APRIORI: A Depth First Implementation

Walter Kosters, Universiteit Leiden
Wim Pijls, Erasmus Universiteit Rotterdam
The Netherlands

Presentation: Bart Goethals (thanks!)

http://www.liacs.nl/home/kosters/df/
Given a dataset of transactions, the Depth First implementation $DF$ of APRIORI (Pijls & Bioch 1999) builds a trie that contains all frequent itemsets.

For example, the itemset $\{3, 4\}$ has support 33, i.e., 33 transactions contain this itemset. Apparently, $\{4, 5\}$ is not frequent. A * denotes “not known yet”.

The right hand part of the trie has just been copied underneath bucket 2, providing the candidates for the next step. Now every transaction is in a depth first way “pushed” through this subtrie, meanwhile updating the counters.
Suppose the frequent items $i_1, i_2, \ldots, i_n$ are sorted with respect to increasing support. Then $\mathcal{DF}$ proceeds as follows:

\[
T := \text{the trie including only bucket } i_n;
\]

for $m := n - 1$ downto 1 do

\[
T' := T;
\]

$T := T'$ with $i_m$ added to the left and

\[
\text{a copy of } T' \text{ appended to } i_m;
\]

$S := T \setminus T'$ (= the subtrie rooted in $i_m$);

\[
\text{count}(S, i_m);
\]

delete the infrequent itemsets from $S$;

procedure $\text{count}(S, i_m) ::$

for every transaction $t$ including item $i_m$ do

\[
\text{for every itemset } I \text{ in } S \text{ do}
\]

\[
\text{if } t \text{ supports } I \text{ then } I.\text{support}++;
\]
The sorting requires some simple preprocessing.

Counting is done “efficiently”: once a bucket is not included in a transaction, the transaction does not go any deeper in the trie.

The newest implementation (that combines and improves upon the two versions included in the FIMI’03 comparison) avoids unnecessary copying of buckets and deletions of subtries.

Both the database and the trie reside in main memory.
The number of database queries equals

\[ \sum_{A \neq \emptyset} \sum_{j=1}^{sm(A)-1} supp(\{j\} \cup A \setminus \{la(A)\}) \]

where \(m\) is the number of transactions, \(n\) is the number of frequent items, and for a non-empty itemset \(A \subseteq \{1, 2, \ldots, n\}\) \(sm(A)\) is its smallest number and \(la(A)\) is its largest number.

The proof relies on the fact that in order for a bucket to occur in the trie the path to it (except for the root) should be frequent, and on the observation that this particular bucket is “questioned” every time a transaction follows this same path.
DF–6

experiments

Database chess

execution time DF

number of frequent sets (scale on right axis)

number of sets in 1,000,000s

relative support (%)

Database mushroom

execution time DF

number of frequent sets (scale on right axis)

number of sets in 1,000,000s

relative support (%)

Database T10I4D100K

execution time DF

number of frequent sets (scale on right axis)

number of sets in 1,000,000s

relative support (%)

Database T40I10D100K

execution time DF

number of frequent sets (scale on right axis)

number of sets in 1,000,000s

relative support (%)
The $DF$ algorithm is simple and transparent.

The $DF$ algorithm performs well on sparse datasets (e.g., real transaction databases).

Future research: reduce the number of database passes. This may be achieved by adding two or three subtries at a time in each iteration of the main loop. Also, an own dedicated memory management system might improve the runtime.