Efficient Mining of Max Frequent Patterns in a Generalized Environment

Daniel Kunkle Donghui Zhang Gene Cooperman
College of Computer and Information Science
Northeastern University
Boston, MA 02215
{kunkle, donghui, gene}@ccs.neu.edu

ABSTRACT

This poster paper summarizes our solution for mining max frequent generalized itemsets (g-itemsets), a compact representation for frequent patterns in the generalized environment.

Categories and Subject Descriptors: H.m Information Systems: Miscellaneous

General Terms: Algorithms.

Keywords: data mining, max frequent itemsets.

1. INTRODUCTION

Mining generalized frequent patterns is a well-motivated existing problem [2, 3]. Here, generalized itemsets (or patterns) employ a taxonomy of items, rather than a flat list of items. This produces more natural frequent itemsets such as (meat, milk) instead of (beef, milk), (chicken, milk), etc.

We address the problem of mining **max** generalized frequent itemsets: those without frequent supersets. This is an extremely compact representation of all generalized frequent itemsets. This compact representation solves a standard dilemma in mining patterns: with a small threshold for frequency, the user is overwhelmed by the hordes of identified patterns; but with a large threshold for frequency, some interesting patterns fail to be identified.

2. PROBLEM DEFINITION

The set of all items form a taxonomy $\mathcal{T}$, which is a tree structure. An example is shown in Figure 1(a). Think of leaf items A as *apple*, B as *banana*, and C as *candy*. And think of non-leaf items W as $fruit$ and Y as $food$. A transactional database $\mathcal{D}$ is a list of transactions, each of which containing some items from the leaf level of $\mathcal{T}$.

DEFINITION 1. *Given a taxonomy $\mathcal{T}$, a* generalized itemset*, or* **g-itemset** *in short, is a non-empty set of items from $\mathcal{T}$, where no two of the items have an ancestor-descendant relationship in $\mathcal{T}$.*

Hence, the set {apple, fruit} is not considered a g-itemset since it is not compact. (The equivalent compact itemset is {apple}.)

CIKM'06, November 5–11, 2006, Arlington, Virginia, USA.
ACM 1-59593-433-2/06/0011.

Given an item $i \in \mathcal{T}$ and a g-itemset S, we say i belongs to S with respect to $\mathcal{T}$, denoted as $i \in_{\mathcal{T}} S$, if $\exists j \in S$ such that $i = j$ or i is an ancestor of j in $\mathcal{T}$. Intuitively, any transaction that contains *apple* is considered to contain $fruit$. Therefore $fruit \in_{\mathcal{T}} \{apple\}$.

Given two g-itemsets S_1 and S_2, we say S_1 is a subset of S_2 with respect to $\mathcal{T}$, denoted as $S_1 \subseteq_{\mathcal{T}} S_2$, if $\forall i \in S_1$, $i \in_{\mathcal{T}} S_2$. We also have the proper subset notation ($\subset_{\mathcal{T}}$) with its obvious meaning.

The *support* of a g-itemset S is the percentage of transactions in $\mathcal{D}$ that are supersets of S with respect to $\mathcal{T}$. A g-itemset is *frequent* if its support is above a given threshold *minsupport*.

DEFINITION 2. *Given a taxonomy $\mathcal{T}$, a transactional database $\mathcal{D}$, and a threshold minsupport, a* **max frequent g-itemset** *is a frequent g-itemset without a frequent proper superset with respect to $\mathcal{T}$.*

We are interested in efficiently mining the set of all max frequent g-itemsets.

3. THE CLASSIFICATION SOLUTION

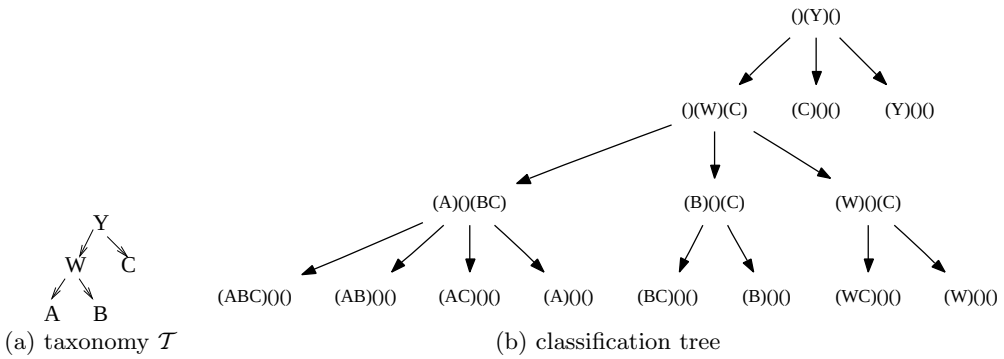
The classification-based solution has two components. Section 3.1 defines a conceptual classification tree. Section 3.2 describes the algorithm **MFGI_class** which dynamically generates the needed part of the tree, while pruning entire branches using three pruning techniques.

3.1 The Conceptual Classification Tree

This section provides a conceptual classification tree. Every g-itemset corresponds to exactly one leaf node in the tree. An index node also corresponds to a g-itemset, which is a superset of all g-itemsets in the sub-tree. An example of a classification tree is shown in Figure 1(b).

In particular, every node in our classification tree has three components, $(S_1)(S_2)(S_3)$. Any g-itemset in the subtree *must-literally-have-all-of* the g-items in S_1, *must-have-part-or-all-of* the g-items in S_2, and *may-have-part-or-all-of* the g-items in S_3. For instance, let the root of the taxonomy be Y. The root of the classification tree is $()(Y)()$. Any g-itemset in the subtree must contain some g-items in the sub-taxonomy of Y.

The children of the classification tree node will be: $()(W)(C)$, $(C)()()$ and $(Y)()()$. The first sub-tree corresponds to the g-itemsets that contain some g-item in the sub-taxonomy of W. The second sub-tree corresponds to



(a) taxonomy $\mathcal{T}$ (b) classification tree

Figure 1: A taxonomy and the corresponding classification tree.

the g-itemsets that contain some g-item in the sub-taxonomy of C but not any g-item in the sub-taxonomy of W. And $(Y)()()$ is a leaf node in the classification tree, which corresponds to a single g-itemset $\{Y\}$.

3.2 The Mining Algorithm MFGI_class

The algorithm **MFGI_class** dynamically generates the classification tree as defined in Section 3.1, with pruning techniques to prune unnecessary branches. This section focus on the pruning techniques.

Every index node in the classification tree has a *corresponding g-itemset*, which is the smallest superset of all g-itemsets in the sub-tree. For example, the corresponding g-itemset for $(W)()(C)$ is WC, and the corresponding g-itemset for $()(Y)()$ is ABC.

- **Pruning Technique 1:** If the corresponding g-itemset of a node N is frequent, prune $subtree(N)$.
- **Pruning Technique 2:** When generating the child nodes of some index node $(S_1)(X)(S_3)$, we check the frequency of $S_1 \cup \{X_i\}$ for every child g-item X_i of X in $\mathcal{T}$. If $S_1 \cup \{X_i\}$ is not frequent, prune X_i before generating the child nodes.

As an example, at node $()(Y)()$, we check the frequency of W and C. Suppose W is not frequent, we know no g-itemset that contains W or descendants of W in $\mathcal{T}$ can be frequent. So to generate the child nodes, we should imagine W does not exist, and Y has a single child C in $\mathcal{T}$. Thus only two child nodes should be generated: $(C)()()$ and $(Y)()()$.

- **Pruning Technique 3:** When generating the child nodes of some index node $(S_1)()(S_3)$, where S_3 only contains leaf g-items in $\mathcal{T}$, instead of enumerating all subsets of S_3, we should use MaxMiner [1].

4. EXPERIMENTAL ANALYSIS

There is no existing algorithm to directly compare with our new algorithm **MFGI_class**, simply because this is the first work that mines max frequent g-itemsets. We instead compare with BASIC [2]. Note that BASIC was proposed to find all frequent g-itemsets. So we give **MFGI_class** the additional handicap of producing all frequent g-itemsets from the set of identified max frequent g-itemsets.

The algorithms were implemented in Sun Java 1.5.0, and executed on a Sun Blade 1500 with 1 GB of memory running SunOS 5.9. The experimental data were generated using the widely used Quest Synthetic Generator with the following parameters. The taxonomy has 1000 g-items. The database contains 10,000 transactions with average size being 5.

We choose to compare with BASIC because it has been widely used as a baseline algorithm and it has a clear, standard implementation whose speed will not be greatly biased by the implementation. But since Srikant and Agrawal also presented Cumulate and EstMerge [2] and reported that they are 2 to 5 times faster than BASIC, in Figure 2 we include a band of a factor of 5 in the speed of BASIC (the "previous best" line was manually generated by taking 1/5 of the execution time of BASIC).

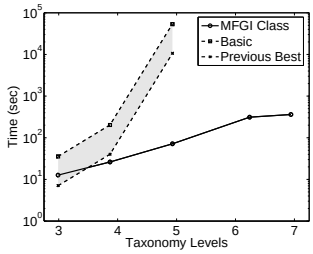


Figure 2: Comparison against BASIC.

Figure 2 demonstrates that **MFGI_class** is exponentially faster than BASIC as the number of levels of the taxonomy increases. The huge speedup over BASIC that we achieve especially for taxonomy levels of 4 and above are far beyond what is achieved by other algorithms for mining frequent g-itemsets.

5. REFERENCES

[1] R. J. Bayardo Jr. Efficiently Mining Long Patterns from Databases. In *SIGMOD*, pages 85–93, 1998.

[2] R. Srikant and R. Agrawal. Mining Generalized Association Rules. In *VLDB*, pages 407–419, 1995.

[3] K. Sriphaew and T. Theeramunkong. Fast Algorithms for Mining Generalized Frequent Patterns of Generalized Association Rules. *IEICE Transactions on Information and Systems*, E87-D(3), March 2004.