AIM2: Improved implementation of AIM

Sagi Shporer
School of Computer Science
Tel-Aviv University
Tel Aviv, Israel
shporer@tau.ac.il

Abstract

We present AIM2-$\mathcal{F}$, an improved implementation of AIM-$\mathcal{F}$ [4] algorithm for mining frequent itemsets. Past studies have proposed various algorithms and techniques for improving the efficiency of the mining task. We have presented AIM-$\mathcal{F}$ at FIMI'03, a combination of some techniques into an algorithm which utilize those techniques dynamically according to the input dataset. The algorithm main features include depth first search with vertical compressed database, diffset, parent equivalence pruning, dynamic reordering and projection. Experimental testing suggests that AIM2-$\mathcal{F}$ outperforms existing algorithm implementations on various datasets.

1. Introduction

Finding association rules is one of the driving applications in data mining, and much research has been done in this field [7, 3, 5]. Using the support-confidence framework, proposed in the seminal paper of [1], the problem is split into two parts — (a) finding frequent itemsets, and (b) generating association rules.

Let I be a set of items. A subset $X \subseteq I$ is called an itemset. Let D be a transactional database, where each transaction $T \in D$ is a subset of $I : T \subseteq I$. For an itemset X, support(X) is defined to be the number of transactions T for which $X \subseteq T$. For a given parameter minsupport, an itemset X is call a *frequent itemset* if support$(X) \geq$ minsupport. The set of all frequent itemsets is denoted by $\mathcal{F}$.

We have presented AIM-$\mathcal{F}$ [4] for mining frequent itemsets. The AIM-$\mathcal{F}$ algorithm build upon several ideas appearing in previous work, a partial list of which is the following: Apriori [2], Lexicographic Trees and Depth First Search Traversal [6], Dynamic Reordering [5], Vertical Bit Vectors [7, 3], Projection [3], Difference sets [9], Dynamic Reordering [5], Parent Equivalence Pruning [3, 8] and Bit-vector projection [3].

High level pseudo code for the AIM-$\mathcal{F}$ algorithm appears in Figure 1.

```
AIM-F(n : node, minsupport : integer)
(1)  t = n.tail
(2)  for each α in t
(3)      Compute s_α = support(n.head ⋃ α)
(4)      if (s_α = support(n.head))
(5)          add α to the list of items removed by PEP
(6)          remove α from t
(7)      else if (s_α < minsupport)
(8)          remove α from t
(9)  Sort items in t by s_α in ascending order.
(10) While t ≠ ∅
(11)     Let α be the first item in t
(12)     remove α from t
(13)     n'.head = n.head ⋃ α
(14)     n'.tail = t
(15)     Report n'.head ⋃{All subsets of items
             removed by PEP} as frequent itemsets
(16)     AIM-F(n')
```

Figure 1. AIM-$\mathcal{F}$

2. Implementation Improvements

We now describe the difference between AIM-$\mathcal{F}$ and AIM2-$\mathcal{F}$ implementations:

- Integer to String conversions - Experiments run time analysis have shown that the conversion of integers to strings is a major CPU consumer. To reduce conversion time two steps are taken:
  - Item name conversion - When printing an itemset all the item names in the itemset are

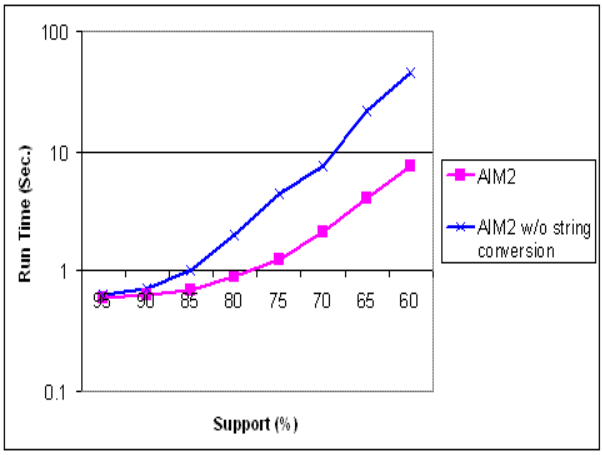

Figure 2. Connect dataset: Testing AIM2-$\mathcal{F}$ with and without the string conversion improvements

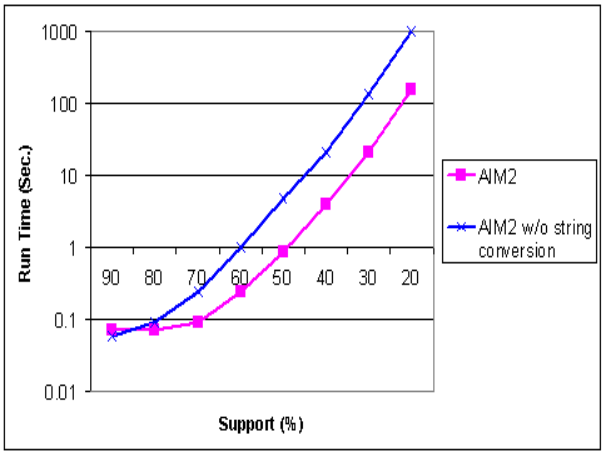

Figure 3. Chess dataset: Testing AIM2-$\mathcal{F}$ with and without the string conversion improvements

printed. In this mining task the items are numbers, and need to be converted to strings. Instead of creating the string every time before printing, the conversion is done once for every item, when the item is loaded during the dataset reading process.

– Support conversion - To print the support it must be converted to a string. To enable fast conversion of the support value to string, a static lookup table from integer to string was added. The lookup table contains the 64K integer values above the *minSupport*. Every entry in the lookup table has the string representation of the entry attached. Every time a support value needs to be converted to string, it is first checked if the value appears in the lookup table, if so, the string is taken from the table, with a very low cost.

In figures 2 and 3 we compare the AIM2-$\mathcal{F}$ algorithm runtime with and without the string conversion improvement. It is clear that this improvement alone contribute up to an order of magnitude improvement. As the size of the input increases (lower support) the contribution of the string conversion improvements increases.

- Late $F2$ matrix construction - The size of the $F2$ matrix is I^2 where I is the number of items. In datasets where the number of items is very large the $F2$ matrix can not be constructed. The improvement in AIM2-$\mathcal{F}$ is that the $F2$ matrix is built only for items for which $support(i) \geq minSupport$. This enables the construction of the $F2$ for larger datasets.

- Input buffer reuse - In AIM-$\mathcal{F}$ the dataset load method allocated an input buffer for every transaction read. Switching to a single input buffer that is re-used for all the transactions reduced the loading time in AIM2-$\mathcal{F}$ by nearly 50%. However the loading time is usually insignificant comparing to the overall runtime (unless the support is very high).

References

[1] R. Agrawal, T. Imielinski, and A. N. Swami. Mining association rules between sets of items in large databases. In *SIGMOD*, pages 207–216, 1993.

[2] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *VLDB*, pages 487–499, 1994.

[3] D. Burdick, M. Calimlim, and J. Gehrke. Mafia: a maximal frequent itemset algorithm for transactional databases. In *ICDE*, 2001.

[4] A. Fiat and S. Shporer. Aim: Another itemset miner. In *FIMI*, 2003.

[5] R. J. B. Jr. Efficiently mining long patterns from databases. In *SIGMOD*, pages 85–93, 1998.

[6] R. Rymon. Search through systematic set enumeration. In *KR-92*, pages 539–550, 1992.

[7] P. Shenoy, J. R. Haritsa, S. Sundarshan, G. Bhalotia, M. Bawa, and D. Shah. Turbo-charging vertical mining of large databases. In *SIGMOD*, 2000.

[8] M. J. Zaki. Scalable algorithms for association mining. *Knowledge and Data Engineering*, 12(2):372–390, 2000.

[9] M. J. Zaki and K. Gouda. Fast vertical mining using diffsets. In *KDD*, pages 326–335, 2003.