Summary Structures for Frequency Queries on Large Transaction Sets *

Dow-Yung Yang, Akshay Johar, Ananth Grama, and Wojciech Szpankowski

Computer Science Department,
Purdue University,
West Lafayette, IN 47907.
{yangdy, johar, ayg, spa}@cs.purdue.edu
Phone: (765) 497 7985

Abstract

As large-scale databases become commonplace, there has been significant interest in mining them for commercial purposes. One of the basic tasks that underlies many of these mining operations is querying of transaction sets for frequencies of specified attribute values. The size of these databases makes it important to develop summary structures capable of high compression ratios as well as supporting fast frequency queries. The nature of the problem and its differences with respect to traditional text compression allows very high compression ratios.

In this paper, we propose a binary trie-based summary structure for representing transaction sets. We demonstrate that this trie structure, when augmented with an appropriate set of horizontal pointers, can support frequency queries several orders of magnitude faster than raw transaction data. We improve the memory characteristics of our scheme by compressing the trie into a Patricia trie and demonstrate that this does not have a significant adverse effect on frequency query time. We further reduce the size of this trie by selectively pruning branches to compute a "dominant" trie that is capable of approximate frequency querying. The complement trie called the "deviant" trie is also useful in many data mining applications. Recompressing the "dominant" trie into a Patricia trie results in further compression of the trie. Finally, we demonstrate that our binary compressed trie structure has better memory (compression) characteristics compared to related schemes. We support our claims with experimental results on datasets from the IBM synthetic association data generator.

*This work is supported in part by the National Science Foundation grants EIA-9806741, ACI-9875899, and ACI-9872101. Computing equipment used for this work was supported by National Science Foundation MRI grant EIA-9871053 and by the Intel Corp.

1 Introduction and Motivation

With the availability of large online databases, there has been significant interest in mining this data for underlying patterns of interest [6, 4]. Despite the strong information theoretic underpinings of such tasks, relatively little work has been done on adapting information theoretic results and techniques to data mining and analysis with some noted exceptions. In this paper, we address the problem of summarizing transactions with a view to supporting fast queries over these transactions. Before we describe the target problems in greater detail, some terminology and definitions need to be formalized.

Definitions and Terminology

The given database D, also known as a transaction-set, consists of variable length transactions t_j for $j = 1, ..., n$. Each transaction in the database consists of a set of items. Each item is drawn from a universe of items represented by the set $I = \{i_1, i_2, ..., i_s\}$. A set of items is referred to as an "itemset". An itemset with k items in it is called a k-itemset. The length of an itemset is the number of items in the itemset. It is easy to see that transactions can also be viewed as fixed-length binary attribute vectors. Each attribute corresponds to an item and the absence or presence of the item is signified by a 0 and 1 respectively. The frequency of an itemset S ($\subset I$) is the number of transactions in which the itemset appears. An itemset is said to have a support s if it appears in more than a fraction s of the transactions. For a given value of s, itemsets satisfying the support criteria are sometimes also called "frequent" itemsets.

One of the common data mining tasks is the derivation of Association rules. An association rule is a rule of the form $S_1 \Rightarrow S_2$ (where $S_1, S_2 \subset I$ and $S_1 \cap S_2 = \emptyset$). The generalized association framework allows exclusions of items in addition to inclusion. That is, if I' were the set of complements (absence) of items, then $S_1, S_2 \subset I \cup I'$. An association rule has, associated with it a notion of strength or confidence. The confidence of a rule of the form $S_1 \Rightarrow S_2$ is given by the ratio of support of $S_1 \cup S_2$ to the support of S_1.

The conventional association rule framework [2] specifies values "minsup" and "minconf" for minimum support and confidence respectively. The objective is to find, from a given transaction-set D, all association rules of the form $S_1 \Rightarrow S_2$ such that the support of $S_1 \cup S_2$ is greater than minsup and the confidence of the rule $S_1 \Rightarrow S_2$ is greater than minconf.

Problems of this nature arise in diverse application domains ranging from document retrieval to portfolio analysis and market-basket analysis. In all of these examples, the objective is to find frequently co-occurring itemsets and to use their frequencies to compute conditional probabilities. Traditional formulations of this problem use the downward closure property of frequent sets, namely, subsets of frequent sets must also be frequent. Fast algorithms for computing frequent sets start by computing 1-frequent sets. Candidate 2-frequent sets are computed by combining 1-frequent sets. This candidate set is pruned to true 2-frequent sets by counting. This process continues until all frequent sets have been computed. The major cost of this operation is the I/O cost associated with repeated passes over data. Algorithms have been developed that reduce the number of passes in this computation to two.

The conventional framework has several drawbacks. Often, it is necessary to repeatedly run the association algorithm with different values of minsup and minconf to arrive at a desirable set of association rules. Although frequent set computation with a single specified support value only makes two passes over the data, this may have to be done several times over. In many cases, we may need to compute associations on exclusions as well, for example rules of the form `MSFT` ⇒ `not(RHAT)` in portfolio analysis. This is not supported by the conventional framework. Finally, it is useful to detect strong associations (rules with high confidence) irrespective of their support. For example, stocks `COBT` and `AKAM` may only appear in a small number of stock portfolios but their presence implies a strong characteristic, namely, an aggressive IPO seeking investor.

With a view to alleviating these drawbacks, we propose a framework for summarizing transaction sets into a binary trie [8]. We demonstrate that this trie structure, when augmented with an appropriate set of horizontal pointers, can support frequency queries several orders of magnitude faster than raw transaction data. We improve the memory characteristics of our scheme by compressing the trie into a Patricia trie [7] and demonstrate that this does not have a significant adverse effect on frequency query time. We further reduce the size of this trie by selectively pruning branches to compute a "dominant" trie that is capable of approximate frequency querying. Recompressing the "dominant" trie into a Patricia trie results in further compression of the trie. We support our claims with experimental results on datasets from the IBM synthetic association data generator [1].

The rest of the paper is organized as follows: Section 2 summarizes related results in literature; Section 3 outlines the family of schemes proposed in this paper; Section 4 presents experimental results in support of our claims; and Section 5 draws conclusions and outlines current research directions.

2 Related Research

An important difference between compression of transactions as opposed to conventional text is the fact that the order of items in the itemset is not important. Rather, the absence or presence of an item is what needs to be represented. Furthermore, it is not necessary to reconstruct the database. Satisfying frequency queries is the target of the summary structure. There have been some efforts at developing such summary structures for transaction sets. Of these, the results of Amir et al. [3] and Han et al. [5] are most closely related.

Amir et al. use a trie for encoding the database with a view to computing frequent sets. Each transaction in the database is sorted on a predefined order of items and inserted into the trie. Nodes of the trie are used to store the count of number of instances of the prefix leading up to the node. For example, if the database has the transactions $\{\{1, 2\}, \{1, 3, 4, 5\}, \{2, 3, 4\}, \{2, 3, 4, 5\}, \{2, 3, 4\}\}$, the corresponding trie structure is illustrated in Figure 1. In its simple form, it is easy to see that the memory requirements of such a trie structure can be overwhelming (exponential in the depth). However, under the assumption of bounded length frequent sets, the depth of this trie can be bounded. Consequently, for many realistic problems, it is possible to generate such trie structures and compute frequent sets. The approach has several advantages: once a trie structure with maximum depth has been computed, frequent sets with varying support can be easily determined provided their length does not exceed the depth of the tree. The framework can also be used to compute frequency of negations. For example, the frequency of of itemset $\{i_1, i_2, i'_3\}$ can be computed as the difference of frequencies of $\{i_1, i_2\}$ and $\{i_1, i_2, i_3\}$.

While the scheme is elegant and useful, its one major drawback is the memory requirement. Since the memory requirement is exponential in depth with a large branching constant, this method is applicable only to shallow trees. In this paper, we present a scheme using binary tries that can handle extremely large transactions and transaction sets.

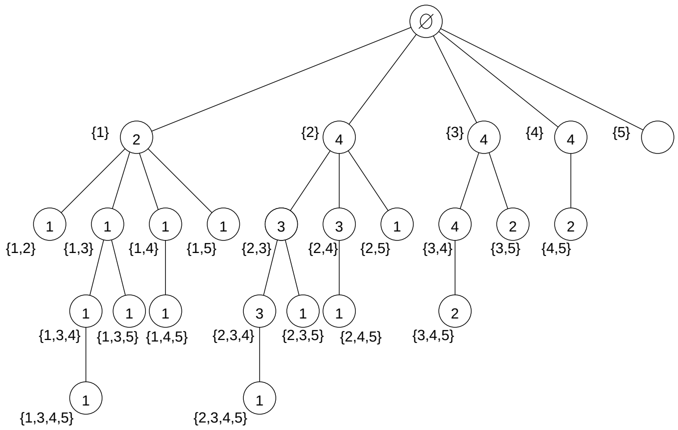

Figure 1: Using a trie for computing frequent sets.

Han et al. [5] reduce the memory requirement of the above scheme by modifying the trie structure of Amir et al. In the scheme of Amir et al., a transaction of the form {1, 3, 4, 5} must be inserted into all paths corresponding to subsets of the transaction: {1, 3, 4, 5}, {3, 4, 5}, {4, 5}, and {5}. This multiplicity is eliminated in the trie structure of Han et al. Given the same set of transactions: {{1, 2}, {1, 3, 4, 5}, {2, 3, 4}, {2, 3, 4, 5}, {2, 3, 4}}, Han et al. first construct a frequency sorted ordering of the items, i.e., 2, 3, 4, 1, 5. In the second pass, the trie structure illustrated in Figure 2 is constructed. This structure is augmented with optimizations such as horizontal pointers for counting and restricting the tree to only those items that are frequent.

While this trie alleviates the memory related drawback of the approach of Amir et al., it also loses some of the desirable features. Frequent set computation is more complex since they are not explicitly available in the tree. It is also more difficult to estimate frequency of exclusions (absence of items) in this framework. If the number of frequent 1-items is large, the memory requirement of this alternate trie structure itself may be large.

In this paper, we propose the use of a binary trie with optimizations to reduce the memory requirements drastically using trie compression. By selectively truncating branches of the trie, we show that the size of the trie can be further reduced at the cost of limited loss in accuracy. Finally, we show that in addition to frequent

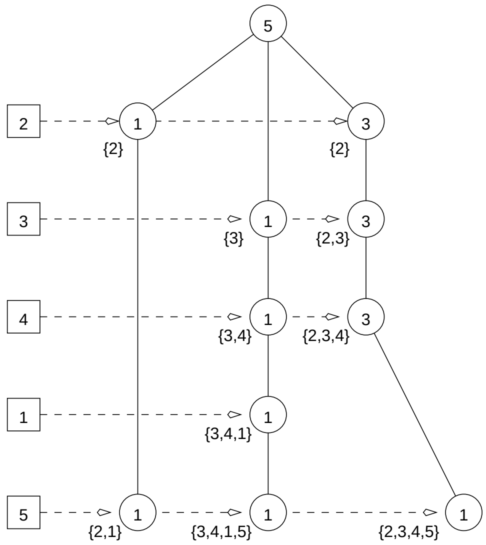

Figure 2: Alternate trie structure for computing frequent sets.

set computations, our framework can be used for general frequency computations. Experimental results are presented using the IBM synthetic association data generator to support all these claims.

3 Summary Structures for Representing Transaction Sets

The basis of our schemes is a binary trie representation of the transactions. Each transaction is viewed as an n-length string of 0s and 1s. Note that n is the total number of items in the set I. Transactions are then inserted into a binary trie. Consider once again, the example of the transaction set {{1, 2}, {1, 3, 4, 5}, {2, 3, 4}, {2, 3, 4, 5}, {2, 3, 4}}. Assuming the universe of items to be set of items from 1 to 5, these transactions can be converted into the following binary attributed strings: 11000, 10111, 01110, and 01111. These are inserted into a trie as illustrated in Figure 3(a). Each internal node in the trie maintains a count of the number of transactions that share the prefix leading to the node in the trie. In addition, it maintains a pointer to a string corresponding to some leaf node reachable from current node, and the count of the prefix size terminating at current node. Note that this is an implicit representation of the prefix of the node in the trie.

Counting in the Augmented Trie Structure

Once such a binary trie structure has been constructed, the task of computing frequencies of a given subset corresponds to traversals in the trie. Consider the simple case in which the frequency of itemset {1,2} is queried. The search traverses top-down to the first item, in this case 1. The search prunes all non-one branches for the item and performs a sum operation on set {2} for all subtrees rooted at nodes corresponding to existence of item 1. In this case, the search terminates quickly returning the

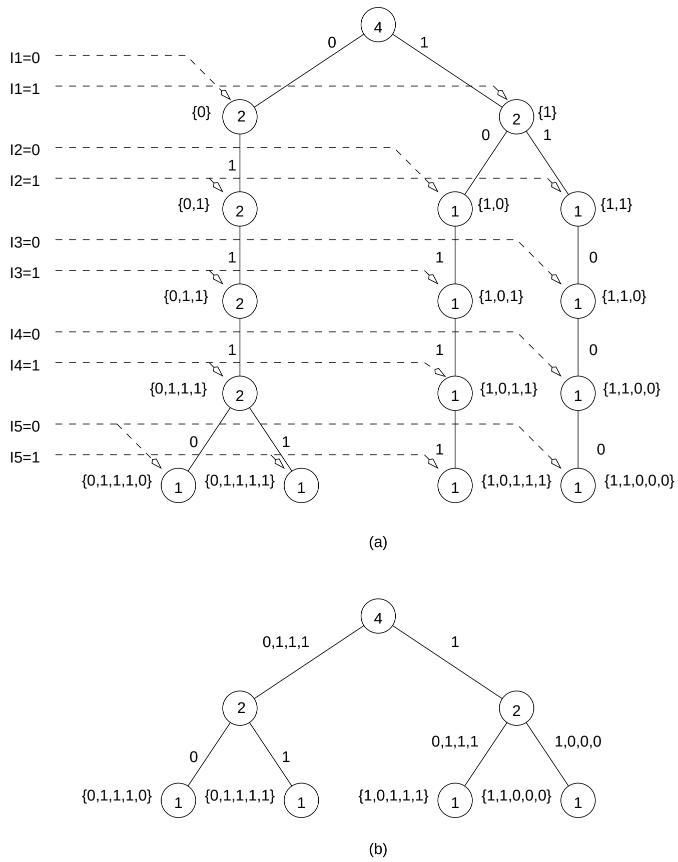

Figure 3: Binary trie structure for storing inclusions and exclusions represented in transactions along with its compressed (Patricia) trie counterpart.

number 1.

In general however, there may be a large number of recursive calls to sum. Specifically, the number of recursive calls to sum for a query on set $\{i_j, ... i_r\}$ which has been ordered according to the trie construction order grows exponentially in $r - j$. This is not desirable. To alleviate this, we augment the trie structure with a set of horizontal pointers that directly index into nodes specifying the absence or presence of itemsets. Using this set of horizontal pointers, the frequency query can be reduced to simple template matching over a list of trie nodes. For example, if the count of items 1 and 5 are desired, the horizontal pointers corresponding to item 5 are traversed. Since an implicit prefix is available at each node, a template corresponding to presence of items 1 and 5 is applied. Wherever this template returns true, the node count is included in the frequency.

An important and desirable feature of this framework is the fact that frequencies can be computed for exclusion of items as well. If we wanted to compute the count of all transactions that had item 1 but not 5, the corresponding pointer chain can be traversed applying the required template and maintaining count.

Compressing Trie for Improved Performance

One of the undesirable features of the above framework is the fact that there may be significant numbers of single child nodes in the trie. This is a direct consequence of the fact that transactions are typically sparse in nature, i.e., a transaction is expected to contain only a very small fraction of the total number of items in the universe I. This implies that the storage and computation characteristics of the above structure can be significantly improved by compressing the trie.

In Figure 3(b), we illustrate the compression of the binary trie structure into a Patricia trie. Each node in the trie with degree one is merged with its child. This process is repeated until all nodes in the trie are of degree two. The simple illustration in Figure 3(b) shows the power of this simple compression in the context of compressing sparse transaction sets. In general, we notice compression ratios of up to two orders of magnitude using this compression step. However, compression in this manner complicates the simple horizontal pointer structure we used for frequency queries. As nodes in the trie are collapsed during compression, each horizontal pointer is changed to point to the top node in the set of nodes that are collapsed. As before, each node in the Patricia trie maintains an implicit representation of the prefix leading to the node. Counts can be obtained as before by applying the desired template to each link in the set of horizontal pointers. We have omited the horizontal pointers in Figure 3(b) for sake of clarity.

Compressing Patricia Tries for Approximate Frequencies

In many applications of frequency querying, it is enough to compute an approximate frequency of specified itemset. This is built into our Particia tree framework by a process of pruning insignificant branches in the tree. Since a node in the trie represents a count (sum) of all its descendants, we can prune nodes with low relative weight. Specifically, if the weights of a node and its two children were represented by w_i, w_j, and w_k respectively , then the ratio w_k/w_i is the fraction of the nodes headed along the specified child. If this ratio is less than some prespecified constant ϵ, then the branch is pruned off. We call the tree resulting from such pruning as the "dominant" trie. The complement of the "dominant" tree is also called a "deviant" trie and can be useful in many data mining applications as well. The dominant trie thus computed is recompressed into an updated Patricia trie. We observe that depending on the strength of patterns in data and the degree of tolerated error, this can lead to significant compression of the trie structure.

4 Experimental Results

In this section, we report on the implementation of our binary trie framework. Specifically, we aim to demonstrate the following key features of our framework:

- The proposed augmented trie structure is capable of extremely fast frequency queries.

- Trie compression into a Particia trie results in significant compression with little overhead in terms of time. Specifically, this compression negates the overhead of explicitly storing item exclusions.
- Pruning and recompression of tries results in further compression while resulting in very small overall errors.
- The preprocessing time to construct these trie structures is not significant compared to the number of frequency queries it gets amortized over.

To demonstrate these results we use the IBM synthetic association data generator [1]. We generate a range of test cases with varying number of transactions, items, pattern lengths, and pattern strengths. The number of transactions is varied from 100K to 10M and the number of items is varied from 100 to 1000.

Query Time for Frequencies. We compute the query time for the augmented trie, truncated trie, and Patricia trie. These times are then compared to querying the raw data file. Since query performance is a function of the itemset, the time reported here is an average of 10 queries of randomly selected itemsets. These query times are reported in Figure 4. Several observations can be made from this figure. The query time on our trie structures is several orders of magnitude faster than querying raw files (Figure 4 is drawn on a log scale). Further reduction in time can be achieved by truncating the trie. With query times in milliseconds even on large datasets, it is possible to compute associations extremely quickly in our framework.

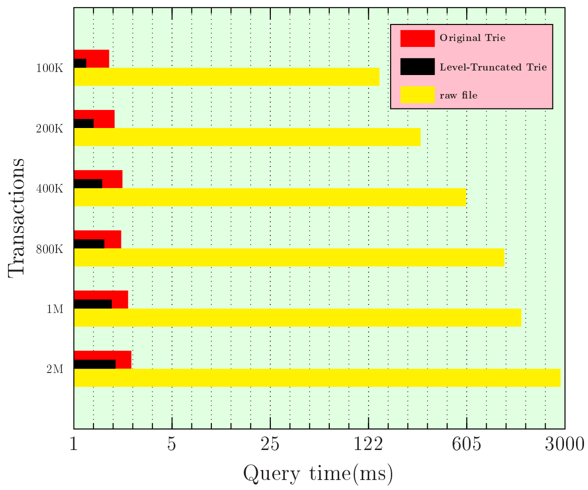

Figure 4: Query time for raw file and augmented binary trie structure. Note that the time axis is logarithmic.

Preprocessing Time. An important aspect of constructing summary structures is their space and time requirement. The trie structure of Amir et al. has excellent time characteristics for querying but its applicability is limited by the memory requirement of the scheme. The time taken to compute our augmented trie structure is shown in Table 1. This time is relatively small and is generally amortized over a large number of frequency queries. If the binary trie is restricted to only those items that are frequent (level truncated trie), this time can be further reduced.

Size of Transaction Set	Trie	Level-truncated Trie
100K	1.54×10^{1}	6.90×10^{-1}
200K	2.94×10^{1}	1.30×10^{0}
400K	5.64×10^{1}	2.34×10^{0}
800K	1.09×10^{2}	4.41×10^{0}
1000K	1.36×10^{2}	5.93×10^{0}
2000K	2.63×10^{2}	1.17×10^{1}

Table 1: CPU time for building the trie and the level-truncated trie of transaction sets

Compression Ratios from Patricia Tries. In Table 2, we present the effect of compressing tries into Patricia tries and its effect on query time. As expected, compression of tries leads to very significant reduction in the overall space requirement of tries. We observe compression ratios in the range of 30 - 35. This is because of the sparse nature of the transactions, i.e., each transaction contains only a fraction of the total universe of items. In addition, we notice that the corresponding frequency computation time also goes down. This is a result of the reduction in number of nodes in the trie, and consequently in the number of traversals.

No. Trans.	Number of Nodes		Query Time	
	Original Trie	Compressed Trie	Original Trie	Compressed Trie
196733	6286290	172295	1.626015e-03	9.241104e-04
393424	10774497	303951	1.809001e-03	1.099944e-03
786872	18337195	532295	1.860023e-03	1.129985e-03
983555	21729339	636397	1.955032e-03	1.152039e-03

Table 2: Result of trie compression on the number of nodes and the frequency computation time.

Size Comparison of Binary Patricia Trie to k-Trie. In Table 3 we present a comparison of sizes of our binary trie with respect to the k-ary trie generated by inserting sorted strings of items (Figure 2) as opposed to binary vectors. With various compressions applied, we can see that our binary trie is up to a factor of two smaller than the k-trie. When higher error tolerances are allowed for frequency counts, this ratio can be improved further. Furthermore, since exclusions are explicitly represented, the binary trie structure can be used to quickly compute frequencies of inclusion as well as exclusions of items.

No. Trans.	Binary Trie	Compressed Trie	k-Trie
98430	3635911	96607	184827
196733	6286290	168787	314803
393424	10774497	303867	532762
786872	18337195	532079	888604
983555	21729339	636129	1048994

Table 3: Comparison of sizes of compressed binary trie with respect to a k-trie.

5 Concluding Remarks

In this paper, we have presented a summary structure for transaction sets that can be used for fast frequency queries. We have shown that this structure is capable of significant improvement in performance over raw data in terms of time and storage. This work alleviates many of the drawbacks of existing work in literature, namely, queries on exclusions and scalability to large itemsets. As a part of our ongoing research, we are building a complete association framework on top of our query mechanism. We are also investigating the use of algebraic transformations to detect co-occurrences for further reducing the trie size.

References

[1] R. Agrawal, A. Arning, T. Bollinger, M. Mehta, J. Shafer, and R. Srikant. The quest data mining system. In *Proceedings of the Second International Conference on Knowledge Discovery in Databases and Data Mining*, Portland, Oregon, August 1996.

[2] Rakesh Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In *20th International Conference on Very Large Data Bases (VLDB)*, Santiago, Chile, June 1994.

[3] Amihood Amir, Ronen Feldman, and Reuven Kashi. A new and versatile method for association generation. *Information Systems*, 22(6/7):333–347, 1999.

[4] U.M. Fayyad et al. *Advances in Knowledge Discovery and Data Mining*. AAAI/MIT Press, Menlo Park, CA, 1996.

[5] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. Technical Report TR-99-12, Computing Science Technical Report, Simon Fraser University, October 1999.

[6] Naren Ramakrishnan and Ananth Grama. Data mining: From serendipity to science. *IEEE Computer Special Issue on Data Mining*, August 1999.

[7] W. Szpankowski. Patricia tries again revisited. *Journal of the ACM*, 37:691–711, 1990.

[8] W. Szpankowski. A generalized suffix tree and its (un)expected asymptotic behaviors. *SIAM J. Computing*, 22:1176–1198, 1993.