Musical Data Mining Pattern Matching Apriori and DHP Algorithm

J.James Alaguraja
Assistant Professor, Department of Computer Application, J.P. College of Arts and Science, Manonmaniam Sundarnar University, Tirunelveli , India,

ABSTRACT
Musical data mining is not a new invention, but as a nation-wide resource of this type it breaks new ground by providing researchers with new ways to analyze musical data. It was Toiviainen and Eerola´s idea to combine specific information with a geographical coordinate database. Now geographical comparisons can be made it is possible to follow the geographical variation of musical features. For instance, schools can now identify and trace folk tune originating from their own regions. Musical Data Mining is used for discovering any kind of relevant similarity between music titles. Several algorithms like Apriori, PHP, partition, sampling and some other parallel algorithm have been developed. In this thesis, Apriori and DHP are implemented. To extract the similarity between music titles and to manipulate their relationships two techniques are used co-occurrence analysis and correlation analysis. By the use of these two techniques it is capable to access the database and then find whether any similarity exist between the music titles. For the purpose of finding a match within the titles in the database Pattern matching is used using the Apriori and DHP algorithms

1. INTRODUCTION
Pattern matching is the act of checking for the presence of the constituents of a given pattern. It is used to check that things have the desired structure to find the relevant structure, to retrieve the aligning parts and to substitute the matching part with something else. Patterns are often described using regular expressions (i.e. Backtracking) and matched using respective algorithms.

2. APRIORI ALGORITHM
In data mining, Apriori is a classic algorithm for learning association rules. Apriori is designed to operate on databases containing transactions (for example, collections of items bought by customers, or details of a website frequentation). Other algorithms are designed for finding association rules in data having no transactions or having no timestamps (DNA sequencing).

As is common in association rule mining, given a set of itemsets (for instance, sets of retail transactions each listing individual items purchased), the algorithm attempts to find subsets which are common to at least a minimum number C (the cutoff, or confidence threshold) of the itemsets. Apriori uses a "bottom up" approach, where frequent subsets are extended one item at a time (a step known as candidate generation, and groups of candidates are tested against the data. The algorithm terminates when no further successful extensions are found

Apriori uses breadth-first search and a hash tree structure to count candidate item sets efficiently. It generates candidate item sets of length k from item sets of length $k - 1$. Then it prunes the candidates which have an infrequent sub pattern. According to the downward closure lemma, the candidate set contains all frequent k-length item sets. After that, it scans the transaction database to determine frequent item sets among the candidates. For determining frequent items quickly, the algorithm uses a hash tree to store candidate itemsets. Apriori, while historically significant, suffers from a number of inefficiencies or trade-offs, which have spawned other algorithms. Candidate generation generates large numbers of subsets (the algorithm attempts to load up the candidate set with as many as possible before each scan). Bottom-up subset exploration (essentially a breadth-first traversal of the subset lattice) finds any maximal subset S only after all $2 | S | - 1$ of its proper subsets.

Apriori is an influential algorithm for mining frequent itemsets for Boolean association rules. The name of the algorithm is based on the fact that the algorithm uses prior knowledge of frequent itemset properties, as it is shown below. Apriori employs an iterative approach known as a level-wise search, where k-itemsets are used to explore (k + 1)-itemsets. First, the set of frequent 1-itemsets is found. This set is denoted L1. L1 is used to find L2, the set of frequent 2-itemsets, which is used to find L3, and so on, until no more frequent k-itemsets can be found. The finding of each Lk requires one full scan of the database.

To improve the efficiency of the level-wise generation of frequent itemsets, an important property called the Apriori property, presented below, is used to reduce the search space. At first this property is described and an example is shown to illustrate it.

In order to use the Apriori property, all nonempty subsets of a frequent itemset must also be frequent. This property is based on the following observation. By definition, if an itemset I does not satisfy the minimum support threshold, minsup, then I is not frequent, that is, P (l) < minsup. If an item A is added to the itemset I, then the resulting itemset (i.e., I U A) cannot occur more frequently than I. Therefore, 1 U A is not frequent either, that is, P (I U A) < minsup.

This property belongs to a special category of properties called anti-monotone in the sense that if a set cannot pass a test, all of its supersets will fail the same test as well. It is called anti-monotone because the property is monotonic in the context of failing a test. “How is the Apriori property used in the algorithm?” To understand this, let us look at how Lk-1is used to find Lk. A two-step process is followed, consisting of join and prune actions.

THE JOIN STEP: To find L_k, a set of candidate k-itemsets is generated by joining L_{k-1} with itself. This set of candidates is denoted C_k. Let l1 and l2 be itemsets in L_{k-1}. The notation $l_i[j]$ refers to the jth item in l_i (e.g., l_1 [k — 2] refers to the second to the last item in l_1. By convention, Apriori assumes that items within a transaction or itemset are sorted in lexicographic order. The join, $L_{k-1} \infty L_{k-1}$, is performed, where members of L_{k-1} are joinable if their first (k — 2) items are in common. That is, members l1 and 12 of L_{k-1} are joined If $(l_1[1] = l_2[1]) \wedge (l_1[2]) = l_2[2]) \wedge \ldots \wedge (l_1[k — 2] = l_2[k — 2]) \wedge(l_1[k — 1] < l_2[k — 1))$. The condition $l_1[k — 1] < 1_2[k — 1]$ simply ensures that no duplicates are generated. The resulting itemset formed by joining l_1 and l_2 is $l_1[1]l_1[2] \ldots 1_1[k — 1]l_2[k— 1]$.

THE PRUNE STEP: Ck is a superset of Lk, that is, its members may or may not be frequent, but all of the frequent k-itemsets are included in Ck. A scan of the database to determine the count of each candidate in Ck would result in the determination of Lk (i.e., all candidates having a count no less than the minimum support count are frequent by definition, and therefore belong to Lk). Ck, however, can be huge, and so this could involve heavy computation. To reduce the size of Ck, the Apriori property is used as follows. 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 Lk-1, then the candidate cannot be frequent either and so can be removed from Ck. This subset testing can be done quickly by maintaining a hash tree of all frequent item sets.

3. DIRECT HASHING AND PRUNING ALGORITHM

In the DHP algorithm, a large hash table can be defined such that each different itemsets is mapped to different locations in the hash table, then the entries of the hash table gives the actual count of each itemset in the database. In that case, it doesn't have any false positives and as a result of this, an extra processing for counting the occurrences of each itemset is eliminated. It has also been showed that, the amount of data that has to be scanned during the large itemset discovery is another performance-related issue. Reducing the number of transactions to be scanned and trimming the number of items in each transaction improves the data mining efficiency in later stages.The algorithm forms all k-subsets of items in each transaction and inserts the ones whose all k-1 subsets are large to the hash table. For that reason the algorithm does not miss any frequent itemset. Since the algorithm makes a pruning during the insertion of the candidate kitemsets to the Hk, the size of the hash table is not large and fits into memory.

EFFICIENT GENERATION OF LARGE ITEMSET:

By utilizing a hash tree, DHP is very effective for generation of large itemsets, in particular for large 2-itemsets, where the number of candidate itemsets is, in orders of magnitude, lesser than that by previous methods, greatly improving the performance bottleneck (Gauhar wadhera 2002).

As a preliminary, the approach adopted by earlier works, notably Apriori for discovering large itemsets from, a transaction database. In Apriori, in each iteration, the candidate set for large itemsets is constructed, and large itemsets are determined based on a pre determined support. In the first iteration, Apriori scans the all transactions to count the number of occurrences of each item (attribute). This is the candidate 1-itemset, denoted by C1. The large 1-itemset L1 is generated from C1 by checking for 1-itemsets with support greater than minsup. To generate large 2-itemsets, Apriori uses L1* L1 to obtain candidate 2-itemset C2, where '*' is the concatenation operation and may be performed as detailed in the Apriori Algorithm (section 3.1). From C2, 2-itemsets having support greater than minsup are stored as L2. This process is repeated for all possible k-itemsets (Jong Soo Park 1997).

DHP uses the technique of hashing to filter out unnecessary itemsets for next candidate generation. When the support of candidate k-itemsets is counted by scanning the database, DHP accumulates information about (k+1)-itemsets in such a way that all possible (k+1)itemsets are hashed to a hash table. Each bucket in the hash table consists of a number to denote how many itemsets have been hashed to that bucket so far. Based on this hash table a bit vector is constructed, where the bit vector is one if the number in the corresponding bucket is greater than or equal lo minsup. In the candidate generation stage, after computing Ck = Lk-l * Lk-1, each k-itemset is checked if it is hashed to a bucket whose bit vector is one. Such use of a hash table considerably decreases number of the candidate k-itemsets, thus serving the purpose of reducing costs for computation of large itemsets at each iteration.

EFFECTIVE DATABASE PRUNING

DHP prunes the database on each attribute. All attributes in Ck which do not occur in at least k of candidate k- itemsets. Since each (k-1) itemset of a large k-itemset must itself be large, this method discards those itemsets that cannot be large. The generation of a smaller number of candidate sets by DHP enables us to effectively trim the database at much earlier iterations, thereby reducing the computational costs for later iterations. The above concept as applied by DHP is used in the function count_support(). This is only a necessary condition, not a sufficient one. In function make_hasht(), Then further check if each item in a transaction is indeed covered by a (k+1)itemset with all of its (k+1) k-itemsets contained in Ck.

4. EXPERIMENTAL RESULTS

The algorithm is implemented in Java. The algorithm is run on the sales record data obtained from the Musical Store. The dataset consists of the transactions that are recorded for a month, and it consists of around 10,000 transactions and around 800 different items. Experimentation is done over the same dataset to compare DHP algorithm with Apriori.

Processing for counting the occurrences of each itemset. Experimental results are shown in the following table.

TABLE 1: Experimental Result of Apriori Apriori and DHP From 10000 Records

	L	L1	C1	L2	C2	Total Time

Apriori	1000	1097	598	298	149	7.50
Dhp	1000	1097	598	74	51	4.22
Apriori	2000	2236	1115	582	226	18.28
Dhp	2000	2236	1115	155	104	10.32
Apriori	3000	3196	1547	673	225	42.19
Dhp	3000	3196	1547	174	147	17.50
Apriori	4000	4123	2205	1088	393	52.97
Dhp	4000	4123	2205	185	159	24.37
Apriori	5000	5245	2982	1699	1023	53.59
Dhp	5000	5245	2982	219	102	30.31

TABLE 2: EXPERIMENTAL RESULT OF AND DHP FROM 5000 RECORDS

	L	L1	C1	L2	C2	Total Time
Apriori	6000	6126	3542	1937	1167	69.37
Dhp	6000	6126	3542	246	149	41.09
Apriori	7000	7246	4105	1995	953	83.28
Dhp	7000	7246	4105	240	185	49.69
Apriori	8000	8041	1286	2143	886	86.56
Dhp	8000	8041	1286	266	200	56.25
Apriori	9000	9131	5031	2243	985	110.78
Dhp	9000	9131	5031	269	214	74.68
Apriori	10000	10331	5577	2967	1020	140.78
Dhp	10000	10331	5577	317	277	102.34

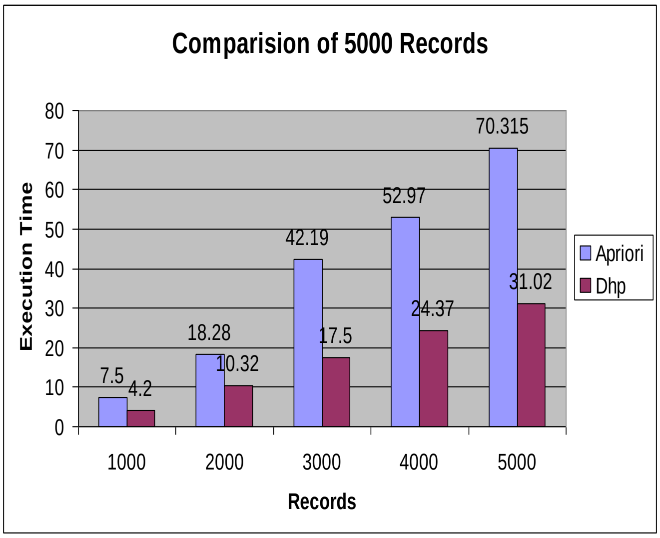

Figure 1: Graph for Comparison of 5000 Records.

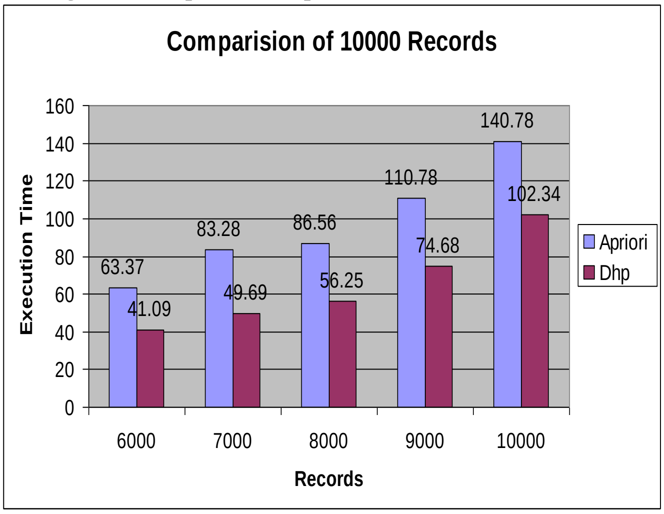

Figure 2: Graph for Comparison of 10000 Records

5. APRIORI ALGORITHM

Any subset of a frequent itemset must be frequent

if {Westlife, Dangerous, Eminem} is frequent, so is {Westlife, Dangerous}

Every transaction having { Westlife, Dangerous, Eminem } also contains { Westlife, Dangerous}}

Apriori pruning principle: If there is any itemset which is infrequent, its superset should not be generated/tested!

Method:

generate length (k+1) candidate itemsets from length k frequent itemsets, and test the candidates against DB

6. CONCLUSION

In this paper, the implementation of Apriori and DHP algorithms are carried out for musical database. In order to test the performance of the algorithm, a comparison of both the algorithms were made over the real dataset that was obtained from Musical store. As the experimentation has showed, DHP algorithm performs better than the Apriori algorithm since at each step it reduces the database size to be scanned, and it generates much smaller sized C2 at the initial step. A larger dataset would yield more meaningful results. As future work, the DHP algorithm may be run over larger sets of data, and experimentation on memory requirement of the algorithm may be performed. A co-occurrence technique automatically extracts musical similarity between titles and between artists. The technique yields a distance matrix for arbitrary sets of items. The preliminary results on small databases show that the technique is able to extract similarities between items. Besides scaling up these experiments to larger databases, different sources of similarity can be integrated and can be used in EMD systems.

7. REFERENCES

[1] D. Pyle, Data Preparation for Data Mining. San Francisco, CA Morgan Kaufmann, 1999.

[2] Munz, Matt. Data Mining in Musicology. Yale University. 2005.

[3] R. Agrawal, T. Imielinski, and A. Swami, "Database Mining: A Performance Perspective," IEEE Trans. Knowledge and Data Eng.,vol. 5, no. 6, Dec. 1993.

[4] Pachet, Francois, Gert Westermann, and Damien Laigre. "Musical Data Mining for Electronic Music Distribution". 1st International Conference on Web Delivering of Music. 2001.

[5] Pavankumar Bondugula, Implementation and Analysis of Apriori Algorithm for Data Mining