ahead information carries an estimation of the potential quality of a partial solution by combining a surrogate function value for the current partial solution.

From the research, it is obvious that an ACO algorithm can be used for generating decision rules, which has inspired the idea for the proposed research.

In [20], the authors introduce an agent-based ACO algorithm. The design is based on an agent model that consists of Percepts, Actions, Goals and Environment. That is to say, the author has combined the ACO algorithm with an agent model.

In an Environment, virtual agents are allowed to read the pheromone and the heuristic value of each edge that is leaving from the node where it is currently located.

Virtual agents manipulate the environment by changing the pheromone weights, that is to say, an environment is deterministic. The pheromone weights at the edges are constantly modified and their value has great influence on the movement of the agents.

Additionally, it is possible for the virtual agents to communicate with each other. Such an environment should provide information about the cost of each possible step in terms of solutions quality, which is to prevent the agents from performing mere random search if no pheromone information is available.

In Percepts, virtual agents are able to perceive the environment in a very local area around their current position. The goal of the individual agents is not to construct high quality solution but to construct a valid solution. To achieve its goal of constructing a valid solution, a sequence of actions is repeatedly performed.

Consequently more ants that crawl over the short path will deposit higher pheromone and that will lead even more ants to choose that path. ACO algorithm not only emphasizes on finding good path; it is also necessary to decrease all pheromone weights after each generation by a small value. The reason to decrease the pheromone weight is to prevent the agents from prematurely getting stuck in a rather bad solution, which is one of the most critical problems of neighborhood-based optimization techniques.

In [18], the authors have introduced the first version of Ant-Miner algorithm. Ant-Miner algorithm is under the concept of Data Mining techniques. It is an Ants Colony

Optimization for discovering classification rules with the system's ant-miner. The argument is that an ant-based search is more flexible and robust than traditional approaches that use heuristic value based on entropy measure.

ACO algorithm involves simple ants that cooperate with one another to achieve an emergent, unified behavior for the system as a whole, producing a robust system capable of finding high-quality solutions for problems with a large search space.

In the design, Ant-Miner has 4 sections, namely: heuristic function, rule pruning, pheromone updating, and indirect pheromone evaporation function.

In general, an Ant-miner is to extract classification rules from data. Ant-Miner discovers rules referring only to categorical attributes; therefore, continuous attributes have to be discretized in a preprocessing step before mining process takes place.

As usual, Ant-Miner has pheromone initialization to start the whole process of rules constructions before calculating the heuristic value. Immediately after the construction of a rule, rule-pruning is undertaken to increase the accuracy of the rule. The quality of a rule is measured using sensitivity, specificity and accuracy measurement.

Ant-Miner follows a sequential covering approach to discover a list of classification rules covering all, or almost all in the training case. The training set consists of all training cases used to compare the generated rule from the system. The generated rule will be added into the list of discovered rules. If the training cases are correctly covered by the generated rule, then it will be removed from the training set. The process is repeated until the number of uncovered training cases is greater than a user-specified threshold.

The heuristic function, used by Ant-Miner is the entropy measure. This entropy measure is computed for an attribute value pair only, so that it is more sensitive to attribute interactions problems. In the design, Ant-Miner entropy measure is used together with pheromone updating which thus makes the rule-construction process more robust and less prone to get trapped into local optima in the search space. The higher the heuristic value the more uniformly distributed the classes are, and the smaller the probability that the current ant chooses to add condition rules to its partial rule.

Rule pruning is a commonplace technique in data-mining. The goal of rule-pruning is to remove irrelevant terms that might have been unduly included in the rule and it increases the predictive power of the rule, thus helping to avoid its overfitting to the training data. This improves the simplicity of the rules, making the rules more understandable to end-user.

The process of rule pruning is to iteratively remove one-term-at-a-time from the rule so as to improve the quality of the rule. The process is repeated until the rule has only one term or until there is no term.

Pheromone updating is to produce better and better rules. The initial amount of pheromone deposited at each path position is inversely proportional to the number of values of all attributes.

In Ant-Miner, pheromone evaporation is implemented in an indirect way. Pheromone evaporation for unused terms is achieved by normalizing the value of each pheromone. At normalization time, the amount of pheromone of an unused term will be computed by dividing its current value by the total summation of pheromone for all terms.

In [16], the author proposes an improved version of Ant-Miner called Ant-Miner3. The author modifies the pheromone update and the choice of transition (transition rule).

The modified pheromone update version is much easier and efficient. After an ant constructs a rule, the amount of pheromone associated with each term that occurs in the constructed rule is updated by the modified pheromone update method, and the pheromone of unused terms is updated by normalization.

In the choice of transition, the result depends not only on the heuristic functions and pheromone but also on a random number, which increases the chance of choosing term not used in previously constructed rules.

From the finding, Ant-Miner3 discovers more rules than Ant-Miner1; the accuracy of the rule sets discovered by Ant-Miner3 is higher compared with Ant-Miner1.

The research discovers a method that incorporates tunable stochastic element when constructing a rule. Thus it provides a balance between exploitation and exploration in its operation. A different strategy for controlling the influence of pheromone values is studied in the research.

2.2.3 Summary

In this proposed research, ACO algorithm is introduced to the internal processing in Apriori algorithm. The purpose of this introduction is to replace candidates' generation (joining and pruning actions) in Apriori algorithm.

From the literature review, ACO algorithm use pheromone trail and state of transition to discover more understandable rules for end-users. Pheromone trail is a substance deposited by an ant as it crawls along a trail. The stronger the trail, the higher is possibility of other ants following that trail. It is a way communication among ants.

However in [20], the authors create an agent-based ACO algorithm. An environment model in agent-based ACO algorithm is that each ant can communicate with each other. As we know in real ant environment, ants are unable to verbally communicate among themselves. Thus the authors create the ACO algorithm which mimics the pheromone trail of natural ants to let each artificial ant communicate with each other. The authors have thus created a new way which provides ACO algorithm with an environment model to live with. The pheromone value has great influence on the movement of the ant-agents in that environment.

In the transition module state, transition rule is created from the value of the pheromone trail and the distance matrices for the respective cities. We use matrix format to generate the transition rule, for example, we extract the value from city a to city b and generate the probability for the respective move. The move is based on the path of the lowest probability. In [20], the author has found out that, instead of using

one distance matrices value in the state of transition module, three distance matrices values provide better result quality.

In the review, a group of researchers [18] have combined data-mining with an ACO algorithm and created Ant-Miner.

Ant-Miner has the same concept as in ACO algorithm. Ant-Miner also has pheromone. The difference is that the pheromone trail is not used to generate any transition rule, but is used to update and evaporate the pheromone trail. In Ant-Miner design, pheromone initialization is used to start the tour, which thus adapts the concept in ACO algorithm.

A rule construction module helps to generate all the possible rules and store them into a discovered rule list and later the generated rule will compare itself with a training set that contains training case. Ant-Miner adapts a heuristic function that is used in ACO algorithm. In Ant-Miner, heuristic value is taken to be an information theoretic measure for the quality of the term to be added to the rule. In rule-pruning module, Ant-Miner uses sensitivity, specificity and accuracy measurement to prune the rules, which have specific quality. The purpose of rule-pruning module is to improve the simplicity of the rules and make the rules more understandable to end-users. The rule-pruning module removes rules one-term-at-a-time, selecting only the terms below the quality measure to be pruned, and this results in a better quality final generated rule.

In [1], Apriori algorithm generates a vast number of non-understandable rules to end-users. This is the weakness in Apriori algorithm. From the review, we find that ACO

algorithm can help to discover understandable rules and yet it ability is never been explored in Apriori algorithm. ACO algorithm can help to solve this weakness.

Another weakness in Apriori algorithm is the tedious candidates generation process. We make use of the ACO algorithm ability to find shortest path in Salesman Traveling problem, and to find the understandable rules in the data in the shortest way. With this method, ACO algorithm can replace the tedious candidates generation process in Apriori algorithm.

Chapter 3: Data Collection Analysis

3.1 Data Collection

This research is to study how the two different algorithms can work together and help each other cover their weakness. This thesis uses Skin-Burn data collected from Kuching General Hospital as a sample data.

Approval from the hospital's Deputy Dean is first sought before the information is obtained from the Record Unit Department where all the patients' case notes are kept.

The patient records are mostly not in digital form. Case-notes of patients kept in the Record Unit Department is in hand-written notes and these have to be converted into digital data for this research purpose. We have to manually go through the patients' burn cases one by one and extract useful information from the cases to make up a data. (Refer to Data Format section).

For the staff in Record Unit department, they need to trace all the skin-burn case-notes manually according to the record number. They have not sorted out the patients' case notes based on the cases. The records are kept in a general basis which is difficult and takes time for the staff to trace the cases.

Staff of Record Unit department would prepare a list of skin-burn patients' records comprising *record number*, *name*, *age*, *gender*, *cause* and the *degree of burns*. Record numbers shown beside the list would lead a searcher to trace the patient's case-note that we needs.

But some of the patients' case-notes are not in Record Unit department because they may be still with the doctors or have been sent to other department for their use.

During the data-mining process from the case-notes, the following obstacles and questions are encountered:

- Trying to understand how the case-notes have been recorded.
- Difficulty in selecting the relevant information in the case-notes needed for this research.
- Understanding how they determine degree of burn.
- Understanding how they determine percentage of burn on burnt-area.
- How do they diagnose a patient when he is first admitted to hospital. Is this information useful?
- What are the first important information pieces they record for each case and how can they useful for this thesis?
- How to identify which data are required for the testing?
- Some case-notes are more than 100 pages long, which is very time-consuming to go through and find whatever useful information.
- Many medical terms and names of medicines are very confusing and difficult to differentiate.
- Hand-written notes are difficult to read.

To convert the extracted data into digital form, a simple form (Appendix A) is designed for recording the data while studying the patients' cases one by one. The form is to contain *case-note number*; *patient's age* and *gender*; *area*, *cause* and *percentage of burn*, *degree of burn*, *description (if any)*, *medication*, *discharge information*, *dressing*,

ointment/cream, etc. The information may also include *plastic surgery*, *wound debridement*, *ambulation*, *hydrotherapy* and so forth.

The table does not need to contain information on *patients' names* and *addresses* as these are not useful information. *Case-note numbers* are required because they help in easy tracing of the individual cases to get more detailed information for each case.

In the experiment, the prototype is designed based on *cause*, *area*, *medication*, *ointment/cream* and *dressing*. The output result will base on these 5 items.

3.2 Data Format

After the manual tracing work is done, the next step is to create a database table in Microsoft Access. In the database table, all the necessary items are created based on the manual form. It includes *age, area, cause, percentage, degree of burn, description, medication, dressing, ointment/cream and others.*

The Microsoft Access database table format as in Figure 3.1. In order to store all the information that is recorded manually, it is necessary to create more fields to store each of the items separately. In item *'area'*, 4 fields are created to store the information. In item *'Medication'*, 10 fields are created, item *'Dressing'*, 5 fields are created, item *'Ointment'*; 4 fields are created, item *'Others'*; 6 fields are created (Figure 3.1).

A human body can be divided into 5 categories: *Head and Neck, Thorax, Upper Limb, Abdomen, Pelvis and Lower Limb* [7, 11 & 12].

In this sample data, the item *'area'* represents the area of a human body that is injured by burn. In the *'area'* items are *Head, Neck, Thorax, Upper Limb* and *Lower Limb*. To be more specific, *Face, Back, Foot, Mouth, Chin, Perinea, Eyes, Genital, Palm, Head, Penis, Finger, Lip, Scrotum, Knee, Ear, Anal, Scalp and Whole body* are also added into *'area'* item.

The reason to categorize the areas is that too many item *'area'* may cause confusion to the system and will not create accurate result. But some areas are categorized under *Upper Limb* or *Lower Limb*. For example, *Abdomen* in this sample data is categorized

under *Lower Limb*. Another example is *Chin* which is under *Neck* category, but in this sample-data it is an individual category.

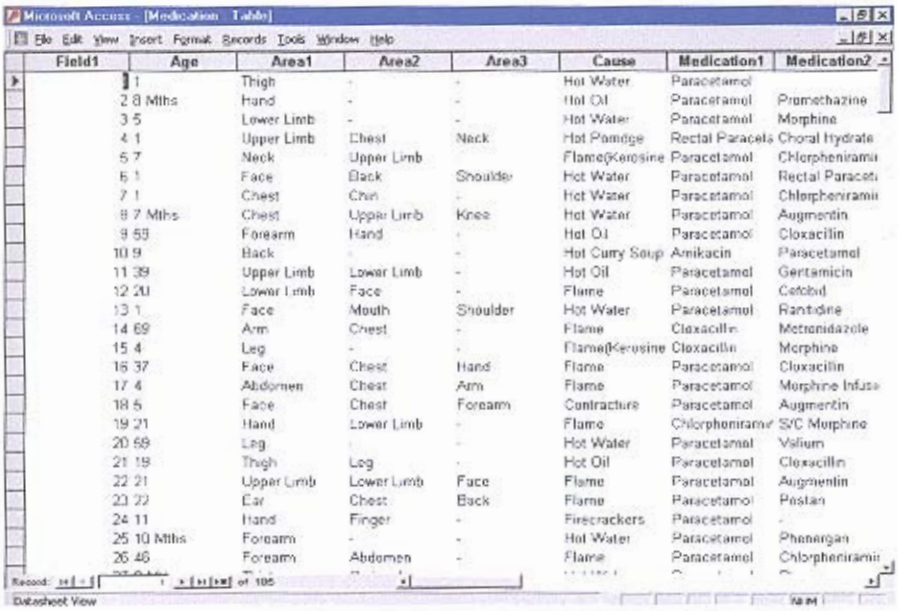
After the sample data are keyed into Microsoft Access database table, a lot of manual checking is needed before the sample data is ready for experiment.

Not all the data item in the database table will be used for experiment. First screening is carried out manually on the database file. Extract out the unnecessary data item from the file. In the extracting process, *record number*, *gender*, *discharge* and *description* is not keyed into the database table (Figure 3.2).

The design of this prototype system is to read the data from text file format. The next step is to export the database data into text file. After exporting the sample data to text file, some touch-up is required on the data in the text file before the experiment is carried out.

Field Name	Data Type
Age	Text
Area1	Text
Area2	Text
Area3	Text
Area4	Text
Cause	Text
Percentage	Text
Degree	Text
Description	Text
Medication1	Text
Medication2	Text
Medication3	Text
Medication4	Text
Medication5	Text
Medication6	Text
Medication7	Text
Medication8	Text
Medication9	Text
Medication10	Text
Dressing1	Text
Dressing2	Text
Dressing3	Text
Dressing4	Text
Dressing5	Text
Ointment1	Text
Ointment2	Text
Ointment3	Text
Ointment4	Text
Other1	Text
Other2	Text
Other3	Text
Other4	Text
Other5	Text
Other6	Text

Table 3.1: Data Structure



Field1	Age	Area1	Area2	Area3	Cause	Medication1	Medication2
1	1	Thigh	-	-	Hot Water	Paracetamol	
2	8 Mths	Hand	-	-	Hot Oil	Paracetamol	Promethazine
3	5	Lower Limb	-	-	Hot Water	Paracetamol	Morphine
4	1	Upper Limb	Chest	Neck	Hot Porridge	Rectal Paracets	Choral Hydrate
5	7	Neck	Upper Limb		Flame(Kerosine	Paracetamol	Chlorpheniramir
6	1	Face	Back	Shoulder	Hot Water	Paracetamol	Rectal Paracet
7	1	Chest	Chin	-	Hot Water	Paracetamol	Chlorpheniramir
8	7 Mths	Chest	Upper Limb	Knee	Hot Water	Paracetamol	Augmentin
9	59	Forearm	Hand	-	Hot Oil	Paracetamol	Cloxacillin
10	9	Back	-	-	Hot Curry Soup	Amikacin	Paracetamol
11	39	Upper Limb	Lower Limb	-	Hot Oil	Paracetamol	Gentamicin
12	20	Lower Limb	Face	-	Flame	Paracetamol	Cefobid
13	1	Face	Mouth	Shoulder	Hot Water	Paracetamol	Ranitidine
14	69	Arm	Chest	-	Flame	Cloxacillin	Metronidazole
15	4	Leg	-	-	Flame(Kerosine	Cloxacillin	Morphine
16	37	Face	Chest	Hand	Flame	Paracetamol	Cloxacillin
17	4	Abdomen	Chest	Arm	Flame	Paracetamol	Morphine Infus
18	5	Face	Chest	Forearm	Contracture	Paracetamol	Augmentin
19	21	Hand	Lower Limb	-	Flame	Chlorpheniramir	S/C Morphine
20	69	Leg	-	-	Hot Water	Paracetamol	Valium
21	19	Thigh	Leg	-	Hot Oil	Paracetamol	Cloxacillin
22	21	Upper Limb	Lower Limb	Face	Flame	Paracetamol	Augmentin
23	22	Ear	Chest	Back	Flame	Paracetamol	Pestan
24	11	Hand	Finger	-	Firecrackers	Paracetamol	-
25	10 Mths	Forearm	-	-	Hot Water	Paracetamol	Phenergan
26	46	Forearm	Abdomen	-	Flame	Paracetamol	Chlorpheniramir

Figure 3.1: Sample data used in this experiment in Microsoft Access print-screen format.

The prototype system is designed to read the data in a straight-line format (Figure 3.2) without any breakage in between the data in text file. A comma is uses to separate the data with each other. Empty data cannot be left empty, in this sample data, "-" is to indicate empty in the specific column. The sample-data from other sources has to go through few steps of process before it can be used for the experiment. Thus, this sample data is special for this prototype system.

"1","Lower Limb","-","-","Hot Water","4","2","Paracetamol","Phenergan","-","-","-","-","-","-","-","-","-","-","Silver Sulfadiazine","Urgotul","-","-","-","Silver Sulfadiazine Cream","-","-","-","Plastic Surgery","Hydrotherapy","Avoid Sunlight","-","-","-";"8 Mths","Hand","-","-","Hot Oil","2","2","Paracetamol","Promethazine","Cloxacillin","Rectal Paracetamol","Ampicillin","Multivitamin","-","-","-","-","-","Silver Sulfadiazine Cream","Vaseline Gauze","Jelnet","-","-","-","-","-","-","-","-","Plastic Surgery","Hydrotherapy","Tepid Sponging","-","-","-";"5","Lower Limb","-","-","Hot Water","16","1","Paracetamol","Morphine","Dormicum","Midazolan","Mist Sodium Chloride","Cefobid","Metronidazole","Mist Nacl","Cefoperazone","Ferrous Fumerate","Silver Sulfadiazine","Vaseline Gauze","Normal Saline","EOD","-","Silver Sulfadiazine Cream","Chloramphenicol","Aqueous","-","Plastic Surgery","Split Skin Graft","Wound Debridement","Hydrotherapy","Avoid Sunlight","Tepid Sponging";"1","Upper Limb","Face","-","Hot Porridge","22","1-2","Rectal Par

Figure 3.2: Sample data used in this experiment after being converted to a text-file format.

Chapter 4: Methodology

The Apriori-Ant algorithm has adapted two well-known methods in the area of mining techniques and optimization techniques. Apriori algorithm is one of the algorithms used in mining area. ACO algorithm is used in optimization area.

4.1 Apriori algorithm

The first part of this technique is to use Apriori algorithm method to generate 1-itemset and 2-itemset and at the same time to calculate the support for each set from the database.

In 1-itemset list, *SupportItem* is to store the items in the database. *SupportCnt* is to store the amount of times that the items in *SupportItem* have been occurred in the database or call frequent count (Table 4.1) (Appendix F & G).

SupportItem	***SupportCnt***
Lower Limb	53
Paracetamol	85
Phenergan	25
Silver Sulfadiazine	142
Urgotul	5
Morphine	5

Table 4.1: Sample of items with their respective count in 1-itemset list.

In 2-itemset list, it is to find all the possible combination from 1-itemset list and store only the items that exist in the database. In this technique, the items stored in 2-itemset list are arranged into x and y position in *ConfidenceItem*. *ConfidenceCnt* is to store the amount of times that the items in *ConfidenceItem* have occurred in the

database or call frequent count. It is to make it easy for later calculation in ACO algorithm section and Rule generation section (Table 4.2) (Appendix H & I).

ConfidenceItem		*ConfidenceCnt*
Position X	***Position Y***	
Lower Limb	Paracetamol	52
Lower Limb	Phenergan	13
Lower Limb	Silver Sulfadiazine	40
Lower Limb	Urgotul	3
Paracetamol	Phenergan	22

Table 4.2: Sample of items with their respective count in 2-itemset list. The table clearly indicates the items at position X and position Y.

In Apriori algorithm stage, pruning is not carried out during the generation of 1-itemset list and 2-itemset list stage. The item generation stops after 2-itemset list is generated. After generating and calculating of 1-itemset list and 2-itemset list, the process can then proceed to Apriori algorithm Computation section.

4.1.1 Apriori algorithm Computation section

In Apriori algorithm Computation section, *Lift measure* method [3 & 10] is adapted. Within the *Lift measure* method, *Support measure* and *Confidence measure* also applied.

The strength of *Lift measure* method is that it is not down-ward closed method and does not suffer from the rare item problem faced by *Confidence measure* method or other measure method.

In this technique, the original *lift measure* method has been modified.

Originally, *lift measure* [10] is:

$$\text{Lift}(x \rightarrow y)=\text{Lift}(y \rightarrow x)=P(x \text{ and } y)/(P(x)P(y))=\text{Conf}(x \rightarrow y)/\text{Supp}(y)=\text{Conf}(y \rightarrow x)/\text{Supp}(x) \quad (4)$$

Probability of *item x* and *item y* divides by the probability of *x* which is equivalent to the following equation *(5)*.

$$P(x \text{ and } y)/(P(x)P(y)) \quad (5)$$

Lift of *item x* and *item y* is the confidence value of item *x* and *y* divided by support value of y *(6)*.

$$\text{Lift}(x \rightarrow y) = \text{Conf}(x \rightarrow y)/\text{Supp}(y) \quad (6)$$

$$\downarrow$$

$$\text{Conf}(x \rightarrow y) = \text{Supp}(x \cap y)/\text{supp}(x) \quad (7)$$

And the Confidence of *item x* and *y* is equal to the support value of *x* and *y* divided by the support value of *x (7)*.

Lift of *item y* and *item x* is the confidence value of item *y* and *x* divided by support value of x *(8)*.

$$\text{Lift}(y \rightarrow x) = \text{Conf}(y \rightarrow x)/\text{Supp}(x) \quad (8)$$

$$\downarrow$$

$$\text{Conf}(y \rightarrow x) = \text{Supp}(y \cap x)/\text{supp}(y) \quad (9)$$

Confidence of item *y* and *x* is equal to the support value of *y* and *x* divided by the support value of *y (9)*.

Note: Support value of *x and y*, and, *y and x*, have no different.

$$\text{Lift}(x \rightarrow y)=\text{Lift}(y \rightarrow x) \quad (10)$$

In the *lift measure* method, Lift (x→y)=Lift (y→x) following table (Table 4.3) shows the validity of the life measure equation is correct *(10)*.

	Item X	*Item Y*	*Item X & Y*	*Conf*((x→y))	*Support Cnt*	*Lift*
Lift (x→y)	85	53	38	38/85= 0.447058823	53	0.447058823/53 = 0.008435072
Lift (y→x)	85	53	38	38/53= 0.716981132	85	0.716981132/85 =0.008435072

Table 4.3: Comparison of formula Lift (x→y) and formula Lift (y→x).

Although the *ConfidenceCnt* value is varies for each other, however the lift values are the same, thus it proves that Lift (x→y)=Lift (y→x) is valid (Table 4.3).

Modified lift measure is as below:

Lift (x→y)=(***Conf***(x→y)/supp(y))*supp(x) *(11)*

From the original lift measure, the calculation is confidence count of item *x* & *y* divided by the frequency count of item y. Frequent count of item *x* & *y* is the amount of item *x* & *y* that occurs in the database. In the modified lift measure, it retains the original formula, the modification part is to multiply the original formula to frequent count of item x. It is to make sure the values are accurate *(11)*.

The *(x→y)* is to present *item x* and *item y*. But in this example *x* and *y* denote the position in 2-itemset list that is generated in Apriori algorithm. You will find *xCount* and *yCount* variables in later section. They are the frequent count for the item in *x position* and *y position* individually in 2-itemset list.

xCount and ***yCount*** formula:

xCount = SupportCnt[loop]/Total - if ConfidenceItem[loop][x]=SupportItem[loop] *(12)*

yCount = SupportCnt[loop]/Total - if ConfidenceItem[loop][y]=SupportItem[loop] *(13)*

Example from the formula above *(12 & 13)*, ***xCount*** is calculated by *SupportCnt* divided by Total. *SupportCnt[loop]* is the frequent count in 1-item set in the array, *'Total'* is the amount of data that is used in this experiment *(12 & 13)*.

ConfidenceItem[loop][x] is a two-dimensional array which stores the combined item from 1-itemset list in *position x*. *SupportItem[loop]* represents the item in 1-itemset list in the form of one-dimensional array *(12 & 13)*.

ConfidenceItem		*ConfidenceCnt*	*xCount*	*yCount*
Position X	*Position Y*			
Lower Limb	Paracetamol	52	53/90=0.588889	85/90=0.944444

Table 4.4: xCount and yCount value for Lower Limb and Paracetamol.

SupportItem	*SupportCnt*
Lower Limb	53
Paracetamol	85

Table 4.5: Lower Limb and Paracetamol in 1-itemset list and their counts.

From the example above, ***xCount*** for *ConfidenceItem 'Lower Limb & Paracetamol'* is 53/90 = 0.588889 (9) (Table 4.4) as the position x in *'ConfidenceItem[loop][x]'* is *Lower Limb*. *SupportCnt* for *Lower Limb* is 53 (Table 4.5).

yCount for *ConfidenceItem 'Lower Limb & Paracetamol'* is 85/90=0.944444 *(13)* (Table 4.4). Position y in *'ConfidenceItem[loop][y]'* is *Paracetamol*. *SupportCnt* for *Paracetamol* is 85 (Table 4.5).

To calculate *Lift measure*, ***ProConf***=*ConfidenceCnt*/total (14) is applied. *ConfidenceCnt* is the probability of count for item x and item y that occur together in the database (Table 4.2).

ProConf=ConfidenceCnt/total *(14)*

From the example above, the support of *ConfidenceItem 'Lower Limb & Paracetamol'* in 2-item set is 52 (Table 4.2) *'Total'* is the amount of data that use in this experiment. The value of *ProConf* is 0.577778 (15).

ProConf = 52/90

= 0.577778 *(15)*

After ***ProConf*** is calculated, next is to calculate the confidence value of *item x* and *item y* by using *Confidence measure* method.

Confidence measure is as below:

$Conf(x \rightarrow y)=P(y \mid x)=P(x \text{ and } y)/P(x) = Supp(x \rightarrow y)/Supp(x)$ *(16)*

$P(y \mid x)=P(x \text{ and } y)/P(x)$ *(17)*

Probability of *item x* and *item y* divided by the probability of *item x*, which is equivalent to *(18)*.

$Supp(x \rightarrow y)/Supp(x)$ *(18)*

Support of item x and item y divided by the support of item x.

Simplified method:

$Conf(x \rightarrow y)= Supp(x \rightarrow y)/Supp(x)$ *(19)*

Supp(x→y) is the amount of count for item x and item y in 2-itemset list and *Supp(x)* is the probability for x in 1-item set.

Conf=ProConf/xCount *(20)*

In order to get *Conf(x→y)*, ***Conf=ProConf/xCount*** is applied *(19)*. ***xCount*** is the support count in which the *SupportItem* matches with the item in position x in 2-itemset list *(12)*.

For example,

ConfidenceItem 'Lower Limb & Paracetamol' in 2-item set. Value of ***ProConf*** is 0.577778 *(14 & 15)*, the item in *x position* in this example is *'Lower Limb'* and the support count for ***xCount*** for *'Lower Limb'* is 53/90=0.588889 *(12)*.

Conf = 0.577778/0.588889

= 0.981132 *(21)*

lift = (Conf/yCount)*xCount *(22)*

In lift measure, lift = (***Conf/yCount***)*****xCount*** *(22)*. ***xCount*** is the support count for 2-itemset list in *position x* and ***yCount*** is the support count for 2-itemset list in *position y* which have described in Apriori algorithm section *(12 & 13)*.

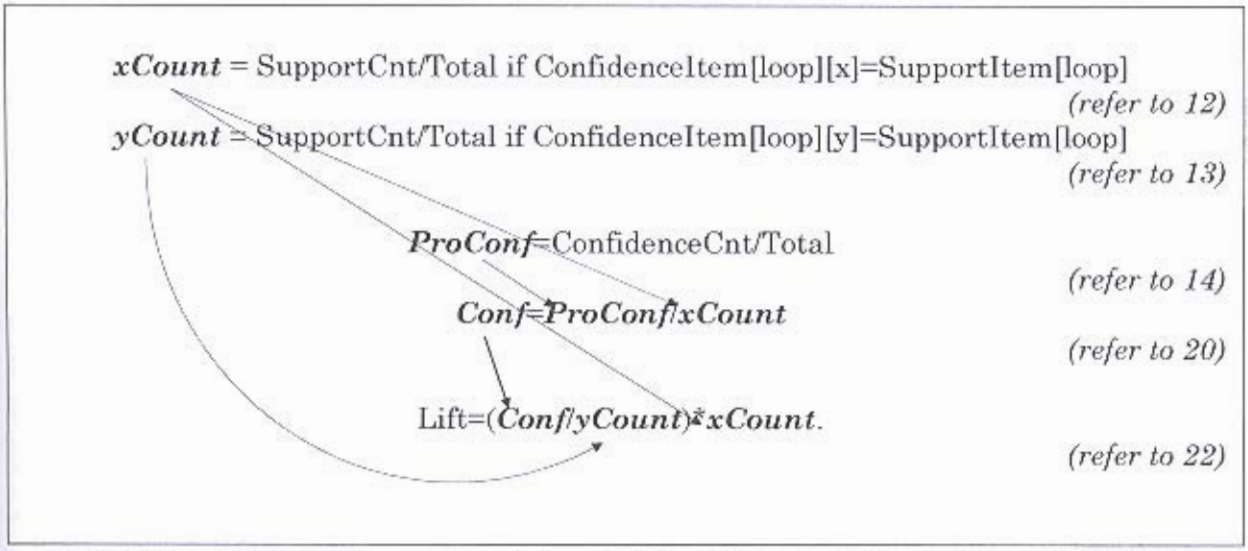


Illustration 1: The relation between formulas for xCount, yCount, ProConf, Conf and Lift measure method.

From the example of 2-item set *'Lower Limb & Paracetamol'*. Position x is *'Lower Limb'* and position y is *'Paracetomal'*, their support count is 53/90 and 85/90 respectively stored in ***xCount*** *(12)* and ***yCount*** *(13)*. We substitute the values of ***Conf*** *(20)*, ***yCount*** *(13)* and ***xCount*** *(12)* into the lift measure formula.

Lift = (0.981132/0.944444)*0.588889

= 0.611765 *(23)*

4.2 Ants Colony Optimization algorithm

Ants Colony Optimization algorithm substitutes the values from Apriori algorithm Computation section into Pheromone calculation like finding the *probability of Pheromone, Pheromone Update* and *Pheromone Evaporation* (Appendix J).

There are two different uses of Pheromone: Exploration and Exploitation. Exploration is a stochastic process in which the choice of the component used to construct a solution to the problem is made in a probabilistic way. Exploitation chooses the component that maximizes a blend of pheromone trail values and partial objective function evaluations.

4.2.1 Pheromone calculation

In Pheromone Calculation, pheromone probability calculation is carried out. In this design, the pheromone probability calculation is divided into two parts. First is to calculate amount of pheromone preference and then follow by calculating the probability of the pheromone. In algorithm, probability of the pheromone is used to decide which item is joined into the rule set and which item is not.

Pheromone formula(1):

Pher = (Pheromone Matrix*α)/(Distance Matrix*β) *(24)*

Pher = (0.577778*.05)/((0.611765+0.944444)*0.8)

= 0.288889/(1.556209)*.08

= 0.288889/1.2449672

= 0.232045 *(25)*

The Pheromone value is proportional to the desirability of having particular assignment in the solution. It is to show the connections between two nodes. Distance is included to enhance the algorithm efficiency. Tuning constants α and β is important as it govern the relative importance of pheromone concentration vs distance. Dorigo et. al(1996) [9] refers α & β as 'desirability measure', deserving the degree of importance of smell and sight in influencing pseudant behavior.

Setting on the α parameter or the pseudant's behavioral sensitivity to the pheromone is also significant. If the setting is too low, the pseudants would fail to heed the preferable pathways, and the system converges on a desirable solution. Set too high, the pseudants rely too much on the pheromone, and in so doing, would fail to adequately explore uncharted territory.

If the β is set too low (a highly volatile pheromone), the algorithm is less efficient. A highly volatile pheromone would potentially result in premature 'forgetting' of optimal pathways, resulting in a time delay of finding the optimal solution.

In pheromone formula*(1)*, we use ***ProConf*** *(14)* and ***lift*** *(23)*+***yCount*** *(13)* to replace Pheromone and Distance in the original formula of Pheromone in ACO algorithm. ***Pher*** = (***ProConf****.05)/((***lift***+***yCount***)*.08), ***ProConf*** denoted as pheromone and ***lift***+***yCount*** denoted as the distance. In this experiment, the α is set to 0.5 and β is set to 0.8 to test the data. Setting α to 0.5 is to balance the pheromone value in the experiment. If setting is too high, it may rely too much on pheromone value. If too low, it may fail to heed the pathways. Setting β to 0.8 is to avoid highly volatile pheromone happening in the experiment *(24 & 25)*.

ProConf (14) is the Confidence value of two items in the experiment. It is calculated by using Confidence measure in Apriori algorithm. In nature, Confidence value denotes the conditional probability of the head of the association rule. It represents the power of relationship of two items to the third one which may join in to form a rule.

In this point, ***ProConf (14)*** serves the same purpose as in Pheromone in ACO algorithm. Instead of denoting the conditional probability of the head of the association rule, it shows the relationship of the connection of two items in the process.

Lift+yCount denotes Distance in ACO algorithm. Distance is used to enhance the algorithm efficiency. ***Lift (23)*** value is to measure how many times more often the two items occur together than expected if it were statistically independent. ***Lift (23)*** is to evaluate the quality of an association rule. It shares the same concept in ACO algorithm. ***Lift (23)*** can help to enhance the efficiency in the algorithm.

But ***Lift (23)*** value alone is not enough. ***yCount (13)*** is the frequent count for the item in position y in 2-itemset list. The reason to add ***yCount (13)*** in this experiment is that item in position y has strong influence. It indicates which item in position y in 2-itemset list is the best match with item in position x. With the highest value, the joint item in 2-item set will be chosen for the next process or until no more match is found.

Lift+yCount value provides more accurate value as compare to a stand-alone ***Lift*** value. ***Lift+yCount*** value also provides stronger relationship for the two joint items in the 2-itemset list.

Pheromone formula(2):

Pheromone=(Pher/TotalPher)*100 *(26)*

Pheromone = (0.232045/47.3862)*100

= (0.00489689)*100

= 0.489689 *(27)*

Pheromone formula*(2)* is to calculate the probability of pheromone. *(26)* The formula is rather easy: divide the ***Pher*** *(24 & 25)* by the total pheromone preference value and multiply by 100%.

In this case, the probability pheromone value is not to be used for starting up a tour but to generate rule with the heaviest pheromone value.

The Pheromone and Distance values are arranged in the form of Matrix. But in Apriori-Ant algorithm, these two denoted values are not arranged in Matrix form. The Pheromone and Distance values are used to calculate the Probability of the Pheromone for the items in 2-itemset list.

4.3 Generating the rule

The idea of ACO algorithm in this experiment is to find the shortest path. This algorithm adapts the idea of ACO algorithm and uses algorithm to generate rule in the shortest time.

ACO algorithm starts the tour through a random search in the database (Appendix K & M). This algorithm adapts the same idea which starts the random search in 1-item set instead of 2-itemset list. 1-itemset list consists of *SupportItem* and *SupportCnt* (Table 4.1). The random search is based on *SupportItem* items. *SupportItem* item serves as the starting point in the rule generation process. If we set the test in 100 times, 100 random searches will be carried out in 1-item set.

To start the tour, *SupportItem* item is first used to find the match in *ConfidenceItem* set in 2-itemset list, and only the highest pheromone value is chosen to form the rule. In *ConfidenceItem* set, the item in position x and y will carry out the rest of responsibility in forming a rule. Item in position y will help to match in position x in next rule generation process until no more matches are found and only the item with highest pheromone value will join into the rule set. This process repeats until no more matches are found.

Example:

In the first random search, *SupportItem "Cloxacillin"* is selected. The next process is to match any *ConfidenceItem* in 2-itemset list in position x with only *"Cloxacillin"* item. And only the highest pheromone value is selected for rule joining and matching process.

SupportItem : Cloxacillin

ConfidenceItem

Position x	Position y	Pheromone
Cloxacillin	Hydrocolloid	0.0996442
Cloxacillin	Vaseline Gauze	0.0922068
Cloxacillin	*Silver Sulfadiazine*	*0.605617*
Cloxacillin	Silver Sulfadiazine Cream	0.113849

Rule: Cloxacillin, Silver Sulfadiazine

Figure 4.1: First cycle of rule generation. The randomly selected item is Cloxacillin. Silver Sulfadiazine is selected because it has the highest pheromone value in the list. Silver Sulfadiazine is added to the rule list and is used for the second cycle of rule generation.

In this process (Figure 4.1), we have 2 items added into the rule set - *Cloxacillin and Silver Sulfadiazine*. In this step, 1-itemset list is done and we use the items in 2-itemset list in position y to find the next item to join into the rule set. From the example above, the item for the next process is *'Silver Sulfadiazine'* in position y. *'Silver Sulfadiazine'* serves the purpose as in the previous process to match the items in 2-itemset list in position x and only with the highest pheromone value is selected for the rule joining and matching process. This process will repeat until no further item is found in the database.

Position y: Silver Sulfadiazine

ConfidenceItem

Position x	Position y	Pheromone
Silver Sulfadiazine	*Silver Sulfadiazine Cream*	*2.78536*
Silver Sulfadiazine	Vaseline Gauze	0.111752
Silver Sulfadiazine	Normal Saline	0.099594
Silver Sulfadiazine	Hydrocolloid	0.133442

Rule: Cloxacillin, Silver Sulfadiazine, Silver Sulfadiazine Cream,

Figure 4.2: Second cycle of rule generation.

In this process (Figure 4.2), '*Silver Sulfadiazine & Silver Sulfadiazine Cream*' with the highest pheromone is selected. '*Silver Sulfadiazine Cream*' is added into the rule set and it will be used for the next matching process to find the rule.

Position y: Silver Sulfadiazine Cream

ConfidenceItem

Position x	Position y	Pheromone
Silver Sulfadiazine Cream	Aqueous	0.0577123
Silver Sulfadiazine Cream	Liquid Paraffin	0.0414607
Silver Sulfadiazine Cream	*Chloramphenicol*	*44.6859*

Rule: Cloxacillin, Silver Sulfadiazine, Silver Sulfadiazine Cream, Chloramphenicol,

Figure 4.3: Third cycle of rule generation.

In this process (Figure 4.3), '*Silver Sulfadiazine Cream & Chloramphenicol*' with the highest pheromone is selected. '*Chloramphenicol*' is added into the rule set and it will be used for the next matching process to find the rule.

Position y: Chloramphenicol

ConfidenceItem

Position x	Position y	Pheromone
Chloramphenicol	Liquid Paraffin	0.0344572
Chloramphenicol	Hydrocortisone	0.00923852
Chloramphenicol	*Aqueous*	*326.734*
Chloramphenicol	Intrasite	0.0330718

Rule: Cloxacillin, Silver Sulfadiazine, Silver Sulfadiazine Cream, Chloramphenicol, Aqueous,

Figure 4.4: Fourth cycle of rule generation.

In this process (Figure 4.4), '*Chloramphenicol & Aqueous*' with the highest pheromone is selected. '*Aqueous*' is added into the rule set and it will be used for the next matching process to find the rule.

ConfidenceItem

Position x	Position y	Pheromone
Aqueous	*Liquid Paraffin*	*3189.07*

Rule: Cloxacillin, Silver Sulfadiazine, Silver Sulfadiazine Cream, Chloramphenicol, Aqueous, Liquid Paraffin

Figure 4.5: Fifth cycle of rule generation.

In this process (Figure 4.5), '*Aqueous & Liquid Paraffin*' with the highest pheromone is selected. '*Liquid Paraffin*' is added into the rule set and it will be used for the next matching process to find the rule.

Position y: Liquid Paraffin

Figure 4.6: Sixth cycle of rule generation.

No more match for *Liquid Paraffin* (Figure 4.6), the matching process will stops here. The rule for this cycle is '***Cloxacillin, Silver Sulfadiazine, Silver Sulfadiazine Cream, Chloramphenicol, Aqueous, Liquid Paraffin***'.

Illustration 2 below is to simplify the rule generating process. In each cycle, the same item in 1-itemset list may be selected in the process.

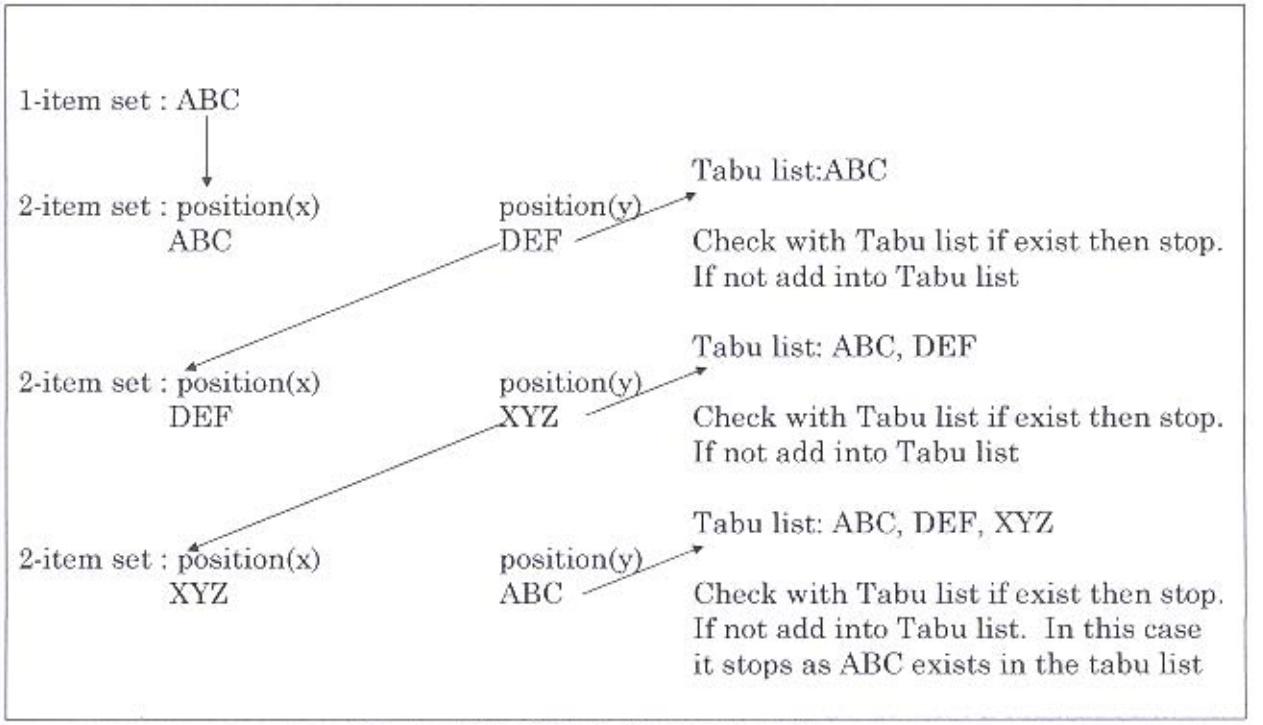


Illustration 2: Rule Generation Process

In each process you may find that the pheromone values are different. In order to make accurate result, *Pheromone Update* and *Pheromone Evaporation* also applied in ACO algorithm. But the formula for *Updating* and *Evaporating* is much simpler than that in ACO algorithm.

Original pheromone update formula as follow:

$$pher_{ij}(t+1) = \lambda pher_{ij}(t) + \sum_{k=1}^{N} \Delta^{k}_{ij} \qquad (28)$$

$$\Delta^{k}_{ij} = \begin{cases} \dfrac{Q}{\text{Length}_{O^k}} \end{cases} \qquad (29)$$

Modified Pheromone Update formula:

(0.04*pheromone-1)+pheromone+(*SupportCnt**0.001) *(30)*

There are two modifications in the original formula used in ACO algorithm. The first modification is that, instead to get 100% from previous pheromone, we get 0.04% of the previous pheromone. Second modification is, we get the 0.001% from the *SupportCnt* value (Figure 3.1). *SupportCnt* value is the frequent value for the item in position y in 2-itemset list. In this experiment, *SupportCnt* value automatically refers to item in position y in 2-itemset list.

Pheromone evaporates over time, preventing levels becoming unbounded, and allowing the ant colony to 'forget' old information. In the formula below, pheromone value is reduces on each edge once per cycle. p is the evaporation rate, where $0 < p < 1$.

Original pheromone evaporation formula as follow:

$$pher_{ij}(t) = (1\text{-}p)^* pher_{ij}(t) \qquad (31)$$

Pheromone Evaporation formula:

$$\text{Pheromone Evaporation} = \text{Pheromone-}(0.001^*\text{Pheromone}) \qquad (32)$$

In the '***Pheromone Evaporation formula***', the formula far more easy as compare to original formula. In each cycle, the pheromone will be reduced by 0.001%.

To avoid searching process repeating the items that have been visited, Tabu memory is used. It keeps track on the items that have been visited. Another use for Tabu memory is for starting a search which is called Tabu Search. Tabu Search is an iterative local search metaheuristic. But we do not adapt this idea in this algorithm.

Chapter 5: Experimental Result

At least 366 MHZ computer speed is required to run the prototype system. The required software is Microsoft Visual C++ version 5 and above. The experiment is done using a 366 MHZ computer speed with Microsoft Visual C++ version 5.

In Apriori algorithm, transactional data is used, but a non-transactional data is used to test Apriori-Ant algorithm. In Apriori-Ant algorithm, the process is divided into 2 steps. The first step, as conducted in Apriori algorithm, is to generate first itemset and second itemset. In the experiment, the data set is separated according to cases; which is different from Apriori algorithm where data used in the experiment are not segmented.

From the data set, records are extracted based on 8 different cases to form their own sample data. The 8 different cases are '*Hot Water*', '*Flame*', '*Hot Oil*', '*Firecrackers*', '*Electrical*', '*Hot Soup*', '*Hot Porridge*', and '*Hot Air*'. In each sample-data, the number of records varies from each other.

The reason to separate cases data into different segments is so that, later, when the rules are generated by the Apriori-Ant algorithm, they are already based on their own cases. Thus, the rules can easily be used in Expert System instead of creating a system to extract the individual rules based on the cases.

In the experiment, 1000 cycles in pheromone section and 20 rules generation is carried out in the process.

Table 5.1 shows the result of the experiment in 8 different sample-data: they are *'Hot Water'*, *'Flame'*, *'Hot Oil'*, *'Firecrackers'*, *'Electrical'*, *'Hot Soup'*, *'Hot Porridge'* and *'Hot Air'*. Each sample-data is arranged in ascending order according to the ***'Number of records'***.

In the research, the experiment takes place in 6 areas: they are ***'Number of Records'***, ***'Total Pheromone value (1st cycle)'***, ***'Processing speed (minutes, seconds)'***, ***'Number of item in 1-itemset list'***, ***'Number of item in 2-itemset list'*** and ***'Number of rules'***.

Sample of Data	Hot Water	Flame	Hot Oil	Firecrackers	Electrical	Hot Soup	Hot Porridge	Hot Air
Number of Records	90	33	15	5	4	2	1	1
Total Pheromone Value (1st Cycle)	47.3862	70.686	42.5054	19.9161	43.9192	6.875	59.375	1.875
20 rules	3 minutes 14.73 seconds	2 minutes 22.79 seconds	1 minutes 01.38 seconds	19.23 seconds	36.28 seconds	6.87 seconds	24.07 seconds	5.01 seconds
100 rules	3 minutes 22.33 seconds	2 minutes 31.13 seconds	1 minutes 05.49 seconds	20.53 seconds	38.56 seconds	7.04 seconds	24.79 seconds	5.39 seconds
500 rules	4 minutes 13.33 seconds	3 minutes 14.40 seconds	1 minutes 23.88 seconds	21.63 seconds	42.93 seconds	7.13 seconds	26.86 seconds	5.45 seconds
1000 rules	5 minutes 16.13 seconds	4 minutes 05.75 seconds	1 minutes 31.35 seconds	22.33 seconds	43.83 seconds	7.23 seconds	27.33 seconds	5.51 seconds
Number of rules.	1668	1357	597	167	312	30	190	6

Table 5.1: Comparison for 8 different sample data in 6 areas.

From Table 5.1, it clearly shows that the number of rules is affected by the pheromone values, and by the number of items in 1-itemset list as well as 2-itemset list. The number of records has little effect on the number of rules generated.

For example, in the *'Hot Porridge'* sample data, only one record is being tested in this experiment. But one record can produce more than 100 useful rules as compared to *'Hot Air'* which is also with one record. This is to prove that the number of records has no effect on the number of rules generated, but the number of items in 1-itemset list and 2-itemset list does. From example in *'Hot Porridge'* and *'Hot Air'*, the number of items in 1-itemset list and 2-itemset list of *'Hot Porridge'* is more than *'Hot Air'*. It will be further explained in later section.

In this experiment, we focus on the following two areas:

- Processing Time.
- Rule generation.

5.1 Experimental: Processing Time

One of the objectives to design Apriori-Ant algorithm is to reduce candidates generation process which also reduces processing time.

In this research, an experiment is done on the processing time but its focus is not to test if Apriori-Ant algorithm has better efficient processing time as compared to other Algorithm such as Apriori algorithm. The experiment is to find out what is that which effects processing time in Apriori-Ant algorithm, and how to enhance the processing time.

Research in this section reveals that processing time is firstly influenced by the total pheromone value, and secondly is influenced the number of records. And these two factors have connection to each other. This will be further explained in the following section.

5.1.1 Total Pheromone Value and Number of Records

The '***Total Pheromone value***' is the total pheromone value of items in 2-itemset list generated in the first processing cycle. The '***Total Pheromone value***' would increase after each cycle.

In the example of *'Hot Water'* sample-data, 1668 items are generated and stored in 2-itemset list (Table 5.2).

Sample of Data	Hot Water	Flame	Hot Oil	Fire-crackers	Electrical	Hot Soup	Hot Porridge	Hot Air
Number of Records	90	33	15	5	4	2	1	1
Total Pheromone Value (1st Cycle)	47.3862	70.686	42.5054	19.9161	43.9192	6.875	59.375	1.875
Processing Speed (minutes, seconds)	3 minutes 14.73 seconds	2 minutes 22.79 seconds	1 minutes 01.38 seconds	19.23 seconds	36.28 seconds	6.87 seconds	24.07 seconds	5.01 seconds
Number of items in 1-itemset list	118	90	56	24	36	9	21	4
Number of items in 2-itemset list	1668	1357	597	167	312	30	190	6
Number of rules.	<1000	<1000	>600	>170	>320	>40	>200	6

Table 5.2: Comparison for 8 different sample data in 6 areas. (Emphasize is on Hot Water sample data)

The ***'Total Pheromone value'*** indicates the number of items generated in 2-itemset list. It also indicates the number of rules that would be generated during the processing. In other words, the items in 2-itemset list also affects the number of rules generated during the processing. Higher pheromone value would produce more rules while lower pheromone value would produce fewer rules. This will be explained under '**Experiment: Rule generation**".

Table 5.2 shows how ***'Total Pheromone value'*** affects the number of items generated in 2-itemset list. *'Hot Porridge'* has higher pheromone than that of *'Hot Air'* although both of them have only one record for processing. Note the number of items generated in 1-itemset list and 2-itemset list. Table 18 shows that *'Hot Porridge'* with higher pheromone value creates more items in 1-itemset list and 2-itemset list as compared to *'Hot Air'*.

Sample of Data	Hot Porridge	Hot Air
Number of Records	1	1
Total Pheromone Value (1st Cycle)	59.375	1.875
Processing Speed (minutes, seconds)	24.07 seconds	5.01 seconds
Number of items in 1-itemset list	21	4
Number of items in 2-itemset list	190	6
Number of rules.	>200	6

Table 5.3: Comparison between Hot Porridge and Hot Air sample data.

Table 5.3 shows that processing speed is affected by ***'Total Pheromone value'*** if both have the same number of records for processing. Processing speed for *'Hot Porridge'* is **24.07 seconds** as compared to *'Hot Air'* which is only **5.01 seconds**.

The above finding proves that the higher it is the ***'Total Pheromone value'***, the more items in 1-itemset list and 2-itemset list are generated and thus more processing speed is needed (Table 5.3).

Another example: If the ***'Total Pheromone value'*** of two cases are almost equal to each other, then the ***'Number of record'*** would be considered. The higher the ***'Number of record'*** is, the more items are generated in 1-itemset list and 2-itemset list (Table 5.4).

Sample of Data	Firecrackers	Electrical
Number of Records	5	4
Total Pheromone Value (1st Cycle)	19.9161	43.9192
Processing Speed (minutes, seconds)	19.23 seconds	36.28 seconds
Number of items in 1-itemset list	24	36
Number of items in 2-itemset list	167	312
Number of rules.	>170	>320

Table 5.4: Comparison between Firecrackers and Electrical sample data.

In Table 5.4, *'Firecrackers'* and *'Electrical'* sample data are tested. The ***'Total Pheromone value'*** for these two sample data is **19.9161** and **43.9192** respectively. And the number of records of these two sample data is 5 and 4 respectively (Table 5.4).

This example shows that, although *'Firecrackers'* sample data has more data than *'Electrical'* sample data, it does not mean that *'Firecrackers'* sample data will have more '***Total Pheromone value***' than *'Electrical'* sample data. The amount of '***Total Pheromone value***' is based on the items in 1-itemset list and 2-itemset list. This experiment reveals that more items generated both in the list would produce more '***Total Pheromone value***'.

No experiment is needed to prove that more records require more processing time. To improve processing time, it is necessary to reduce pheromone value because pheromone value has an effect on processing time.

The experiment shows that more pheromone value requires more processing time regardless of the number of items generated or the number of rules generated. But pheromone value is the one that determines which item is to be joined into the rule list. Setting pheromone too low affects quality of generated rules. Thus, it is important to find an effective formula which can reduce processing time and at the same time retain the quality of generated rules.

5.2 Experimental: Rule Generation

In rule generation experiment, 2 areas are tested:

- What influences the number of rules generated?
- How rules are generated.

The objective of this experiment is to find out how to improve the number of rules generated and how to enhance the rule generation process.

5.2.1 What influences the number of rules generated?

In this section, the research reveals factors that influence the number or rules generated in Apriori-Ant algorithm. There is a relation in the number of items in 1-itemset list and the number of items in 2-itemset list.

No	*Sample Data*	*Number of Record*	*Number of items in 1-itemset list*	*Number of items in 2-itemset list*	*Number of rules*
1	**Hot Porridge**	1	21	190	>200
2	**Hot Air**	1	4	6	>10

Table 5.5: Comparison of Hot Porridge and Hot air sample data under 'Number of Record', 'Number of items in 1-itemset list', 'Number of items in 2-itemset list', and 'Number of rules'.

In Table 5.5, it shows the relationship between '***Total Pheromone value***', the '***Number of Items in 1-itemset list***', the '***Number of Items in 2-itemset list***', and the '***Number of Rules***' generated in the process.

First, we look at the relationship between '***Total Pheromone value***' and '***Number of items in 1-itemset list***'. More pheromone value produced more items in 1-itemset list, but this hypothesis does not apply to all cases. Take '*Flame*' sample-data and '*Hot*

Water' sample-data as an example, '*Flame*' sample-data produce more pheromone value than '*Hot Water*', but it does not generate more items in 1-itemset list. In this example, '***Number of records***' has influence on the number of items in 1-itemset list.

And second is the ***'Number of items in 1-itemset list'*** and the ***'Number of items in 2-itemset list'*** have great influence on the number of rules generated. The highest total pheromone value would create more items in 2-itemset list. In other words, highest ***'Total Pheromone value'*** would produce more rules as compared to the lower ***'Total Pheromone value'***.

The reason why the highest pheromone value produces more rules is that the number of items in 2-itemset list with highest pheromone value are more as compared to lowest pheromone value.

In the experiment, sample-data *'Hot Porridge'* and *'Hot Air'* are used. Both of these two sample-data are with 1 record. But the rule produced by the system for these two sample-data are different. *'Hot Porridge'* can produce more than 100 useful rules but is less than 10 useful rules for *'Hot Air'*.

Table 5.5 explains why *'Hot Porridge'* produces more rules than *'Hot Air'*. The number of items for *'Hot Porridge'* in 1-itemset list is 21 while the number of items in 2-itemset list is 190. From this relationship, you can see that the largest ***'Number of items in 1-itemset list'*** will produce more items in 2-itemset list. Items in 2-itemset list are to store the possible combination of items in 1-itemset list.

The process is based on the hypothesis of probability in mathematic formula. The probability of the combination of 4 items in 1-itemset list is 6 items in 2-itemset list (Table 5.6).

1-itemset list	*2-itemset list*
A	A&B, A&C, A&D
B	B&C, B&D
C	C&D
D	

Table 5.6: Example of 2-itemset list generation based on 4 items; A, B, C, and D.

However, the probability of the combination for 5 items in 1-itemset is 10 items and would be generated in 2-itemset list. For example in Table 5.7.

1-itemset list	*2-itemset list*
A	A&B, A&C, A&D, A&E
B	B&C, B&D, B&E
C	C&D, C&E
D	D&E
E	

Table 5.7: Example of 2-itemset list generation based on 5 items; A, B, C, D, and E.

In this case, it indirectly indicates that the more items in 1-itemset list, the more rules are generated. The number of items in 1-itemset list can be used to predict the number of rules to be generated. The items in 2-itemset list is the possible combination in 1-itemset list. The items in 1-itemset list is not repeated from each other. So the number of items in 1-itemset has great effect in 2-itemset list.

The reason why the number of items in 1-itemset list can be used to indicate the number of rules is that, the more items in 1-itemset list that are generated, the more items are generated in 2-itemset list which can be used to form a rule that is why the highest

pheromone value. The less number of items in 1-itemset list, the less items are generated in 2-itemset list that can be joined to form a rule.

Indirectly the number of items in 1-itemset list can be used to predict the number of rules generated, but the direct affect is the items in 2-itemset list. The reason is that items in 2-itemset list are the possible combination of items in 1-itemset list. In the algorithm, rule generation is based on the items in 2-itemset list. In other words, the number of items in 2-itemset list is the number of rules to be generated in the process.

The less items in 2-itemset list the less rules are generated. In hypothesis, 190 items in 2-itemset list would produce 190 rules and 6 items in 2-itemset list would only produce 6 rules.

As a result of this experiment, the rule generation process has a relation with the ***Total Pheromone value***, ***Number of items in 1-itemset list*** and ***Number of items in 2-itemset list***. The value of pheromone would influence the number of items in 1-itemset list. More items in 1-itemset list would also create more items in 2-itemset list. And more items in 2-itemset list would generate more rules in the end of the process. It can thus be assumed that the number of items in 2-itemset list is related to the number of rules generated in the process.

To prove that the hypothesis is correct, we carry out another experiment on rule generation by algorithm. Experiment is carried out in 3 selected sample data, *'Hot Porridge'*, *'Hot Air'* and *'Hot Soup'*.

No	*Sample Data*	*Number of Record*	*Number of items in 1-itemset list*	*Number of items in 2-itemset list*	*Number of rules*
1	**Hot Porridge**	1	21	190	190
2	**Hot Air**	1	4	6	6
3	**Hot Soup**	2	9	30	30

Table 5.8: Shows the relationship between 'Number of items in 2-itemset list' and 'Number of rules generated'.

Table 5.8 clearly shows that the ***'Number of items in 2-itemset list'*** is the same as in the ***'Number of rules'***. It thus proves that the hypothesis is correct as the ***'Number of items in 2-itemset list'*** indicates the ***'Number of rules'*** generated. This is because the rule generation process is based on the items in 2-itemset list.

Checking whether the hypothesis is correct is by looking at the rule list and the 2-itemset list with the pheromone value. *'Hot Air'* sample data is tested.

The first rule for *'Hot Air'* is ***[1 Wet Gauze, Chloramphenicol]*** as it contains the highest pheromone value in the process that is 614.568 after 1000 cycles of *pheromone update* and *pheromone evaporation* process takes place (Figure 5.1 & Figure 5.2).

1 Wet Gauze, Chloramphenicol
2 Paracetamol, Wet Gauze, Chloramphenicol
3 Face, Chloramphenicol
4 Face, Paracetamol, Wet Gauze, Chloramphenicol
5 Paracetamol, Chloramphenicol
6 Face, Wet Gauze, Chloramphenicol

Figure 5.1: 2-itemset list without pheromone values for **Hot Air** sample data.

```
0 | Wet Gauze|,| Chloramphenicol|,|614.568
1 | Paracetamol|,| Wet Gauze|,|51.4225
2 | Face|,| Chloramphenicol|,|4.53017
3 | Face|,| Paracetamol|,|4.53017
4 | Paracetamol|,| Chloramphenicol|,|4.53017
5 | Face|,| Wet Gauze|,|4.53017
```

Figure 5.2: 2-itemset list with pheromone values for **Hot Air** sample data.

The second highest pheromone value is *'Paracetamol', 'Wet Gauze'* with 51.4225 pheromone value. The rule has been formed starting from *'Paracetamol', 'Wet Gauze'*. The next step is to find which item combines with *'Wet Gauze'* will produce higher pheromone value. In this experiment, *'Wet Gauze', 'Chloramphenicol'* produces highest pheromone value in the 2-itemset list. Then the process is continued to find out which item combines with *'Chloramphenicol'* will produce higher pheromone value, but the process stops here because there is no more match for *'Chloramphenicol'*. In this experiment, ***[2 Paracetamol, Wet Gauze, Chloramphenicol]*** rule is formed (Please refer to Chapter 4-Methodology for rule generation) (Figure 5.2).

Another proves shown by the Figure 5.2 is, the number of record has no influence on the number of rules generated. The sample data *'Hot Soup'* has 2 records for the process, the rule generated is less than *'Hot Porridge'* that has 1 record for the process. Such result is because the number of items generated in 1-itemset list and 2-itemset list is less as compared to *'Hot Porridge'*. But the processing time for *'Hot Soup'* is less than *'Hot Porridge'*.

5.2.2 How rules are generated.

This experiment reveals that rules are generated based on the total pheromone value after 1000 cycles of pheromone update and evaporation process. And in this experiment, 20 rules are generated in the process. Users can define the number of rules they need in the process. Rules generated in this step have no meaning to the user and hardly to understand. Rules generated only meaning and useful in an Expert System.

A sample data - *'Flame'* is selected to show how rules in this algorithm are generated.

In the above section, it can be seen that the number of items in 2-itemset list is the number of generated rules.

First example: Rule list: **Flame** (20 rules)

1 Finger, Silver Sulfadiazine, Vaseline Gauze, Silver Sulfadiazine Cream, Chloramphenicol, Aqueous
2 Lower Limb, Paracetamol, Vaseline Gauze, Silver Sulfadiazine Cream, Chloramphenicol, Aqueous
3 Lower Limb, Vaseline Gauze, Silver Sulfadiazine Cream, Chloramphenicol, Aqueous
4 Finger, Silver Sulfadiazine Cream, Chloramphenicol, Aqueous
5 Perindropil, Silver Sulfadiazine Cream, Chloramphenicol, Aqueous
6 Cefobid, Vaseline Gauze, Silver Sulfadiazine Cream, Chloramphenicol, Aqueous
7 Upper Limb, Silver Sulfadiazine Cream, Chloramphenicol, Aqueous
8 Finger, Paracetamol, Vaseline Gauze, Silver Sulfadiazine Cream, Chloramphenicol, Aqueous
9 Upper Limb, Paracetamol, Vaseline Gauze, Silver Sulfadiazine Cream, Chloramphenicol, Aqueous
10 Difflam Gargle, Silver Sulfadiazine, Vaseline Gauze, Silver Sulfadiazine Cream, Chloramphenicol, Aqueous
11 Cefobid, Silver Sulfadiazine, Vaseline Gauze, Silver Sulfadiazine Cream, Chloramphenicol, Aqueous
12 Lower Limb, Silver Sulfadiazine, Vaseline Gauze, Silver Sulfadiazine Cream, Chloramphenicol, Aqueous
13 Ranitidine, Vaseline Gauze, Silver Sulfadiazine Cream, Chloramphenicol, Aqueous
14 Lower Limb, Silver Sulfadiazine Cream, Chloramphenicol, Aqueous
15 Cefobid, Silver Sulfadiazine Cream, Chloramphenicol, Aqueous
16 Lower Limb, Normal Saline, Vaseline Gauze, Silver Sulfadiazine Cream
17 Face, Paracetamol, Vaseline Gauze, Silver Sulfadiazine Cream, Chloramphenicol, Aqueous
18 Difflam Gargle, Vaseline Gauze, Silver Sulfadiazine Cream, Chloramphenicol, Aqueous
19 Difflam Gargle, Silver Sulfadiazine Cream, Chloramphenicol, Aqueous
20 Palm, Silver Sulfadiazine Cream, Chloramphenicol, Aqueous

Figure 5.3: 20 rules list for Flame sample data.

Explanation:

In *'Flame'* sample data, 20 rules are generated and the generated rules pattern in this algorithm shows that the first 20 rules have the highest pheromone value and so are most appropriate rules to be used (Figure 5.3).

In the rules list, the first rule generated by this prototype is ***[1 Finger, Silver Sulfadiazine, Vaseline Gauze, Silver Sulfadiazine Cream, Chloramphenicol, Aqueous].*** The pattern shows that ***[Vaseline Gauze, Silver Sulfadiazine Cream, Chloramphenicol, Aqueous]*** will appear most frequently the generated rules, especially the combination of ***[Silver Sulfadiazine Cream, Chloramphenicol, Aqueous].***

From the sorted pheromone list, ***[Finger, Silver Sulfadiazine]*** has the highest pheromone value. ***[Finger]*** is classified as the burnt-area. The later section demonstrates how generated rules are to be used by the Expert System.

Rule list: **Flame** (10 rules)

1 Finger, Silver Sulfadiazine, Vaseline Gauze, **Silver Sulfadiazine Cream, Chloramphenicol, Aqueous**
2 Lower Limb, Paracetamol, Vaseline Gauze, **Silver Sulfadiazine Cream, Chloramphenicol, Aqueous**
3 Lower Limb, Vaseline Gauze, **Silver Sulfadiazine Cream, Chloramphenicol, Aqueous**
4 Finger, **Silver Sulfadiazine Cream, Chloramphenicol, Aqueous**
5 Perindropil, **Silver Sulfadiazine Cream, Chloramphenicol, Aqueous**
6 Cefobid, Vaseline Gauze, **Silver Sulfadiazine Cream, Chloramphenicol, Aqueous**
7 Upper Limb, **Silver Sulfadiazine Cream, Chloramphenicol, Aqueous**
8 Finger, Paracetamol, Vaseline Gauze, **Silver Sulfadiazine Cream, Chloramphenicol, Aqueous**
9 Upper Limb, Paracetamol, Vaseline Gauze, **Silver Sulfadiazine Cream, Chloramphenicol, Aqueous**
10 Difflam Gargle, Silver Sulfadiazine, Vaseline Gauze, **Silver Sulfadiazine Cream, Chloramphenicol, Aqueous**

Figure 5.4: 10 rules list for Flame sample data.

To find out if the combination of ***[Finger, Silver Sufaldiazine]*** has the highest pheromone value, we can check it with the pheromone value of these two combinations (Figure 5.5).

```
0 | Finger|,| Silver Sulfadiazine|,|0.52231
1 | Lower Limb|,| Paracetamol|,|0.508486
2 | Lower Limb|,| Vaseline Gauze|,|0.505676
3 | Finger|,| Silver Sulfadiazine Cream|,|0.499009
4 | Perindropil|,| Silver Sulfadiazine Cream|,|0.475194
5 | Cefobid|,| Vaseline Gauze|,|0.472852
6 | Upper Limb|,| Silver Sulfadiazine Cream|,|0.471133
7 | Finger|,| Paracetamol|,|0.461946
8 | Upper Limb|,| Paracetamol|,|0.444938
9 | Difflam Gargle|,| Silver Sulfadiazine|,|0.439419
10 | Cefobid|,| Silver Sulfadiazine|,|0.436857
```

Figure 5.5: 2-itemset list with pheromone values for Flame sample data.

From the list above (Figure 5.5), it can be seen that ***[Finger, Silver Sufaldiazine]*** has the highest pheromone value with 0.52231. It is the value after 1000 cycles of pheromone update and pheromone evaporation process. From ***[Finger, Silver Sulfadiazine],*** the next rule combined with rule 1 is ***Vaseline Gauze*** as ***Silver Sulfadiazine*** combined with ***Vaseline Gauze*** has the highest pheromone value.

It may not get a correct output if it just depends on the 2-itemset count. In the 2-itemset list, the count for ***[Finger, Silver Sulfadiazine]*** is only 1. It means only one record contains ***[Finger, Silver Sulfadiazine].*** But after the 1000 cycles of pheromone update and evaporation process, the value of ***[Finger, Silver Sulfadiazine]*** is 0.52231. So the rule is formed based on the pheromone value after the process.

In Apriori algorithm method, the combination of ***[Finger, Silver Sulfadiazine]*** is pruned as the count is less than the threshold if the threshold is set to 5. A lot of interesting items will be pruned and a lot of interesting rules will not be formed if Apriori algorithm is used.

This new algorithm shows that uninteresting items (infrequent count items) can also form a rule. It can be seen that the result from the above output. After 1000 cycles of

pheromone update and pheromone evaporation, the uninteresting items might become interesting. ***[Finger, Silver Sulfadiazine]*** has proven that an uninteresting item in 2-itemset count becoming an interesting combination after the process.

From the Figure 5.3, the highest count value for 2-itemset is ***[Lower Limb, Paracetamol]***, which is 27 if Apriori algorithm is used. ***[Lower Limb, Paracetamol]*** is a frequent itemset, an interesting item. But the pheromone value is only 0.338489 which is less than ***[Finger, Silver Sulfidiazine].***

Although ***[Finger, Silver Sulfidiazine]*** is an infrequent itemset, which is an uninteresting item in 1-itemset list, the pheromone value is higher than ***[Lower Limb, Paracetamol]*** (Figure 5.6 & Figure 5.7).

```
52  Lower Limb;  Paracetamol|27
882  Finger;  Silver Sulfadiazine|1
```

Figure 5.6: 2-itemset list for Flame sample data with count values for the highest and lowest count.

```
0 | Finger|,| Silver Sulfadiazine|,|0.52231
55 | Lower Limb|,| Paracetamol|,|0.338489
```

Figure 5.7: 2-itemset list for Flame sample data with highest and lowest pheromone value.

5.3 From Generated-Text file to Expert System

In this section, it demonstrates how the generated rules are used by the Expert System. The demonstration is carried out in two ways: In-General and In-Burn-Area.

The output rules in the text file are to guide or assist in creation of an Expert System. They are based on the 8 cases: *Hot Water-burn, Flame-burn, Hot Oil-burn, Firecrackers-burn, Electrical-burn, Hot Soup-burn, Hot Porridge-burn,* and *Hot Air-burn.* For example, in *'Hot Water-burn'* and *'Flame-burn'* case, the 2 sets of output rules are different from each other.

From the 'general' rules list, a special system is needed to rearrange the rules into a compiled text file which is based on: In-General and Burn-Area.

The format of this compiled text file must be compatible with and in a readable format for the Expert System so as to ease the Expert System developer's job and save time. An Expert System can be created or designed in such a way that the input rules can be read from a text file rather than to key-in hundreds or thousands of rules into the coding list.

How the Expert System read the rules from the text file is not discussed in this thesis. This thesis does not focus on the Expert System development.

5.3.1 Based on 'In-General'

Take the case of *'Flame-burn'* for example. From the general output rules list, it is obvious that the pattern of the rule is that the most frequent items always appear in the rule-list, for example, ***'Finger & Silver Sulfadiazine'***. In 'In-General', the compiled medical-items are not related to the area of body that is involved. Here, *'Finger'* from the rule list, is taken out as an example.

Compilation process is required in the system so as to avoid redundancy or duplication in the compiled medical-items. Compilation is performed on the general rules list. For example, if only one rule is adapted from the rules list into the Expert System, the recommended medication for *'Flame-burn'* is ***"Silver Sulfadiazine & Vaseline Gauze & Silver Sulfadiazine Cream & Chloramphenicol & Aqueous"***.

Definitely one rule is not enough to build up an Expert System. Here, the number of rules needed for the compiling process is an issue. But it depends on the user requirements. This research chooses to have at least 10 rules to 20 rules to be more specific.

Let us look at the number of rules for the compiling process. Take 10 to 20 rules for example. ***'Finger & Silver Sulfadiazine'*** have the highest pheromone value, but *'Finger'* is withdrawn from the list, leaving only ***'Silver Sulfadiazine'*** which always appear in each rule. If based on 10 rules, ***'Silver Sulfadiazine'*** will appear 10 times in the rules list. Overlapping or redundancy will happen if no special action is taken. To avoid this from happening, specific compilation process should take place.

After the compiling process, all the compiled items will need to be divided into 4 categories: *Medication, Dressing, Ointment* and *Others.*

Below are the sample outputs for Expert System in 10 rules and 20 rules respectively. Obviously, the 20 rules recommend more medication (Figure 5.8 & Figure 5.9).

10-rules
Case: Flame-burn

Medication: Paracetamol, Perindropil, Cefobid
Dressing: Vaseline Gauze, SSD, Difflam Gargle
Ointment: Silver Sulfadiazine cream, Aqueous, Chloramphenicol
Others: none

Figure 5.8: Sample output rule for Expert System, based Flame-burn case and 10 rules generation.

20-rules
Case: Flame-burn

Medication: Paracetamol, Perindropil, Cefobid, Ranitidine
Dressing: Vaseline Gauze, SSD, Difflam Gargle, Normal Saline
Ointment: Silver Sulfadiazine cream, Aqueous, Chloramphenicol
Others: none

Figure 5.9: Sample output rule for Expert System, based Flame-burn case and 20 rules generation.

For example, an Expert System is developed based on 20 rules generation. Doctor can assist by the Burn Expert System to diagnose a patient who suffers from *Flame* burn. The Burn Expert System will suggest what immediate medication to give to the patient regardless of *Burn-Area.* For example, medication will be '***Paracetamol & Perindropil***', and '***Cefobid & Ranitidine***' maybe optional. Ointments to apply on the burn area will be '***Silver Sulfadiazine cream & Aqueous***' and '***Chloramphenicol***' maybe optional. Dressing

on the wound will be *'Vaseline Gauze & SSD & Normal Saline'* and *'Difflame Gargle'* maybe optional.

5.3.2 Based on 'Burn-Area'

Burn-Area is different from 'In-General' as the compiled-items are based on the body which is burnt. The required medical items are different from one burn area to another. For example, the medical treatment for *'Finger'* burn and *'Face'* burn are different from each other, but they still share some common medications.

This time the compilation is slightly different as compared with 'In-General' compilation. The 'In-General' compilation process is not based on burn area. But in *'Burn-Area'*, the compilation process is to first look into the *'Burn-Area'* items to determine what items available in the rules-list to be stored into a temporary array for later processing purpose.

Compilation of the Burn-Area medical items list is based on the area of burn in whereby the items will be stored separately according to *'Burn-Area'*.

Take *'Finger'* burn area for example. The compiling process is the same as in 'In-General' but this time, it is based on *'Finger'* items in the rules-list

In the *'Finger'* example 5 rules are generated which are rule 1, 4, 8, 38 and 132. The first recommend medication for the patient is ***[Silver Sulfadiazine, Vaseline Gauze, Silver Sulfadiazine Cream, Chloramphenicol, Aqueous]*** (Figure 5.10).

1 Finger, Silver Sulfadiazine, Vaseline Gauze, Silver Sulfadiazine Cream, Chloramphenicol, Aqueous.
4 Finger, Silver Sulfadiazine Cream, Chloramphenicol, Aqueous, Silver Sulfadiazine
8 Finger, Paracetamol, Vaseline Gauze, Silver Sulfadiazine Cream, Chloramphenicol, Aqueous, Silver Sulfadiazine
38 Finger, Cloxacillin, Silver Sulfadiazine, Vaseline Gauze, Silver Sulfadiazine Cream, Chloramphenicol, Aqueous,
132 Finger, Sorbsan, Silver Sulfadiazine Cream, Chloramphenicol, Aqueous, Silver Sulfadiazine

Figure 5.10: Output rules based on burn area - 'Finger'.

In Rule 1, ***[Silver Sulfadiazine Cream, Chloramphenicol & Aqueous]*** also appear in rule 4, 8, 38 and 132. In this case, ***[Silver Sulfadiazine Cream, Chloramphenicol & Aqueous]*** is only added once in the rule-list which belongs to *'Finger'*. In other words, the rules list for *'Finger'* only stores those items that do not appear in the rule-list but are in the general rule-list. But we have to make sure that those rules are covered in *'Finger'* area also.

The later part is to divide items into 4 categories, which are: *Medication, Dressing, Ointment* and *Others* (Figure 5.11).

The sample output for Expert System for 'Burn-Area' is as below. In this example, 10 rules are used as the exact numbers of rules that contain *'Finger'* are less than 10. So at this point, the number of rules depends on the requirement and availability of the rules in the rules-list (Figure 5.11).

Case: Flame-burn
Area: Finger

Medication: Paracetamol, Cloxacillin, Sorbsan
Dressing: Vaseline Gauze, SSD
Ointment: Silver Sulfadiazine cream, Aqueous, Chloramphenicol
Others: none

Figure 5.11: Sample output rule for Expert System, based on the Flame-burn case and burn area– 'Finger'.

Now, the Burn Expert System can provide detail assistant to user. User can defined the area of burn based on the burnt cases. In this example, user besides to define the burn case but also the area – '*Finger*', and the rule generated is based on the '*Finger*' area.

Patient who suffers from *Flame* burn on *Finger* area, the suggested medication will be '***Paracetamol & Cloxacillin***' and '***Sorbsan***' maybe optional. Ointment to apply on the burnt area will be '***Silver Sulfadiazine cream & Aqueous & Chloramphenicol***'. Dressing on the wound will be '***Vaseline Gauze & SSD***'.

5.4 Manually-Checking Analysis

Apriori System uses transactional data in its process. But this Apriori-Ant algorithm uses non-transactional data. Due to time constraint and data format used for the experiment being different from transactional data format used in Apriori System, no experiment has been carried out to test non-transactional data for Apriori system.

The objective of Manually Checking Analysis section is to tally the results generated by Apriori-Ant algorithm with the results generated manually in the experiment.

Manually-Checking Analysis is carried out in two parts: In-Apriori-algorithm and In-Apriori-Ant-Internal-Calculation.

5.4.1 In Apriori algorithm

In Apriori algorithm [1, 3, 4 & 13], *'Firecrackers'* records (Table 5.9) are used for the manual checking. There are 5 records for *'Firecrackers'* case. Table 5.9 shows items which are kept inside the 5 records respectively. This table is for working out the Apriori algorithm manually in later part.

From the *'Firecrackers'* records, there are 24 single items and their counts are generated respectively in Figure 5.13, it is called 1-itemset list. The count for each item is calculated out from Figure 5.12. The *'Finger'* as an example, the count for *'Finger'* is 2, as it appears 2 times in the 5 RecordSet and *Paracetamol* is 5, as it appears 5 times in the RecordSet.

In Apriori algorithm technique, all items in 1-itemset list below 1 are pruned. 1 in this example is count. In this case, only 6 items survive and to proceed to the next step to form 2-itemset list. The 6 items and their respective counts are in Figure 5.12:

1. Finger	2
2. Paracetamol	5
3. Silver Sulfadiazine	4
4. Silver Sulfadiazine Cream	4
7. Vaseline Gauze	4
8. Chloramphenicol	2

Figure 5.12: Items that above threshold values.

Record Number	Items
24	Finger Paracetamol Silver Sulfadiazine Silver Sulfadiazine Cream
38	Face Eye Paracetamol Vaseline Gauze Chloramphenicol Solcoceryl
54	Finger Paracetamol Silver Sulfadiazine Vaseline Gauze Calcium Alginate Silver Sulfadiazine Cream Liquid Paraffin
60	Lower Limb Paracetamol Morphine Cloxacillin Gentamicin Ponstan Mefanemic Morphine Infusion Folic Acid Ferrous Fumerate Chlorpheniramine Silver Sulfadiazine Normal Saline Vaseline Gauze Pressure Garment Silver Sulfadiazine Cream Chloramphenicol
137	Hand Paracetamol Silver Sulfadiazine Vaseline Gauze Silver Sulfadiazine Cream

Table 5.9: Record Set.

1. Finger | 2	13. Morphine | 1
2. Paracetamol | 5	14. Cloxacillin | 1
3. Silver Sulfadiazine | 4	15. Gentamicin | 1
4. Silver Sulfadiazine Cream | 4	16. Ponstan | 1
5. Face | 1	17. Mefanemic | 1
6. Eye | 1	18. Morphine Infusion | 1
7. Vaseline Gauze | 4	19. Folic Acid | 1
8. Chloramphenicol | 2	20. Ferrous Fumerate | 1
9. Solcoceryl | 1	21. Chlorpheniramine | 1
10. Calcium Alginate | 1	22. Normal Saline | 1
11. Liquid Paraffin | 1	23. Pressure Garment | 1
12. Lower Limb | 1	24. Hand | 1

Figure 5.13: Items in 1-ItemSet List and their counts.

From the 6 items in 1-itemset list, 2-itemset list is generated and with their counts respectively (Table 5.10). In 2-itemset list, 2-combined-items items are generated. The 2-combined-items items are generated based on the 6 items that have survived from 1-itemset list.

For example:

The first survived item in 1-itemset list is *Finger* and the second survived item is *'Paracetamol'*, so the combined items in 2-itemset list is *'Finger & Paracetamol'*. Then the next combined item for *'Finger'* is *'Finger & Silver Sulfadiazine'*, as *'Silver Sulfadiazine'* is the third item in the 1-itemset list.

As a result, 15 combined items are generated in the 2-itemset list as compared with Apriori-Ant algorithm which has 30 combined items generated. In Apriori-Ant algorithm, the process stops at this level of generation. In Apriori algorithm, items that are below the threshold (in this example is 2) will be pruned.

2-Items	Counts
Finger & Paracetamol	2
Finger & Silver Sulfadiazine	2
Finger & Silver Sulfadiazine Cream	2
Finger & Vaseline Gauze	1
Finger & Chloramphenicol	0
Paracetamol & Silver Sulfadiazine	4
Paracetamol & Silver Sulfadiazine Cream	4
Paracetamol & Vaseline Gauze	4
Paracetamol & Chloramphenicol	2
Silver Sulfadiazine & Silver Sulfadiazine Cream	4
Silver Sulfadiazine & Vaseline Gauze	3
Silver Sulfadiazine & Chloramphenicol	1
Silver Sulfadiazine Cream & Vaseline Gauze	3
Silver Sulfadiazine Cream & Chloramphenicol	1
Vaseline Gauze & Chloramphenicol	1

Table 5.10: Items in 2-itemset list and their counts (before pruning process takes place).

From the 15 combined items in 2-itemset list, only 10 items survived and to proceed to the next step to generate 3-itemset list (See Table 5.11). In 3-itemset list, 3-combined-items items are generated. The combination is from the survived items at 2-itemset list and 1-itemset list.

For example: The first survived item at 2-itemset list is '*Finger & Paracetamol*'. '*Finger*' and '*Paracetamol*' has appeared in 1-itemset list while the next survived item in 1-itemset list is '*Silver Sulfadiazine*'. Thus, the combined items for 3-itemset list are, '*Finger & Paracetamol & Silver Sulfadiazine*'.

There are 22 combined items generated in the 3-itemset list (See Table 5.11). In this level, the threshold is still set to 2. So whatever items below 2 will be pruned. In this level, 9 combine items survive and proceed to the next step to generate 4-itemset list.

3-Items	Counts
Finger & Paracetamol & Silver Sulfadiazine	2
Finger & Paracetamol & Silver Sulfadiazine Cream	2
Finger & Paracetamol & Vaseline Gauze	1
Finger & Paracetamol & Chloramphenicol	0
Finger & Silver Sulfadiazine & Silver Sulfadiazine Cream	2
Finger & Silver Sulfadiazine & Vaseline Gauze	1
Finger & Silver Sulfadiazine & Chloramphenicol	0
Finger & Silver Sulfadiazine Cream & Vaseline Gauze	1
Finger & Silver Sulfadiazine Cream & Chloramphenicol	0
Paracetamol & Silver Sulfadiazine & Silver Sulfadiazine Cream	3
Paracetamol & Silver Sulfadiazine & Vaseline Gauze	3
Paracetamol & Silver Sulfadiazine & Chloramphenicol	1
Paracetamol & Silver Sulfadiazine Cream & Vaseline Gauze	3
Paracetamol & Silver Sulfadiazine Cream & Chloramphenicol	1
Paracetamol & Vaseline Gauze & Silver Sulfadiazine Cream	2
Paracetamol & Vaseline Gauze & Chloramphenicol	2
Paracetamol & Chloramphenicol & Silver Sulfadiazine Cream	1
Silver Sulfadiazine & Silver Sulfadiazine Cream & Vaseline Gauze	3
Silver Sulfadiazine & Silver Sulfadiazine Cream & Chloramphenicol	1
Silver Sulfadiazine & Vaseline Gauze & Chloramphenicol	1
Silver Sulfadiazine Cream & Vaseline Gauze & Finger	1
Silver Sulfadiazine Cream & Vaseline Gauze & Chloramphenicol	1

Table 5.11: Items in 3-itemset list and their counts (before pruning process takes place).

In 4-itemset list, 4-combined-item items are generated. The combination is from the survived items a 3-itemset list and 1-itemset list.

For example: The first survived items at 3-itemset list are *'Finger & Paracetamol & Silver Sulfadiazine'*. *'Finger'* and *'Paracetamol'* and *'Silver Sulfadiazine'* have appeared in 1-itemset list, and the next survived item in 1-itemset list is *'Silver Sulfadiazine Cream'*. So the first combined items for 4-itemset list are *'Finger & Paracetamol & Silver Sulfadiazine & Silver Sulfadiazine Cream'*. The count is 2, as this combination appears twice in the Table 5.9.

4-Items	Counts
Finger & Paracetamol & Silver Sulfadiazine & Silver Sulfadiazine Cream	2
Finger & Paracetamol & Silver Sulfadiazine & Vaseline Gauze	1
Finger & Paracetamol & Silver Sulfadiazine & Chloramphenicol	1
Finger & Paracetamol & Silver Sulfadiazine Cream & Vaseline Gauze	0
Finger & Paracetamol & Silver Sulfadiazine Cream & Chloramphenicol	1
Finger & Silver Sulfadiazine & Silver Sulfadiazine Cream & Vaseline Gauze	0
Finger & Silver Sulfadiazine & Silver Sulfadiazine Cream & Chloramphenicol	1
Paracetamol & Silver Sulfadiazine & Silver Sulfadiazine Cream & Vaseline Gauze	0
Paracetamol & Silver Sulfadiazine & Silver Sulfadiazine Cream & Chloramphenicol	3
Paracetamol & Silver Sulfadiazine & Vaseline Gauze & Chloramphenicol	1
Paracetamol & Vaseline Gauze & Silver Sulfadiazine Cream & Chloramphenicol	1
Paracetamol & Vaseline Gauze & Chloramphenicol & Finger	1
Silver Sulfadiazine & Silver Sulfadiazine Cream & Vaseline Gauze & Chloramphenicol	0
	1

Table 5.12: Items in 4-itemset list and their counts (before pruning process takes place).

There are 14 combined items generated in the 4-itemset list (See Table 5.12). In this level, the threshold is set to 2. Thus, which items below 2 will be pruned. In this level, 2 combined items survive and proceed to next step to generate 5-itemset list (Table 5.13).

5-Items	Counts
Finger & Paracetamol & Silver Sulfadiazine & Silver Sulfadiazine Cream & Vaseline Gauze	1
Finger & Paracetamol & Silver Sulfadiazine & Silver Sulfadiazine cream & Chloramphenicol	0
Paracetamol & Silver Sulfadiazine & Silver Sulfadiazine Cream & Vaseline Gauze & Chloramphenicol	1

Table 5.13: Items in 5-itemset list and their counts (before pruning process takes place).

3 combined items are generated in the list in this level (Table 5.13). The combined items are 5 single items. In this level, the threshold is set to 1. Thus, which items below 1

will be pruned. As a result, 2 combined items survive and the process stops at this level. Two rules are generated at the end of this process. There are:

1. *Finger & Paracetamol & Silver Sulfadiazine & Silver Sulfadiazine Cream & Vaseline Gauze*
2. *Paracetamol & Silver Sulfadiazine & Silver Sulfadiazine Cream & Vaseline Gauze & Chloramphenicol*

Rule 1: Finger & Paracetamol & Silver Sulfadiazine & Silver Sulfadiazine cream & Vaseline Gauze is appears in rule 64, 130, 142, 159 and 167 respectively which are overlapping (Figure 5.14).

64 Finger, Silver Sulfadiazine Cream, Chloramphenicol, Solcoceryl
130 Finger, Vaseline Gauze, Chloramphenicol, Solcoceryl
142 Finger, Calcium Alginate, Silver Sulfadiazine Cream, Chloramphenicol, Solcoceryl
159 Finger, Silver Sulfadiazine, Silver Sulfadiazine Cream, Chloramphenicol, Solcoceryl
167 Finger, Paracetamol, Silver Sulfadiazine Cream, Chloramphenicol, Solcoceryl

Figure 5.14: Rules list based on burn area – 'Finger'.

After compilation of these rules, it will be *[Finger, Silver Sulfadiazine, Silver Sulfadiazine Cream, Chloramphenicol, Solcocery, Vaseline Gauze, Calcium Alginate].*

And *Rule 2: Paracetamol & Silver Sulfadiazine & Silver Sulfadiazine Cream & Vaseline Gauze & Chloramphenicol* is appears in rule 1, 12 and 29 respectively which are overlapping (Figure 5.15).

1 Paracetamol, Silver Sulfadiazine Cream, Chloramphenicol, Solcoceryl
12 Paracetamol, Silver Sulfadiazine, Silver Sulfadiazine Cream, Chloramphenicol, Solcoceryl
29 Paracetamol, Vaseline Gauze, Chloramphenicol, Solcoceryl

Figure 5.15: Rules list based on general medicine terms.

After compilation of these rules, it will be ***[Paracetamol, Silver Sulfadiazine, Silver Sulfadiazine Cream, Chloramphenicol, Solcocery, Vaseline Gauze]***.

This proves that Apriori-Ant algorithm can produce the same rules as Apriori algorithm does. Beside that, more information can be found out from this technique.

5.4.2 In-Apriori-Ant-Internal-Calculation

In In-Apriori-Ant-Internal-Calculation, *'Hot Soup'* case is used. This case consists of only 2 records for the checking which is mainly on the internal calculation part involved in Apriori-Ant algorithm.

The manual checking is carried out in 3 calculation areas:

- To calculate the *xCount* and *yCount* for *Item-X* and *Item-Y* respectively (Table 5.16).

  Following are the simple formula for the calculation

  Formula:

  xCount = count for item-X/total number of record. *(refer to 12)*

  yCount = count for item-Y/total number of record. *(refer to 13)*

- To calculate the Phe, following are the formula to calculate Phe (Table 5.16):

Formula:

ProConf = ConfidenceCnt/Total Record *(refer to 14)*

Conf = ProConf/xCount. *(refer to 20)*

Lift = (Conf/yCount)*xCount *(refer to 22)*

Phe = (ProConf*0.5)/(Lift+yCount)*0.8 *(refer to 24)*

* ConfidenceCnt is the count for both Item-X and Item-Y appear together in the record-set.

- To calculate the Pheromone (Table 5.16).

  Formula:

  Pheromone = (Pher/TotalPher)*100 *(refer to 26)*

* TotalPher is the total of Phe

Before calculation, two important stages are required. First is to generate 1-itemset list and the counts for each item in the list (Table 5.14). Second is to generate 2-itemset list and the counts for each combined items in the list (Table 5.15).

No	Item	SupportCnt
1	Lower Limb	2
2	Upper Limb	1
3	Paracetamol	2
4	Silver Sulfadiazine	2
5	Hydrocolloid	1
6	Silver Sulfadiazine Cream	2
7	Phenergan	1
8	Askina	1
9	Vaseline Gauze	1

Table 5.14: Items in 1-itemsetlist and their counts for Hot Soup sample data.

No	Item-X	Item-Y	ConfidenceCnt
1	Lower Limb	Upper Limb	1
2	Lower Limb	Paracetamol	2
3	Lower Limb	Silver Sulfadiazine	2
4	Lower Limb	Hydrocolloid	1
5	Lower Limb	Silver Sulfadiazine Cream	2
6	Upper Limb	Paracetamol	1
7	Upper Limb	Silver Sulfadiazine	1
8	Upper Limb	Hydrocolloid	1
9	Upper Limb	Silver Sulfadiazine Cream	1
10	Paracetamol	Silver Sulfadiazine	2
11	Paracetamol	Hydrocolloid	1
12	Paracetamol	Silver Sulfadiazine Cream	2
13	Silver Sulfadiazine	Hydrocolloid	1
14	Silver Sulfadiazine	Silver Sulfadiazine Cream	2
15	Hydrocolloid	Silver Sulfadiazine Cream	1
16	Lower Limb	Phenergan	1
17	Lower Limb	Askina	1
18	Lower Limb	Vaseline Gauze	1
19	Paracetamol	Phenergan	1
20	Paracetamol	Askina	1
21	Paracetamol	Vaseline Gauze	1
22	Phenergan	Silver Sulfadiazine	1
23	Phenergan	Askina	1
24	Phenergan	Vaseline Gauze	1
25	Phenergan	Silver Sulfadiazine Cream	1
26	Silver Sulfadiazine	Askina	1
27	Silver Sulfadiazine	Vaseline Gauze	1
28	Askina	Vaseline Gauze	1
29	Askina	Silver Sulfadiazine Cream	1
30	Vaseline Gauze	Silver Sulfadiazine Cream	1

Table 5.15: Items in 2-itemset list and their counts for Hot Soup sample data.

The following part is to show how each formula is to be carried out. First to work out on record number 1, *Lower Limb* and *Upper Limb*. *xCount (12)* and *yCount (13)* for *Lower Limb* and *Upper Limb* is 1 and 0.5 respectively (Table 5.16). To calculate Phe, following formula is used.

Formula(Phe):

ProConf = ConfidenceCnt/Total Record *(refer to 14)*

Conf = ProConf/xCount. *(refer to 20)*

Lift = (Conf/yCount)*xCount *(refer to 22)*

Phe = (ProConf*0.5)/(Lift+yCount)*0.8 *(refer to 24)*

Answer:

ProConf = 1/2 = 0.5

Conf = 0.5/1 = 0.5

Lift = (0.5/0.5) * 1 = 1

Phe = (0.5*0.5)/(1+0.5)0.8

= 0.25/(1.5)*0.8

= 0.25/1.2

= 0.208333 *(33)*

Pheromone formula as below:

Formula (Pheromone):

Pheromone = (Pher/TotalPher)*100 *(refer to 26)*

Answer:

Pheromone = (0.208333/6.875)*100

= 0.030302981

= 3.0303 *(34)*

Checking on record number 2, *Lower Limb* and *Upper Limb*. *xCount* and *yCount* for *Lower Limb* and *Upper Limb* is 1 and 0.5 respectively (Table 5.16). Formula refers to *Formula(Phe)(14,20,22&24)* and *Formula(Pheromone)(26)* on above.

Answer:

ProConf = 2/2 = 1

Conf = 1/1 = 1

Lift = (1/1) * 1 = 1

Phe = (1*0.5)/(1+1)0.8

= 0.5/(2)*0.8

= 0.5/1.6

= 0.3125 *(35)*

Answer:

Pheromone = (0.3125/6.875)*100

= 0.0454545

= 4.5454 *(36)*

The following table (Table 5.16) is to display the answers of *xCount*, *yCount*, *Phe (33&35)* and *Pheromone (34&36)* value with there respective items in 2-itemset list.

Figure 5.16 is used to tally the result with Table 5.16. The calculation formula is accurate at this level. In Table 5.16, in ***[Lower Limb; Upper Limb | 0.208333 | 3.0303]***, 0.208333 denoted as Phe value and 3.0303 denoted as Pheromone value.

No	Item-X	Item-Y	xCount	yCount	Phe	Pheromone
1	Lower Limb	Upper Limb	2/2 = 1	1/2 = 0.5	0.208333	3.0303
2	Lower Limb	Paracetamol	2/2 = 1	2/2 = 1	0.3125	4.5454
3	Lower Limb	Silver Sulfadiazine	2/2 = 1	2/2 = 1	0.3125	4.5454
4	Lower Limb	Hydrocolloid	2/2 = 1	1/2 = 0.5	0.208333	3.0303
5	Lower Limb	Silver Sulfadiazine Cream	2/2 = 1	2/2 = 1	0.3125	4.5454
6	Upper Limb	Paracetamol	1/2 = 0.5	2/2 = 1	0.208333	3.0303
7	Upper Limb	Silver Sulfadiazine	1/2 = 0.5	2/2 = 1	0.208333	3.0303
8	Upper Limb	Hydrocolloid	1/2 = 0.5	1/2 = 0.5	0.208333	3.0303
9	Upper Limb	Silver Sulfadiazine Cream	1/2 = 0.5	2/2 = 1	0.208333	3.0303
10	Paracetamol	Silver Sulfadiazine	2/2 = 1	2/2 = 1	0.3125	4.5454
11	Paracetamol	Hydrocolloid	2/2 = 1	1/2 = 0.5	0.208333	3.0303
12	Paracetamol	Silver Sulfadiazine Cream	2/2 = 1	2/2 = 1	0.3125	4.5454
13	Silver Sulfadiazine	Hydrocolloid	2/2 = 1	1/2 = 0.5	0.208333	3.0303
14	Silver Sulfadiazine	Silver Sulfadiazine Cream	2/2 = 1	2/2 = 1	0.3125	4.5454
15	Hydrocolloid	Silver Sulfadiazine Cream	1/2 = 0.5	2/2 = 1	0.208333	3.0303
16	Lower Limb	Phenergan	2/2 = 1	1/2 = 0.5	0.208333	3.0303
17	Lower Limb	Askina	2/2 = 1	1/2 = 0.5	0.208333	3.0303
18	Lower Limb	Vaseline Gauze	2/2 = 1	1/2 = 0.5	0.208333	3.0303
19	Paracetamol	Phenergan	2/2 = 1	1/2 = 0.5	0.208333	3.0303
20	Paracetamol	Askina	2/2 = 1	1/2 = 0.5	0.208333	3.0303
21	Paracetamol	Vaseline Gauze	2/2 = 1	1/2 = 0.5	0.208333	3.0303
22	Phenergan	Silver Sulfadiazine	1/2 = 0.5	2/2 = 1	0.208333	3.0303
23	Phenergan	Askina	1/2 = 0.5	1/2 = 0.5	0.208333	3.0303
24	Phenergan	Vaseline Gauze	1/2 = 0.5	1/2 = 0.5	0.208333	3.0303
25	Phenergan	Silver Sulfadiazine Cream	1/2 = 0.5	2/2 = 1	0.208333	3.0303
26	Silver Sulfadiazine	Askina	1/2 = 0.5	1/2 = 0.5	0.208333	3.0303
27	Silver Sulfadiazine	Vaseline Gauze	2/2 = 1	1/2 = 0.5	0.208333	3.0303
28	Askina	Vaseline Gauze	2/2 = 1	1/2 = 0.5	0.208333	3.0303
29	Askina	Silver Sulfadiazine Cream	1/2 = 0.5	1/2 = 0.5	0.208333	3.0303
30	Vaseline Gauze	Silver Sulfadiazine Cream	1/2 = 0.5	2/2 = 1	0.208333	3.0303
			Total Phe		6.875	

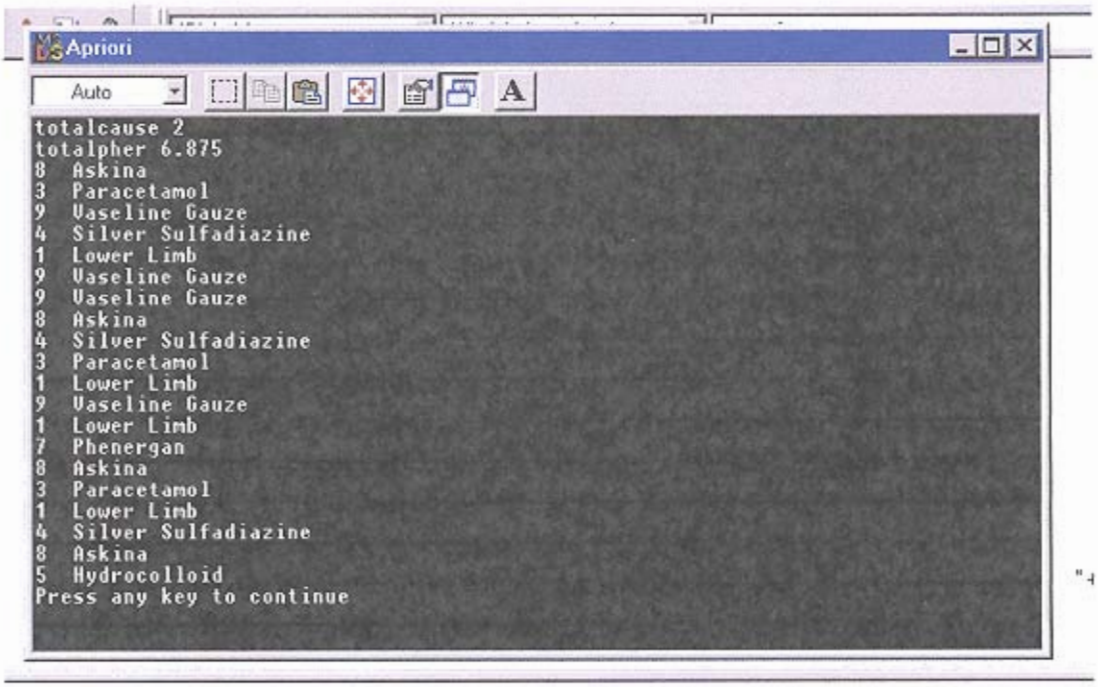
able 5.16: Answers for xCount, yCount, Phe and Pheromone for Hot Soup sample data based on 2-itemset list.

```
Lower Limb;  Upper Limb|0.208333|3.0303
Lower Limb;  Paracetamol|0.3125|4.54545
Lower Limb;  Silver Sulfadiazine|0.3125|4.54545
Lower Limb;  Hydrocolloid|0.208333|3.0303
Lower Limb;  Silver Sulfadiazine Cream|0.3125|4.54545
Upper Limb;  Paracetamol|0.208333|3.0303
Upper Limb;  Silver Sulfadiazine|0.208333|3.0303
Upper Limb;  Hydrocolloid|0.208333|3.0303
Upper Limb;  Silver Sulfadiazine Cream|0.208333|3.0303
Paracetamol;  Silver Sulfadiazine|0.3125|4.54545
Paracetamol;  Hydrocolloid|0.208333|3.0303
Paracetamol;  Silver Sulfadiazine Cream|0.3125|4.54545
Silver Sulfadiazine;  Hydrocolloid|0.208333|3.0303
Silver Sulfadiazine;  Silver Sulfadiazine Cream|0.3125|4.54545
Hydrocolloid;  Silver Sulfadiazine Cream|0.208333|3.0303
Lower Limb;  Phenergan|0.208333|3.0303
Lower Limb;  Askina|0.208333|3.0303
Lower Limb;  Vaseline Gauze|0.208333|3.0303
Paracetamol;  Phenergan|0.208333|3.0303
Paracetamol;  Askina|0.208333|3.0303
Paracetamol;  Vaseline Gauze|0.208333|3.0303
Phenergan;  Silver Sulfadiazine|0.208333|3.0303
Phenergan;  Askina|0.208333|3.0303
Phenergan;  Vaseline Gauze|0.208333|3.0303
Phenergan;  Silver Sulfadiazine Cream|0.208333|3.0303
Silver Sulfadiazine;  Askina|0.208333|3.0303
Silver Sulfadiazine;  Vaseline Gauze|0.208333|3.0303
Askina;  Vaseline Gauze|0.208333|3.0303
Askina;  Silver Sulfadiazine Cream|0.208333|3.0303
Vaseline Gauze;  Silver Sulfadiazine Cream|0.208333|3.0303
```

Figure 5.16: Sample rules for Hot Soup sample data (generated by the prototype system). The output figures are Phe and Pheromone values.

In this *'Hot Soup'* case, checking is done in 20 cycles in which the randomly selected items for testing are, ***[Askina, Paracetamol, Vaseline Gauze, Silver Sulfadiazine, Lower Limb, Phenergan, Hydrocolloid]*** (Display 1).

Items randomly selected are not in proper sequence and may appear more than one time.



Display 1: Random items selected in 20-cycle process in Hot Soup sample data (generated by the prototype system).

The following section shows the calculation of pheromone update and pheromone evaporation.

5.4.2.1 Pheromone Update.

The format for the pheromone update is based on the randomly selected item. In the first cycle, *Askina* is selected and will be updated using formula *(30)* below. From Figure: 1, *Askina* combination items are at no 28 and 29 respectively. In this case, only the 'Phe' value is used for calculation.

The formula for pheromone update as below:

(0.04*pheromone-1)+pheromone+(*SupportCnt**0.001) *(refer to 30)*

No	Item-X	Item-Y	Phe	Update
27	Silver Sulfadiazine	Vaseline Gauze	3.0303	X
28	Askina	Vaseline Gauze	3.0303	
29	Askina	Silver Sulfadiazine Cream	3.0303	

Table 5.17: Pheromone Update for 'Askina' item in Hot Soup sample data.

In Table 5.17, the *X* in the Update column denotes that no update process has been carried out. Pheromone-1 is denoted as the pheromone value in No27 (In this example that is 3.0303). Also in this example, *SupportCnt* is the count for *Askina & Vaseline Gauze* that appears in the recordset is 1 (Table 5.17).

(0.04*3.0303)+3.0303+(1*0.001)

= 0.121212 + 3.0303 + 0.001

= 3.15251 *(37)*

No	Item-X	Item-Y	Phe	Update
27	Silver Sulfadiazine	Vaseline Gauze	3.0303	X
28	Askina	Vaseline Gauze	3.0303	3.15251
29	Askina	Silver Sulfadiazine Cream	3.0303	

Table 5.18: Pheromone Update for 'Askina' item in Hot Soup sample data.

Next Pheromone update for No29 is slightly different from No28. The reason is that, in record No28, no 'updated' pheromone-1 value can be used, thus the most 'updated' pheromone-1 value is the 'Phe' value in No27. But for No29, the previous pheromone value in No28 is 3.15251 *(37)* instead of 3.0303. In this case, the *SupportCnt* for *Askina & Silver Sulfadiazine Cream* is 2 (Table 5.18 & Table 5.19).

(0.04*3.15251)+3.0303+(2*0.001)

= 0.1261004 + 3.0303 + 0.002

= 3.1584 *(38)*

No	Item-X	Item-Y	Phe	Update
27	Silver Sulfadiazine	Vaseline Gauze	3.0303	X
28	Askina	Vaseline Gauze	3.0303	3.15251
29	Askina	Silver Sulfadiazine Cream	3.0303	3.1584

Table 5.19: Pheromone Update for 'Askina' item in Hot Soup sample data.

This calculation is carried out in Microsoft Excel. A simple formula is used in the worksheet. The result for the first cycle: *Askina* is as below (Display 2).

G2 =

	A	B	C	D	E	
1						
2						
3				Cycle 1		
4	No	2-ItemSet	P	U	E	L
5	21	Paracetamol & Vaseline Gauze	3.030298	x	3.027268	
6	22	Phenergan & Silver Sulfadiazine	3.030298	x	3.027268	x
7	23	Phenergan & Askina	3.030298	x	3.027268	x
8	24	Phenergan & Vaseline Gauze	3.030298	x	3.027268	x
9	25	Phenergan & Silver Sulfadiazine Cream	3.030298	x	3.027268	x
10	26	Silver Sulfadiazine & Askina	3.030298	x	3.027268	x
11	27	Silver Sulfadiazine & Vaseline Gauze	3.030298	x	3.027268	x
12	28	Askina & Vaseline Gauze	3.030298	3.15251	3.149358	x
13	29	Askina & Silver Sulfadiazine Cream	3.030298	3.158399	3.15524	x
14	30	Vaseline Gauze & Silver Sulfadiazine Cream	3.030298	x	3.027268	x
15						
16						
17		Note: P-Pheromone value, U-Pheromone UpDate, E-Pheromone Evaporate				
18						

Display 2: Pheromone Update value for 'Askina & Vaseline Gauze' and 'Askina & Silver Sulfadiazine Cream' produced by Microsoft Excel (Hot Soup sample data).

5.4.2.2 Pheromone Evaporation.

After Pheromone Update formula is carried out, the next step is to perform Pheromone Evaporation. The difference between Pheromone Update and Evaporation is that, Update is only carried out in the random selected items while the process is carried out in every item for Pheromone Evaporation.

Take the previous section as an example. Evaporation is performed in the first cycle: '*Askina*'. From the diagram below (Display 3), every item will experience evaporation process.

G2 =

	A	B	C	D	E	
1						
2						
3				Cycle 1		
4	No	2-ItemSet	P	U	E	U
5	21	Paracetamol & Vaseline Gauze	3.030298	x	3.027268	
6	22	Phenergan & Silver Sulfadiazine	3.030298	x	3.027268	x
7	23	Phenergan & Askina	3.030298	x	3.027268	x
8	24	Phenergan & Vaseline Gauze	3.030298	x	3.027268	x
9	25	Phenergan & Silver Sulfadiazine Cream	3.030298	x	3.027268	x
10	26	Silver Sulfadiazine & Askina	3.030298	x	3.027268	x
11	27	Silver Sulfadiazine & Vaseline Gauze	3.030298	x	3.027268	x
12	28	Askina & Vaseline Gauze	3.030298	3.15251	3.149358	x
13	29	Askina & Silver Sulfadiazine Cream	3.030298	3.158399	3.15524	x
14	30	Vaseline Gauze & Silver Sulfadiazine Cream	3.030298	x	3.027268	x
15						
16						
17		Note: P-Pheromone value, U-Pheromone UpDate, E-Pheromone Evaporate				
18						

Display 3: Pheromone Evaporation value for 'Askina & Vaseline Gauze' and 'Askina & Silver Sulfadiazine Cream' produced by Microsoft Excel (Hot Soup sample data).

The formula for pheromone evaporation as below:

Pheromone Evaporation = Pheromone-(0.001*Pheromone) *(refer to 32)*

No	Item-X	Item-Y	Phe	Update	Evaporate
27	Silver Sulfadiazine	Vaseline Gauze	3.0303	X	
28	Askina	Vaseline Gauze	3.0303	3.15251	
29	Askina	Silver Sulfadiazine Cream	3.0303	3.1584	

Table 5.20: Pheromone Evaporation for 'Askina' item.

For No.27, '*Silver Sulfadiazine & Vaseline Gauze*', Pheromone Evaporation is carried out based on **Phe** value as no Pheromone Update process is carried out in this cycle.

Evaporate = 3.0303 - (0.001 * 3.0303)

= 3.0303 - 0.0030303

= 3.02727 *(39)*

For No28, *'Askina & Vaseline Gauze'*, Pheromone Evaporation is carried out based on the 'Update' value in the column as that is the updated pheromone value in this cycle (Table 5.20 & Table 5.21).

Evaporate = 3.15251 - (0.001 * 3.15251)

= 3.15251- 0.00315251

= 3.14936 *(40)*

No	Item-X	Item-Y	Phe	Update	Evaporate
27	Silver Sulfadiazine	Vaseline Gauze	3.0303	X	3.02727
28	Askina	Vaseline Gauze	3.0303	3.15251	3.14936
29	Askina	Silver Sulfadiazine Cream	3.0303	3.1584	

Table 51: Pheromone Evaporation for 'Askina' item in Hot Soup sample data.

Manual checking is done using Microsoft Excel in 20 cycles and the following are the output results. See more detail calculation from cycle 1 to cycle 20 in Appendix B, C, D, & E, each diagram consists 5 cycles in process.

The final result in 20-cycle is tallied to the results generated by the prototype (Figure 5.17) and the results generated by Microsoft Excel (Display 4).

	A	B	L	M	N	O	P
115							
116							
117			Cycle 20				
118	No	2-ItemSet	U	E			
119	1	LowerLimb & UpperLimb	x	2.97423	1		
120	2	LowerLimb & Paracetamol	x	4.93897	2		
121	3	LowerLimb & Silver Sulfadiazine	x	5.22454	2		
122	4	LowerLimb & Hydrocolloid	x	3.76304	1		
123	5	LowerLimb & Silver Sulfadiazine Cream	x	5.01629	2		
124	6	UpperLimb & Paracetamol	x	2.97026	2		
125	7	UpperLimb & Silver Sulfadiazine	x	2.97026	2		
126	8	UpperLimb & Hydrocolloid	x	2.97026	1		
127	9	UpperLimb & Silver Sulfadiazine Cream	x	2.97026	2		
128	10	Paracetamol & Silver Sulfadiazine	x	4.81777	2		
129	11	Paracetamol & Hydrocolloid	x	3.53687	1		
130	12	Paracetamol & Silver Sulfadiazine Cream	x	4.86271	2		
131	13	Silver Sulfadiazine & Hydrocolloid	x	3.53442	1		
132	14	Silver Sulfadiazine & Silver Sulfadiazine Cream	x	4.86246	2		
133	15	Hydrocolloid & Silver Sulfadiazine Cream	3.16993	3.16676	2		
134	16	LowerLimb & Phenergan	x	3.44947	1		
135	17	LowerLimb & Askina	x	3.49739	1		
136	18	LowerLimb & Vaseline Gauze	x	3.50122	1		
137	19	Paracetamol & Phenergan	x	3.35026	1		
138	20	Paracetamol & Askina	x	3.35944	1		
139	21	Paracetamol & Vaseline Gauze	x	3.36038	1		
140	22	Phenergan & Silver Sulfadiazine	x	3.10125	2		
141	23	Phenergan & Askina	x	3.09531	1		
142	24	Phenergan & Vaseline Gauze	x	3.09507	1		
143	25	Phenergan & Silver Sulfadiazine Cream	x	3.09605	2		
144	26	Silver Sulfadiazine & Askina	x	3.3347	1		
145	27	Silver Sulfadiazine & Vaseline Gauze	x	3.35862	1		
146	28	Askina & Vaseline Gauze	x	3.48014	1		
147	29	Askina & Silver Sulfadiazine Cream	x	3.50338	2		
148	30	Vaseline Gauze & Silver Sulfadiazine Cream	x	3.47871	2		
149							
150							
151		Note: Pr-Previous Pheromone value in Cycle 15	U-Pheromone UpDate, E-Pheromone Evaporate				
152							
153							

Display 4: Pheromone Update and Evaporation value after 20-cycle in Hot Soup sample data (generated by Microsoft Excel).

1 Lower Limb Upper Limb 2.97423
2 Lower Limb Paracetamol 4.93897
3 Lower Limb Silver Sulfadiazine 5.22454
4 Lower Limb Hydrocolloid 3.76304
5 Lower Limb Silver Sulfadiazine Cream 5.01629
6 Upper Limb Paracetamol 2.97027
7 Upper Limb Silver Sulfadiazine 2.97027
8 Upper Limb Hydrocolloid 2.97027
9 Upper Limb Silver Sulfadiazine Cream 2.97027
10 Paracetamol Silver Sulfadiazine 4.81777
11 Paracetamol Hydrocolloid 3.53687
12 Paracetamol Silver Sulfadiazine Cream 4.86271
13 Silver Sulfadiazine Hydrocolloid 3.53443
14 Silver Sulfadiazine Silver Sulfadiazine Cream 4.86246
15 Hydrocolloid Silver Sulfadiazine Cream 3.16676
16 Lower Limb Phenergan 3.44947
17 Lower Limb Askina 3.49739
18 Lower Limb Vaseline Gauze 3.50123
19 Paracetamol Phenergan 3.35026
20 Paracetamol Askina 3.35944
21 Paracetamol Vaseline Gauze 3.36038
22 Phenergan Silver Sulfadiazine 3.10125
23 Phenergan Askina 3.09531
24 Phenergan Vaseline Gauze 3.09507
25 Phenergan Silver Sulfadiazine Cream 3.09606
26 Silver Sulfadiazine Askina 3.3347
27 Silver Sulfadiazine Vaseline Gauze 3.35862
28 Askina Vaseline Gauze 3.48015
29 Askina Silver Sulfadiazine Cream 3.50338
30 Vaseline Gauze Silver Sulfadiazine Cream 3.47871

Figure 5.1: Pheromone values after 20-cycle process in Hot Soup sample data (generated by the prototype system). It is used to tally the values done by manual calculation (carried out by using Microsoft Excel).

From the two output results, Apriori-Ant-Internal-Calculation proves to be accurate. Rule generation is based on the highest pheromone value. In this example, the first rule generated by this prototype is ***[Lower Limb, Silver Sulfadiazine]*** with the highest pheromone value of *[5.22454]*.

Chapter 6: Discussion

The combination of ACO algorithm [7 & 15] and Apriori algorithm [1, 11 & 12] can remedy each other's weaknesses. This research discovers that the two combined algorithm can work together to generate rules from a non-transactional type of data and at the same time replace the repetitive I/O scan during the candidates generation in Apriori algorithm.

There are a few significant findings from this research:

- The candidates generation process which involve a lot of pruning and joining process in Apriori algorithm is replaced by the Apriori-Ant algorithm. Only one joining process is carried out, and the process stops at first level of joining. No pruning is involved here.

- In this case, no interesting rules of the infrequent items are left out in the joining process because this new joint algorithm excludes pruning in the process. This is to avoid pruning away those infrequent items which may generate interesting rules at the end of the process [4].

- The rule-generation process is replaced by ACO algorithm [15]. Only items in 2-itemset list with the highest pheromone value will form rules. And the first items which appear in the rules list are items with the highest pheromone value.

- Some of the items that appear in rule No.1 to 20 are infrequent or uninteresting in Apriori algorithm [7] as the counts are below certain threshold. In Apriori algorithm

[7], such items would have been pruned from the list. But in this Apriori-Ant algorithm, such items are interesting and may produce good rules.

In this thesis, the experiment is carried out based on two areas: Number of Rules, and the Processing Time.

- Number of Rules.

  The number of items in the itemset list greatly affects the number of rules generated.

  Take *'Hot Air'* as an example. Only 1 record is involved in the process. Although there is only 1 record, it can generate 4 items in 1-itemset list and 6 items in 2-itemset list. Thus, 6 rules are generated at the end of the process. From the 6 rules, at least 1 or 2 rules may be interesting to a user.

  Certain cases only involve 1 record like *'Hot Air'* and *'Hot Porridge'*. For these two cases, the number of record is the same, but the items generated in 1-itemset list, 2-itemset list and rules are different from each other. At this point, pheromone value has an impact on the number of rules generated, and indirectly it has relation with the number of items generated in both lists.

  Another finding under this research is that the number of rules is correlated with the items in 2-itemset list. The more items that are generated, the more rules are produced. And the more items there are in 1-itemset list, the more items in 2-itemset list. In other words, items in 1-itemset list and in 2-itemset list and the

number of rules are correlated with each other. But the number of records cannot indicate the number of rules to be produced.

- Processing Time.

No processing time is wasted on the candidates generation and the I/O scan on the database. Only two I/O scans over the database are required in Apriori-Ant algorithm. A lot of research is still required in the processing time. In an Apriori-Ant algorithm, the factor that influences the processing time is the '***Total Pheromone value***'.

In Apriori-Ant algorithm, pheromone value replaces the threshold used to determine the destination of the items to form a rule. Threshold is a user-defined value used to determine which items are the uninteresting items that will be pruned in Apriori algorithm.

The sensitivity of Pheromone value may indicate the quality of each rules generated, which plays the same role as threshold in Apriori algorithm. As the pheromone value has impact on the processing time, an effective formula on calculating the pheromone is important and the formula still can retain the quality of rules generated.

Obviously, the number of records has an impact on processing time, but not on all sample-data. For example, in *'Firecrackers'* and *'Electrical'* sample-data, *'Firecrackers'* has *5 records with 19.9161 pheromone value* and it takes *19 seconds* generation time as compared to *'Electrical'* of *4 records and 43.9192 pheromone*

which takes *36 seconds*. It clearly shows that it is the pheromone value which affects processing time and not the number of records.

Chapter 7: Conclusion

Apriori-Ant algorithm is a combination of Apriori algorithm [1, 11 & 12] and ACO algorithm [7 & 15]. The technique is to replace the candidates generation process which requires repetitive I/O scans over the database in Apriori algorithm at the initial stage. The candidates generation involves pruning and joining process. In Apriori-Ant algorithm, only the joining process is carried up to level-2. The Level-2 process is to generate item and count of 1-itemset (Level-1), 2-itemset (Level-2).

The second step of Apriori-Ant algorithm is ACO algorithm. In ACO algorithm, the original calculation formula for Pheromone is used in Apriori-Ant algorithm, but the formula to calculate the pheromone has been modified. The number of counts in 1-itemset and 2-itemset has replaced the value used in Pheromone and Heuristic (Matrix) in the pheromone formula.

Lift measurement [10] method is introduced in Apriori-Ant algorithm to replace Confidence measurement in Apriori algorithm. This method proves that it has no bias and duplication of combined items. For example: X & Y or Y & X makes no difference and produces the same value after calculation. So either one combination can be removed from the list to improve processing speed.

The use of ACO algorithm is to shorten the rule generation processing time. Rule generation is based on pheromone value and pheromone value is generated after a pheromone update and a pheromone evaporation process. This is different from what is in Apriori algorithm where the rules are formed after a series of repetitive candidates generations.

In Apriori algorithm, infrequent items are pruned from the list which may result in some interesting rules being left out. But in Apriori-Ant algorithm, no rules are pruned. Which means, at the early stage, all items are treated equal even if they are infrequent.

Conclusion:

First, Apriori-Ant algorithm introduces a new way to replace the candidates generation process which requires repetitive I/O scans over the database. In Apriori-Ant algorithm, only two I/O disk scans are required and only joining process in the candidates generation process occurs up to level-2.

Second, Apriori-Ant algorithm should produce better rules from a non-transactional data as compared to Apriori algorithm because no infrequent items which may be interesting are wasted [4].

Third, Apriori-Ant algorithm introduces the Lift measurement [10] method to calculate the pheromone value which replaces the Confidence measurement in ACO algorithm. The advantage is that it does not suffer from the rare items problem.

Appendix A

No	Age	Gender	Area	Cause	Per	Deg	Description

Medication:
Discharge:
Dressing:
Ointment/Cream:

Appendix B

			Cycle 1		Cycle 2		Cycle 3		Cycle 4		Cycle 5	
No	2-ItemSet	P	U	E	U	E	U	E	U	E	U	E
1	LowerLimb & UpperLimb	3.030298	x	3.027268	x	3.024241	x	3.021216	x	3.018195	3.019195	3.016176
2	LowerLimb & Paracetamol	4.545455	x	4.540909	x	4.536368	x	4.531832	x	4.5273	4.650068	4.645418
3	LowerLimb & Silver Sulfadiazine	4.545455	x	4.540909	x	4.536368	x	4.531832	x	4.5273	4.715303	4.710587
4	LowerLimb & Hydrocolloid	3.030298	x	3.027268	x	3.024241	x	3.021216	x	3.018195	3.207807	3.204599
5	LowerLimb & Silver Sulfadiazine Cream	4.545455	x	4.540909	x	4.536368	x	4.531832	x	4.5273	4.657612	4.652955
6	UpperLimb & Paracetamol	3.030298	x	3.027268	x	3.024241	x	3.021216	x	3.018195	x	3.015177
7	UpperLimb & Silver Sulfadiazine	3.030298	x	3.027268	x	3.024241	x	3.021216	x	3.018195	x	3.015177
8	UpperLimb & Hydrocolloid	3.030298	x	3.027268	x	3.024241	x	3.021216	x	3.018195	x	3.015177
9	UpperLimb & Silver Sulfadiazine Cream	3.030298	x	3.027268	x	3.024241	x	3.021216	x	3.018195	x	3.015177
10	Paracetamol & Silver Sulfadiazine	4.545455	x	4.540909	4.664	4.659336	x	4.654676	x	4.650022	x	4.645372
11	Paracetamol & Hydrocolloid	3.030298	x	3.027268	3.214828	3.211613	x	3.208401	x	3.205193	x	3.201988
12	Paracetamol & Silver Sulfadiazine Cream	4.545455	x	4.540909	4.671502	4.666831	x	4.662164	x	4.657502	x	4.652844
13	Silver Sulfadiazine & Hydrocolloid	3.030298	x	3.027268	x	3.024241	x	3.021216	3.208703	3.205494	x	3.202289
14	Silver Sulfadiazine & Silver Sulfadiazine Cream	4.545455	x	4.540909	x	4.536368	x	4.531832	4.66218	4.657518	x	4.65286
15	Hydrocolloid & Silver Sulfadiazine Cream	3.030298	x	3.027268	x	3.024241	x	3.021216	x	3.018195	x	3.015177
16	LowerLimb & Phenergan	3.030298	x	3.027268	x	3.024241	x	3.021216	x	3.018195	3.139923	3.136783
17	LowerLimb & Askina	3.030298	x	3.027268	x	3.024241	x	3.021216	x	3.018195	3.144792	3.141647
18	LowerLimb & Vaseline Gauze	3.030298	x	3.027268	x	3.024241	x	3.021216	x	3.018195	3.144987	3.141842
19	Paracetamol & Phenergan	3.030298	x	3.027268	3.149359	3.146209	x	3.143063	x	3.13992	x	3.13678
20	Paracetamol & Askina	3.030298	x	3.027268	3.154242	3.151088	x	3.147937	x	3.144789	x	3.141644
21	Paracetamol & Vaseline Gauze	3.030298	x	3.027268	3.154438	3.151283	x	3.148132	x	3.144984	x	3.141839
22	Phenergan & Silver Sulfadiazine	3.030298	x	3.027268	x	3.024241	x	3.021216	x	3.018195	x	3.015177
23	Phenergan & Askina	3.030298	x	3.027268	x	3.024241	x	3.021216	x	3.018195	x	3.015177
24	Phenergan & Vaseline Gauze	3.030298	x	3.027268	x	3.024241	x	3.021216	x	3.018195	x	3.015177
25	Phenergan & Silver Sulfadiazine Cream	3.030298	x	3.027268	x	3.024241	x	3.021216	x	3.018195	x	3.015177
26	Silver Sulfadiazine & Askina	3.030298	x	3.027268	x	3.024241	x	3.021216	3.143065	3.139922	x	3.136782
27	Silver Sulfadiazine & Vaseline Gauze	3.030298	x	3.027268	x	3.024241	x	3.021216	3.147939	3.144791	x	3.141646
28	Askina & Vaseline Gauze	3.030298	3.15251	3.149358	x	3.146208	x	3.143062	x	3.139919	x	3.136779
29	Askina & Silver Sulfadiazine Cream	3.030298	3.158399	3.15524	x	3.152085	x	3.148933	x	3.145784	x	3.142638
30	Vaseline Gauze & Silver Sulfadiazine Cream	3.030298	x	3.027268	x	3.024241	3.152324	3.149172	x	3.146023	x	3.142876

Note: P-Pheromone value, U-Pheromone UpDate, E-Pheromone Evaporate

Appendix C

			Cycle 6		Cycle 7		Cycle 8		Cycle 9		Cycle 10	
No	2-ItemSet	Pr	U	E	U	E	U	E	U	E	U	E
1	LowerLimb & UpperLimb	3.016176	x	3.01316	x	3.010147	x	3.007136	x	3.004129	x	3.001125
2	LowerLimb & Paracetamol	4.645418	x	4.640772	x	4.636132	x	4.631495	x	4.626864	x	4.622237
3	LowerLimb & Silver Sulfadiazine	4.710587	x	4.705877	x	4.701171	x	4.69647	x	4.691773	x	4.687082
4	LowerLimb & Hydrocolloid	3.204599	x	3.201395	x	3.198193	x	3.194995	x	3.1918	x	3.188608
5	LowerLimb & Silver Sulfadiazine Cream	4.652955	x	4.648302	x	4.643653	x	4.63901	x	4.634371	x	4.629736
6	UpperLimb & Paracetamol	3.015177	x	3.012162	x	3.00915	x	3.00614	x	3.003134	x	3.000131
7	UpperLimb & Silver Sulfadiazine	3.015177	x	3.012162	x	3.00915	x	3.00614	x	3.003134	x	3.000131
8	UpperLimb & Hydrocolloid	3.015177	x	3.012162	x	3.00915	x	3.00614	x	3.003134	x	3.000131
9	UpperLimb & Silver Sulfadiazine Cream	3.015177	x	3.012162	x	3.00915	x	3.00614	x	3.003134	x	3.000131
10	Paracetamol & Silver Sulfadiazine	4.645372	x	4.640726	x	4.636086	x	4.63145	x	4.626818	4.748944	4.744195
11	Paracetamol & Hydrocolloid	3.201988	x	3.198786	x	3.195587	x	3.192391	x	3.189199	3.380157	3.376777
12	Paracetamol & Silver Sulfadiazine Cream	4.652844	x	4.648191	x	4.643543	x	4.6389	x	4.634261	4.771467	4.766696
13	Silver Sulfadiazine & Hydrocolloid	3.202289	x	3.199086	x	3.195887	x	3.192691	3.379247	3.375868	x	3.372492
14	Silver Sulfadiazine & Silver Sulfadiazine Cream	4.65286	x	4.648207	x	4.643559	x	4.638916	4.776086	4.771309	x	4.766538
15	Hydrocolloid & Silver Sulfadiazine Cream	3.015177	x	3.012162	x	3.00915	x	3.00614	x	3.003134	x	3.000131
16	LowerLimb & Phenergan	3.136783	x	3.133646	x	3.130513	x	3.127382	x	3.124255	x	3.12113
17	LowerLimb & Askina	3.141647	x	3.138506	x	3.135367	x	3.132232	x	3.1291	x	3.12597
18	LowerLimb & Vaseline Gauze	3.141842	x	3.1387	x	3.135561	x	3.132426	x	3.129293	x	3.126164
19	Paracetamol & Phenergan	3.13678	x	3.133643	x	3.13051	x	3.127379	x	3.124252	3.250423	3.247173
20	Paracetamol & Askina	3.141644	x	3.138503	x	3.135364	x	3.132229	x	3.129096	3.260113	3.256853
21	Paracetamol & Vaseline Gauze	3.141839	x	3.138697	x	3.135558	x	3.132423	x	3.12929	3.260695	3.257434
22	Phenergan & Silver Sulfadiazine	3.015177	x	3.012162	x	3.00915	x	3.00614	x	3.003134	x	3.000131
23	Phenergan & Askina	3.015177	x	3.012162	x	3.00915	x	3.00614	x	3.003134	x	3.000131
24	Phenergan & Vaseline Gauze	3.015177	x	3.012162	x	3.00915	x	3.00614	x	3.003134	x	3.000131
25	Phenergan & Silver Sulfadiazine Cream	3.015177	x	3.012162	x	3.00915	x	3.00614	x	3.003134	x	3.000131
26	Silver Sulfadiazine & Askina	3.136782	x	3.133645	x	3.130512	x	3.127381	3.248627	3.245378	x	3.242133
27	Silver Sulfadiazine & Vaseline Gauze	3.141646	x	3.138505	x	3.135366	x	3.132231	3.263176	3.259913	x	3.256653
28	Askina & Vaseline Gauze	3.136779	x	3.133642	x	3.130509	3.256923	3.253666	x	3.250413	x	3.247162
29	Askina & Silver Sulfadiazine Cream	3.142638	x	3.139496	x	3.136356	3.268633	3.265364	x	3.262099	x	3.258837
30	Vaseline Gauze & Silver Sulfadiazine Cream	3.142876	3.270582	3.267311	3.394891	3.391496	x	3.388105	x	3.384717	x	3.381332

Note: Pr-Previous Pheromone value in Cycle 5, U-Pheromone UpDate, E-Pheromone Evaporate

Appendix D

			Cycle 11		Cycle 12		Cycle 13		Cycle 14		Cycle 15	
No	2-ItemSet	Pr	U	E	U	E	U	E	U	E	U	E
1	LowerLimb & UpperLimb	3.001125	3.002125	2.999123	x	2.996124	2.997124	2.994127	x	2.991133	x	2.988142
2	LowerLimb & Paracetamol	4.622237	4.744322	4.739578	x	4.734838	4.856723	4.851866	x	4.847015	x	4.842168
3	LowerLimb & Silver Sulfadiazine	4.687082	4.878854	4.873976	x	4.869102	5.06537	5.060305	x	5.055245	x	5.05019
4	LowerLimb & Hydrocolloid	3.188608	3.384763	3.381378	x	3.377997	3.581611	3.57803	x	3.574452	x	3.570877
5	LowerLimb & Silver Sulfadiazine Cream	4.629736	4.767127	4.76236	x	4.757597	4.902862	4.897959	x	4.893061	x	4.888168
6	UpperLimb & Paracetamol	3.000131	x	2.997131	x	2.994134	x	2.99114	x	2.988149	x	2.985161
7	UpperLimb & Silver Sulfadiazine	3.000131	x	2.997131	x	2.994134	x	2.99114	x	2.988149	x	2.985161
8	UpperLimb & Hydrocolloid	3.000131	x	2.997131	x	2.994134	x	2.99114	x	2.988149	x	2.985161
9	UpperLimb & Silver Sulfadiazine Cream	3.000131	x	2.997131	x	2.994134	x	2.99114	x	2.988149	x	2.985161
10	Paracetamol & Silver Sulfadiazine	4.744195	x	4.73945	x	4.734711	x	4.729976	x	4.725246	x	4.720521
11	Paracetamol & Hydrocolloid	3.376777	x	3.3734	x	3.370026	x	3.366656	x	3.36329	x	3.359927
12	Paracetamol & Silver Sulfadiazine Cream	4.766696	x	4.761929	x	4.757167	x	4.75241	x	4.747657	x	4.74291
13	Silver Sulfadiazine & Hydrocolloid	3.372492	x	3.36912	x	3.365751	x	3.362385	x	3.359023	x	3.355664
14	Silver Sulfadiazine & Silver Sulfadiazine Cream	4.766538	x	4.761772	x	4.75701	x	4.752253	x	4.747501	x	4.742753
15	Hydrocolloid & Silver Sulfadiazine Cream	3.000131	x	2.997131	x	2.994134	x	2.99114	x	2.988149	x	2.985161
16	LowerLimb & Phenergan	3.12113	3.242136	3.238894	x	3.235655	3.35642	3.353064	x	3.349711	x	3.346361
17	LowerLimb & Askina	3.12597	3.256656	3.253399	x	3.250146	3.385403	3.382017	x	3.378635	x	3.375257
18	LowerLimb & Vaseline Gauze	3.126164	3.25743	3.254173	x	3.250919	3.387335	3.383947	x	3.380563	x	3.377183
19	Paracetamol & Phenergan	3.247173	x	3.243926	x	3.240682	x	3.237441	x	3.234204	x	3.23097
20	Paracetamol & Askina	3.256853	x	3.253596	x	3.250343	x	3.247092	x	3.243845	x	3.240602
21	Paracetamol & Vaseline Gauze	3.257434	x	3.254177	x	3.250922	x	3.247672	x	3.244424	x	3.241179
22	Phenergan & Silver Sulfadiazine	3.000131	x	2.997131	x	2.994134	x	2.99114	3.123047	3.119924	x	3.116804
23	Phenergan & Askina	3.000131	x	2.997131	x	2.994134	x	2.99114	3.117062	3.113945	x	3.110831
24	Phenergan & Vaseline Gauze	3.000131	x	2.997131	x	2.994134	x	2.99114	3.116822	3.113705	x	3.110592
25	Phenergan & Silver Sulfadiazine Cream	3.000131	x	2.997131	x	2.994134	x	2.99114	3.117813	3.114695	x	3.11158
26	Silver Sulfadiazine & Askina	3.242133	x	3.238891	x	3.235652	x	3.232416	x	3.229184	x	3.225954
27	Silver Sulfadiazine & Vaseline Gauze	3.256653	x	3.253396	x	3.250143	x	3.246893	x	3.243646	x	3.240402
28	Askina & Vaseline Gauze	3.247162	x	3.243915	x	3.240671	x	3.237431	x	3.234193	3.364939	3.361574
29	Askina & Silver Sulfadiazine Cream	3.258837	x	3.255578	x	3.252322	x	3.24907	x	3.245821	3.382419	3.379036
30	Vaseline Gauze & Silver Sulfadiazine Cream	3.381332	x	3.377951	3.510174	3.506664	x	3.503157	x	3.499654	x	3.496154

Note: Pr-Previous Pheromone value in Cycle 10, U-Pheromone UpDate, E-Pheromone Evaporate

Appendix E

			Cycle 16		Cycle 17		Cycle 18		Cycle 19		Cycle 20	
No	2-ItemSet	Pr	U	E	U	E	U	E	U	E	U	E
1	LowerLimb & UpperLimb	2.988142	x	2.985153	2.986153	2.983167	x	2.980184	x	2.977204	x	2.974227
2	LowerLimb & Paracetamol	4.842168	x	4.837325	4.958771	4.953813	x	4.948859	x	4.94391	x	4.938966
3	LowerLimb & Silver Sulfadiazine	5.05019	x	5.045139	5.24549	5.240245	x	5.235005	x	5.229769	x	5.22454
4	LowerLimb & Hydrocolloid	3.570877	x	3.567306	3.778126	3.774348	x	3.770573	x	3.766803	x	3.763036
5	LowerLimb & Silver Sulfadiazine Cream	4.888168	x	4.88328	5.036405	5.031368	x	5.026337	x	5.021311	x	5.016289
6	UpperLimb & Paracetamol	2.985161	x	2.982175	x	2.979193	x	2.976214	x	2.973238	x	2.970265
7	UpperLimb & Silver Sulfadiazine	2.985161	x	2.982175	x	2.979193	x	2.976214	x	2.973238	x	2.970265
8	UpperLimb & Hydrocolloid	2.985161	x	2.982175	x	2.979193	x	2.976214	x	2.973238	x	2.970265
9	UpperLimb & Silver Sulfadiazine Cream	2.985161	x	2.982175	x	2.979193	x	2.976214	x	2.973238	x	2.970265
10	Paracetamol & Silver Sulfadiazine	4.720521	4.841927	4.837085	x	4.832248	x	4.827416	x	4.822589	x	4.817766
11	Paracetamol & Hydrocolloid	3.359927	3.554604	3.551049	x	3.547498	x	3.54395	x	3.540407	x	3.536866
12	Paracetamol & Silver Sulfadiazine Cream	4.74291	4.887094	4.882207	x	4.877325	x	4.872447	x	4.867575	x	4.862707
13	Silver Sulfadiazine & Hydrocolloid	3.355664	x	3.352308	x	3.348956	3.545049	3.541504	x	3.537962	x	3.534424
14	Silver Sulfadiazine & Silver Sulfadiazine Cream	4.742753	x	4.73801	x	4.733272	4.877074	4.872197	x	4.867325	x	4.862458
15	Hydrocolloid & Silver Sulfadiazine Cream	2.985161	x	2.982175	x	2.979193	x	2.976214	x	2.973238	3.169931	3.166761
16	LowerLimb & Phenergan	3.346361	x	3.343014	3.463301	3.459838	x	3.456378	x	3.452922	x	3.449469
17	LowerLimb & Askina	3.375257	x	3.371881	3.511413	3.507902	x	3.504394	x	3.50089	x	3.497389
18	LowerLimb & Vaseline Gauze	3.377183	x	3.373806	3.515262	3.511747	x	3.508235	x	3.504727	x	3.501222
19	Paracetamol & Phenergan	3.23097	3.367057	3.36369	x	3.360326	x	3.356966	x	3.353609	x	3.350255
20	Paracetamol & Askina	3.240602	3.376284	3.372908	x	3.369535	x	3.366165	x	3.362799	x	3.359436
21	Paracetamol & Vaseline Gauze	3.241179	3.377231	3.373854	x	3.37048	x	3.367109	x	3.363742	x	3.360378
22	Phenergan & Silver Sulfadiazine	3.116804	x	3.113687	x	3.110573	x	3.107463	x	3.104355	x	3.101251
23	Phenergan & Askina	3.110831	x	3.10772	x	3.104612	x	3.101508	x	3.098406	x	3.095308
24	Phenergan & Vaseline Gauze	3.110592	x	3.107481	x	3.104374	x	3.101269	x	3.098168	x	3.09507
25	Phenergan & Silver Sulfadiazine Cream	3.11158	x	3.108469	x	3.10536	x	3.102255	x	3.099153	x	3.096053
26	Silver Sulfadiazine & Askina	3.225954	x	3.222728	x	3.219506	3.34472	3.341375	x	3.338034	x	3.334696
27	Silver Sulfadiazine & Vaseline Gauze	3.240402	x	3.237162	x	3.233924	3.368713	3.365345	x	3.361979	x	3.358617
28	Askina & Vaseline Gauze	3.361574	x	3.358212	x	3.354854	x	3.351499	3.487113	3.483626	x	3.480142
29	Askina & Silver Sulfadiazine Cream	3.379036	x	3.375657	x	3.372281	x	3.368909	3.510394	3.506883	x	3.503376
30	Vaseline Gauze & Silver Sulfadiazine Cream	3.496154	x	3.492658	x	3.489165	x	3.485676	x	3.482191	x	3.478708

Note: Pr-Previous Pheromone value in Cycle 15, U-Pheromone UpDate, E-Pheromone Evaporate

Appendix F

Pseudocode for Single Itemset

```
//to create one-itemset list/single-itemset list

While singleitem aLoop        //first loop for single item set
        While singleitem bLoop        //second loop for single item set

                //substitute the items in itemjoin into temp variable
                temp=itemjoin[aLoop][bLoop];
                        If (temp!="-")
                                bool;
                //substitute the items in SupportItem into temp variable
                While singleitem cLoop        //third loop for single item set
                        If (temp=SupportItem[cLoop])
                                break;
                end cLoop
                //If not the same, substitute the items in temp variable into SupportItem
                //array.
                If(!same)
                        SupportItem[s]=temp;
                increase s;
        end While singleitem bLoop
end While singleitem aLoop
```

Appendix G

Pseudocode for single itemset count

```
//to calculate the items in one-itemset list

While singlecount aLoop
        While singlecount bLoop
                //substitute the item in itemjoin into temp variable
                temp=itemjoin[aLoop][bLoop];
                While singlecount cLoop
                        //if the items in temp variable same as in SupportItem loop, then
                        //increase SupportCnt with 1
                        //the purpose is to calculate the count with the respective items in
                        //SupportItem array
                        If(temp=SupportItem[cLoop])
                                increase SupportCnt[cLoop];
                end While singlecount cLoop
        end While singlecount bLoop
end While singlecount aLoop
```

Appendix H

Pseudocode for two itemset

```
//to create two-itemset list

While twoitem aLoop
    While twoitem bLoop
        //substitute items at itemjoin into temp1 variable
        temp1=itemjoin[aLoop][bLoop];
        while twoitem dLoop
            //substitute items at itemjoin into temp2 variable
            temp2=itemjoin[aLoop][bLoop];
            if ((temp1!="-") and (temp2!= "-"))
            bool same=false;
            //checking if items in temp1 and temp2 exist in ConfidenceItem
            //array
            While twoitem cLoop
                if((temp1=ConfidenceItem[cLoop][0]) and
                  (temp2=ConfidenceItem[cLoop][1]))
                same=true;
                break;
            end while twoitem cLoop
            //if not the same, then substitute the items in temp1 into
            //ConfidenceItem array at position 0 and substitute the items in
            //temp2 into ConfidenceItem array at position 1.
            if(!same)
            ConfidenceItem[st][0]=temp1;
            Confidenceitem[st][1]=temp2;
            increase st;
        end while twoitem dLoop
    end while twoitem bLoop
end while twoitem aLoop
```

Appendix I

Pseudocode for two itemset count

```
//to calculate the items in two-itemset list


While twoitemcount aLoop
    While twoitemcount bLoop
        //substitute items in itemjoin into temp1 variable
        temp1=itemjoin[aLoop][bLoop];
        while twoitemcount dLoop
            //substitute items in itemjoin into temp1 variable
            temp2=itemjoin[aLoop][dLoop];
            while twoitemcount cLoop
                //if the items in temp1 and temp2 variable exist in
                //itemjoin array then increase ConfidenceCnt with 1
                //it is to calculate the count with the respective items in
                //itemjoin array.
                if((temp1=itemjoin[cLoop][0]) and
                  (temp2=itemjoin[cLoop][1]))
                increase ConfidenceCnt[cLoop];
            end while twoitemcount cLoop
        end while twoitemcount dLoop
    end while twoitemcount bLoop
end while twoitemcount aLoop
```

Appendix J

Pseudocode or Apriori Internal Calculation

```
//Apriori calculation

While aLift Loop
    For bLift Loop
        //only if the items at ConfidenceItem at position 0 equal to
        //SupportItem, xCount calculation is get from the SupportCnt divided
        //by the total cause
        If(ConfidenceItem[aLift][0]=SupportItem[bLift])
            xCount=SupportCnt[bLift]/totalcause;

        //only if the items at ConfidenceItem at position 1 equal to
        //SupportItem, yCount calculation is get from the SupportCnt divided
        //by the total cause
        If(ConfidenceItem[aLift][1]=SupportItem[bLift])
            yCount=SupportCnt[bLift]/totalcause;

        //if ConfidenceCnt value is not equal to zero(O), then perform the
        //ProConf, Conf, Lift and Phe calculation
        if(ConfidenceCnt[aLift]!=0)
            ProConf;
            Conf;
            Lift;
```

Phe[aLift];

end For Loop

end While Loop

Appendix K

Generate random item to match in 2-itemset list

```
//generate a random item

Initialize number of Ant cycles loop
        Generate a random number [RanNum];
                TempCharRan=SupporItem[RanNum];
                        While Check<st+1

                                //check whether the random items exist in ConfidenceItem
                                If ConfidenceItem[Check][0]=TempCharRan
                                        Checking=true;
                                        break;
                                Else
                                        increase Check;
                        End while check

                //if random items not found in ConfidenceItem, get another random
                //item
                If(!checking)
                        Get another random number;

                //if random items found in ConfidenceItem, substitute the random item
                //into CharRan variable.
                If(checking)
                        If TempCharRan=""
                                Get another random number;
                        Else
                                CharRan=TempCharRan;

//Create a Tabu List to store the existed items in the process
                Create Tabu array;
                While Ant LoopA<st+1
                        If ConfidenceItem[Ant LoopA][0]=CharRan
                                If Pheromone[Ant LoopA]>PherTemp
                                        PherTemp=Pheromone[Ant LoopA];
                                End if
                                Pher LoopA = Ant LoopA;
                        End If
                        increase Ant LoopA;
                End While Ant LoopA

                CharRanAnt=ConfidenceItem[Pher LoopA][1];
                //items in position 0 will store in Tabu List
                Tabu[t]=ConfidenceItem[Pher LoopA][0];
```

```
//pheromone values for the respective Tabu items are store in TempPher
variable
TempPher[TP]=Pheromone[Pher LoopA];
increase TP;
increase t;
For Ant AlgoLoop<10
        While Ant LoopB<st+1
                Flag1, Flag2=false;
                If ConfidenceItem[Ant LoopB][0]=CharRanAnt
                        Flag0=true;
                        For TabuLoop<t
                                If Tabu[TabuLoop]=CharRanAnt
                                        Flag1=true;
                                        break;
                                If (!Flag1)
                                        For TabuLoop<t

                                        If ConfidenceItem[Ant
                                                LoopB][0]=Tabu[TabuLoop]
                                        Flag2=true;
                                        break;
                        End For TabuLoop
                        If (!Flag2)
                                //if the pheromone values greater than
                                //PherTempB, substitute the pheromone
                                //values into PerTempB variable
                                If Pheromone[Ant LoopB]>PherTempB
                                        PherTempB=Pheromone[Ant LoopB];
                                        PherLoopB=Ant LoopB;
                                End If
                                PherLoopB=Ant LoopB;
                                Flag0=true;
                        End If
                        End If
                End If
                Increase Ant LoopB;
                If (Flag1 | |Flag2)
                        Break;
                End while Ant LoopB
        End For AntAlgoLoop


//Pheromone Update and Evaporation
    For PherUpdate<st+1
        If ConfidenceItem[PheUpdate][0]=TempCharRan
            For CheckUpdate<s
                If ConfidenceItem[PherUpdate][1]
                        =SupportItem[CheckUpdate]
                    Pheromone Calculation;
```

```
                End If
            End For CheckUpdate

            For Evaporate<st+1
                Evaporate calculation;
            End For Evaporate
        End If
    End for PherUpate
End for Ant Cycles Loop
```

Appendix L

Sorting the items with highest pheromone values

```
For Empty Loop<st+1
        TempPher[Empty]=0.0;
        TempConfidenceItem[empty][0]="";
        TempConfidenceItem[empty][1]="";
End For Empty Loop


//substitute all the necessary items into a temporary variable

For Change Loop<st+1
        TempPher[Change]=Pheromone[Change];
        TempConfidenceItem[Change][0]=ConfidenceItem[Change][0];
        TempConfidenceItem[Change][1]=ConfidenceItem[Change][1];
End For Change Loop


//sorting out all the items according to pheromone value in ascending order

For Sort iLoop<=number
        For Sort jLoop<=number+1
                If TempPher[Sort iLoop]<TempPher[Sort jLoop]
                        ChangePher=TempPher[Sort iLoop];
                        TempPher[Sort iLoop]=TempPher[Sort jLoop];
                        TempPher[Sort jLoop]=ChangePher;

                        ChangeConfidence1=TempConfidenceItem[Sort iLoop][0];
                        TempConfidenceItem[Sort iLoop][0]=
                                                TempConfidenceItem[Sort jLoop][0];
                        TempConfidenceItem[Sort jLoop][0]=ChangeConfidence1;

                        ChangeConfidence2=TempConfidenceItem[Sort iLoop][1];
                        TempConfidenceItem[Sort iLoop][1]=
                                                TempConfidenceItem[Sort jLoop][1];
                        TempConfidenceItem[Sort jLoop][1]=ChangeConfidence2;
                End If
        End For Sort jLoop
End For Sort iLoop
```

Appendix M

Rule Generation

```
Create RuleConstruction array;

For TestRule<number of rules for testing
        RuleTemp=TempConfidenceItem[TestRule][1];
        TabuRule[cTabu]=TempConfidenceItem[TestRule][0];
        increase cTabue;

        //join only the first 2 items in TempConfidenceItem into RuleConstruction
        //array
        RuleConstruction[cRule]=TempConfidenceItem[0]+RuleTemp;

        For SecondRuleLoop<10
                While Rule Loop2<st+1
                        RuleFlage1, RuleFlag2 = false;
                        If TempConfidenceItem[Rule Loop2][0]=RuleTemp
                                RuleFlage0=true;
                                For RuleTabuLoop<cTabu

                                        //RuleConstruction process will stop if the items
                                        //exist in Tabu list.
                                        If TabuRule[RuleTabuLoop]=RuleTemp
                                                RuleFlag1=true;
                                                break;
                                        End If
                                End For RuleTabuLoop

                                If (!RuleFlag1)
                                For RuleTabuLoop<cTabu

                                        //Items at position O to find the items at position 1
                                        //with higher pheromone values.
                                        If TempConfidenceItem[Rule Loop2][0]=
                                                                TabuRule[RuleTabuLoop]
                                                RuleFlag2=true;
                                                break;
                                        End If
                                End For RuleTabuLoop

                                If (!RuleFlag2)
                                        If TempPher[Rule Loop2]>PherUpdate2
                                                PherUpdate2=TempPher[Rule Loop2];
                                                Rule LoopB=Rule Loop2;
                                        End If
                                        RuleFlag0=true;
```

```
                End If
                increase Rule Loop2;
                End If
            End If
            If (RuleFlag0)

                //store only the items at position 1 into RuleConstruction
                //array
                RuleTemp=TempConfidenceItem[Rule LoopB][1];
                TabuRule[cTabu]=TempConfidenceItem[Rule LoopB][0];
                increase cTabu;
                RuleConstruction[cTabu]=
                                    RuleConstruction[cRule]+RuleTemp;

            End If
            If (RuleFlag1||RuleFlag2)
                break;
        End For
increase cTabu;
```

References

1. Agrawal R. & Srikant R., *Fast Algorithm for Mining Association Rules* [Online], Available: http://www.cs.duke.edu/~junyang/courses/cps296.1-2002-spring/papers/AS-VLDB94.pdf [05.07.2005]

2. Cerin C., Gay J., Mahec L. G. & Koskas M., *Efficient Data-Structures and Parallel Algorithm for Association Rules Discovery* [Online], Available: http://www.laria.u-picardie.fr/~cerin/documents/cerin_c_enc04.pdf [26.07.2004]

3. Craw S., *Applications of Data Mining - Association Rules*, 2000 [Online], Available: http://www.comp.rgu.ac.uk/staff/smc/teaching/datamining/apriori-lab/ [10.10.2004]

4. Doxygen, *An efficient implemenation of MSAPRIORI Algorithm: Frequent Itemset Mining problem* 2004, [Online] Available: http://www.cs.bme.hu/~bodon/en/fim/msapriori/Documentation/html/ [24.11.2004]

5. Drozdek A. (2000), *Data Structures and Algorithm in C++, second edition.* Brooks/Cole Thomson Learning. ISBN:0-534-37597-9

6. El-Hajj M. & Osmar R.Zaiane, *COFI-tree Mining: A New Approach to Pattern Growth with Reduced Candidacy Generation* [Online], Available: http://ceur-ws.org/vol-90/elhajj.pdf [05.07.2005]

7. Fjalldal B. J., *An Introduction to Ant Colony Algorithm*, 1999 [Online], Available: http://www.informatics.susx.ac.uk/research/nlp/gazdar/teach/atc/1999/web/johannf/index.html [18.09.2004]

8. Gagne C., Gravel M. & Wilson L. Price, *A look-ahead addition to the ant colony optimization metaheuristics and its application to an industrial scheduling problem* [Online], Available: http://dimcom.uqac.uquebec.ca/~mgravel/mic-abs-066.pdf [25.06.2004]

9. Gambardella M. L. & Dorigo M., *Solving Symmetric and Asymmetrics TSPs by Ant Colonies* [Online], Available: http://www.idsia.ch/~luca/icec96-acs.pdf [30.09.2004]

10. Hahsler M., *A Comparison of Commonly Used Interest Measures for Association Rules*, 2004 [Online], Available: http://wwwai.wu-wien.ac.at/~hahsler/research/association_rules/measures.html#lift [10.10.2004]

11. Hand D., Mannila H. & Smyth P. (2001), *Principles of Data Mining*, A Bradford book MIT Press. ISBN: 0-262-08290-X.

12. Ian H. Witten & Frank E. (2000), *Data Mining, Practical Machine Learning tools and Techniques with Java Implementation*, Morgan Kaufmann Publishers.

13. J. Hau, J. Pei and Y. Yin, *Mining Frequent Pattern candidate generation.* In ACM-SIGMOD, Dallas, 2000.

14. James P. Cohoon & Jack W. Davidson (1999), *C++ Program Design, An Introduction to Programming and Object-Oriented Design, second edition*. MacGraw-Hill International Editions, Computer Science Series. ISBN:0-07-116147-3

15. Kolesnikov A., *Polygonal approximation*, 2003 [Online], Available: http://www.cs.joensuu.fi/~koles/approximation/Ch3_2.html [16.09.2004]

16. L. Bo, Hussein A. Abbass & McKay B., *Classification Rule Discovery with Ant Colony Optimization* [Online], Available: http://www.comp.hkbu.edu.hk/~cib/2004/feb/cib_vol3no1_article4.pdf [17.06.2004]

17. Osmar R.Zaiane, El-Hajj M. & L. Paul, *Fast Parallel Association Rule Mining without Candidacy Generation* [Online], Available: http://www.cs.ualberta.ca/~zaiane/postscript/icd.01.pdf [05.07.2005]

18. Rafael S. Parpinelli, Heitor S. Lopes & Alex A. Freitas, *Data Mining with an Ant Colony Optimization Algorithm* [Online], Available: http://www.ppgia.pucpr.br/~alex/pub_papers.dir/Ant-IEEE-TEC.pdf [31.10.2004]

19. Relue R. & Xindong Wu, *Rule Generation with the Pattern Repository*[Online], Available: http://www.cs.uvm.edu/tr/Papers/cs-02-10.pdf [05.07.2005]

20. Wagner S., Affenzeller M. & Ibrahim K. I., *Agent-Based problem solving: The Ant Colonies Metaphor* [Online], Available: http://www.heuristiclab.com/publications/papers/wagner03a.pdf [30.09.2004]