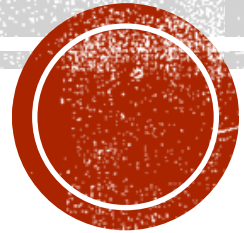
IMPROVING THE EFFICIENCY OF APRIORI

"How can we further improve the efficiency of Apriori-based mining?" Many variations of the Apriori algorithm have been proposed that focus on improving the efficiency of the original algorithm. Several of these variations are summarized here.



HASH-BASED TECHNIQUE

A hash-based technique can be used to reduce the size of the candidate k-itemsets, C_k, for k >1.

For example, when scanning each transaction in the database to generate the frequent 1-itemsets, L_1, we can generate all the 2-itemsets for each transaction, hash (i.e., map) them into the different buckets of a hash table structure, and increase the corresponding bucket counts (Figure in next slide). A 2-itemset with a corresponding bucket count in the hash table that is below the support threshold cannot be frequent and thus should be removed from the candidate set. Such a hash-based technique may substantially reduce the number of candidate k-itemsets examined (especially when k=2).

Create hash table H_2 using hash function $h(x, y) = ((order\ of\ x) \times 10 + (order\ of\ y))\ mod\ 7$ →

H_2

bucket address	0	1	2	3	4	5	6
bucket count	2	2	4	2	2	4	4
bucket contents	{I1, I4} {I3, I5}	{I1, I5} {I1, I5}	{I2, I3} {I2, I3} {I2, I3} {I2, I3}	{I2, I4} {I2, I4}	{I2, I5} {I2, I5}	{I1, I2} {I1, I2} {I1, I2} {I1, I2}	{I1, I3} {I1, I3} {I1, I3} {I1, I3}

Hash table, H_2, for candidate 2-itemsets. This hash table was generated by scanning Table 6.1's transactions while determining L_1. If the minimum support count is, say, 3, then the itemsets in buckets 0, 1, 3, and 4 cannot be frequent and so they should not be included in C_2.

TRANSACTION REDUCTION

A transaction that does not contain any frequent k-itemsets cannot contain any frequent (k +1)-itemsets. Therefore, such a transaction can be marked or removed from further consideration because subsequent database scans for j-itemsets, where j >k, will not need to consider such a transaction.

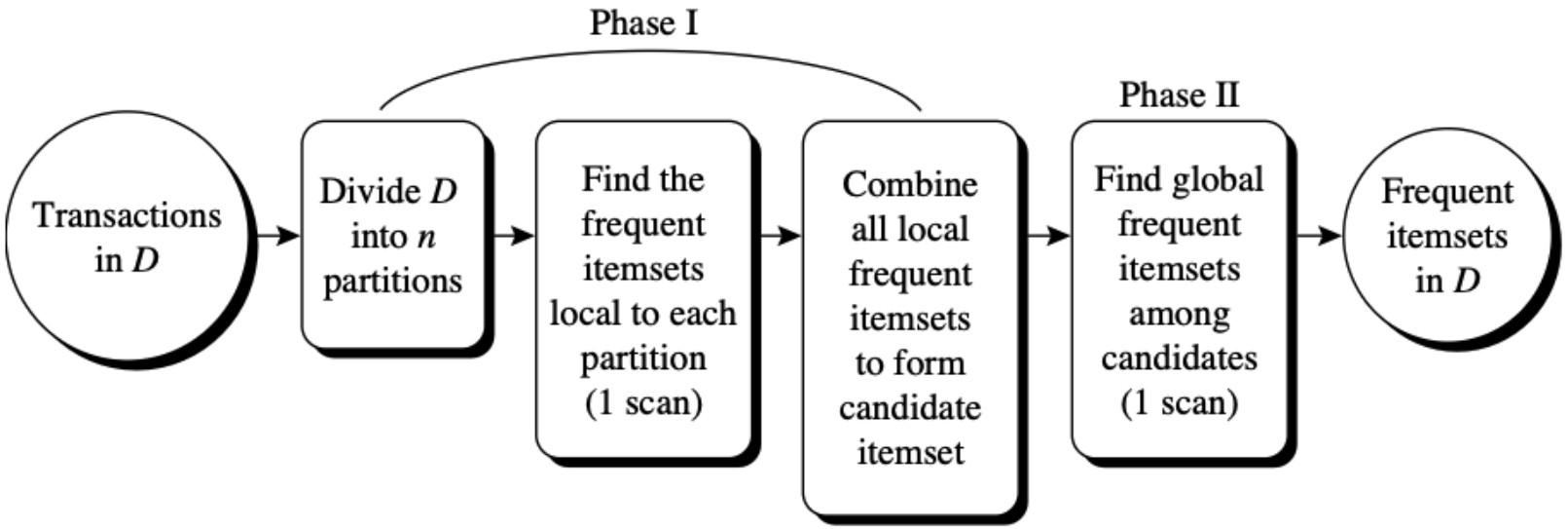
PARTITIONING`

A partitioning technique can be used that requires just two database scans to mine the frequent itemsets (Figure in next slide). It consists of two phases. In phase I, the algorithm divides the transactions of D into n nonoverlapping partitions. If the minimum relative support threshold for transactions in D is min sup, then the minimum support count for a partition is min sup ×the number of transactions in that partition. For each partition, all the local frequent itemsets (i.e., the itemsets frequent within the partition) are found.

A local frequent itemset may or may not be frequent with respect to the entire database, D. However, any itemset that is potentially frequent with respect to D must occur as a frequent itemset in at least one of the partitions.

Therefore, all local frequent itemsets are candidate itemsets with respect to D. The collection of frequent itemsets from all partitions forms the global candidate itemsets with respect to D. In phase II, a second scan of D is conducted in which the actual support of each candidate is assessed to determine the global frequent itemsets. Partition size and the number of partitions are set so that each partition can fit into main memory and therefore be read only once in each phase.

PARTITIONING



Mining by partitioning the data.

SAMPLING

The basic idea of the sampling approach is to pick a random sample S of the given data D, and then search for frequent itemsets in S instead of D. In this way, we trade off some degree of accuracy against efficiency. The S sample size is such that the search for frequent itemsets in S can be done in main memory, and so only one scan of the transactions in S is required overall. Because we are searching for frequent itemsets in S rather than in D, it is possible that we will miss some of the global frequent itemsets. To reduce this possibility, we use a lower support threshold than minimum support to find the frequent itemsets local to S (denoted L^S).

The rest of the database is then used to compute the actual frequencies of each itemset in LS. A mechanism is used to determine whether all the global frequent itemsets are included in L^S. If L^S actually contains all the frequent itemsets in D, then only one scan of D is required.

Otherwise, a second pass can be done to find the frequent itemsets that were missed in the first pass. The sampling approach is especially beneficial when efficiency is of utmost importance such as in computationally intensive applications that must be run frequently.

DYNAMIC ITEMSET COUNTING

A dynamic itemset counting technique was proposed in which the database is partitioned into blocks marked by start points. In this variation, new candidate itemsets can be added at any start point, unlike in Apriori, which determines new candidate itemsets only immediately before each complete database scan.

The tech-nique uses the count-so-far as the lower bound of the actual count. If the count-so-far passes the minimum support, the itemset is added into the frequent itemset collectionand can be used to generate longer candidates. This leads to fewer database scans thanwith Apriori for finding all the frequent itemsets.