A New Method for Similarity Indexing of Market Basket Data

Charu C. Aggarwal
IBM T. J. Watson Research Center
Yorktown Heights, NY 10598
charu@watson.ibm.com

Joel L. Wolf
IBM T. J. Watson Research Center
Yorktown Heights, NY 10598
jlw@watson.ibm.com

Philip S. Yu
IBM T. J. Watson Research Center
Yorktown Heights, NY 10598
psyu@watson.ibm.com

Abstract

In recent years, many data mining methods have been proposed for finding useful and structured information from market basket data. The association rule model was recently proposed in order to discover useful patterns and dependencies in such data. This paper discusses a method for indexing market basket data efficiently for similarity search. The technique is likely to be very useful in applications which utilize the similarity in customer buying behavior in order to make peer recommendations. We propose an index called the *signature table*, which is very flexible in supporting a wide range of similarity functions. The construction of the index structure is independent of the similarity function, which can be specified at query time. The resulting similarity search algorithm shows excellent scalability with increasing memory availability and database size.

1 Introduction

This paper addresses the problem of retrieval and similarity search for market basket data. In recent years, the progress of bar code technology has made it possible to collect information containing consumer transaction data efficiently. Information about market baskets is collected in the form of sets of items which are bought together in a *transaction*. Let U denote the universal set of items. Thus, a transaction T is a set of items which is a subset of U, and represents those items from U which were bought in the corresponding transaction. Since the correlations in the buying behavior of the different items may be deduced from transactions, they may be used in order to make decisions in many target marketing applications.

The market basket problem has been studied extensively with reference to the issue of finding association rules between the different items [2, 3]. In this paper, we will discuss this problem from the context of *similarity search*. For a given *target transaction*, we wish to find the k transactions which are most similar to it. The straightforward solution of using a sequential scan may require considerable I/O for very large data collections. This is especially true for transaction data, which is usually of the order of gigabytes or terabytes.

The problem of indexing has been studied considerably in the database literature from the perspective of multidimensional quantitative data. Some examples of data structures for performing indexing are the R-Tree and its variants [1, 5, 12, 22]. These data structures can support many kinds of queries, such as point queries, range queries, or similarity to a predefined target. After the introduction of the R-Tree, substantial work was done on improving indexing techniques for high dimensional data [6, 15, 16]. As the dimensionality increases, the performance of these methods deteriorates rapidly because of the so called *dimensionality curse*. As a rule of thumb, when the dimensionality is more than 10, none of the above methods work well. A variety of indexes and query processing techniques have been proposed in the database literature which are specially suited to the similarity search query [4, 7, 13, 14, 15, 17, 20, 21, 23]. Recently, the pyramid technique was proposed for range queries, which reduces the performance deterioration caused by high dimensionality [6]. It has also been observed [20], that the query performance is quite sensitive to the similarity (or distance) function which is used.

A typical transaction contains only a few items out of a universe of hundreds or thousands of items [2]. Furthermore, the items in a transaction are closely correlated. For example, in a database for a supermarket application, if a transaction contains the item "Milk", then it may often also contain the items "Bread" or "Butter". The total dimensionality of transaction data is clearly out of the range of the available indexing techniques for continuous valued attributes. On the other hand, the sparsity and correlations in transaction data provide an opportunity

which can be exploited in order to design efficient index structures.

It has been observed in earlier work [20], that index structures are often sensitive to the nature of the similarity function which is used. Two simple similarity functions between transactions are the number of matches and the hamming distance. The number of matches between two transactions is defined by the number of items which were bought in both. In other words, if T_1 and T_2 are two transactions, then the match function between T_1 and T_2 is defined by the cardinality of $T_1 \cap T_2$. The hamming distance between two transactions is defined to be the number of items which were bought in one of the transactions, but not both. Thus, the hamming distance between the transactions T_1 and T_2 is the sum of the cardinalities of the two sets $(T_1 - T_2)$ and $(T_2 - T_1)$. In reality, good similarity functions for transaction data are much more complex than the above mentioned functions. Often the similarity criterion may be complex function of the number of matches and the hamming distance.

The inverted index is a data structure which can support queries based on the number of matches between the target and the database transactions. It relies on the sparsity of the data in order to provide efficient responses to similarity queries [18] and is widely used for information retrieval applications. Unfortunately, the inverted index cannot efficiently resolve queries in which the similarity criterion can be any general function of the number of matches and hamming distance between the target and database transactions. It also cannot easily support queries which are variations on the traditional similarity search problem (eg. range queries or multi-target variants of similarity queries). Furthermore, although transaction data is sparse, the degradation in performance of inverted indexes with increasing density is too steep to be of practical use for many real applications.

We will introduce a data structure called the *Signature Table*, and show how to use it effectively for similarity search. This data structure is quite flexible in being able to support different kinds of similarity functions. Furthermore, it shows excellent scalability with increasing database size. This is a characteristic which is crucial to effective deployment for very large databases.

This paper is organized as follows. In the next section, we will discuss the nature of the similarity function and the kinds of queries that we would like the index structure to support. In Section 3, we will introduce the signature table, and discuss the details of its construction. In Section 4, we will discuss the branch and bound method which uses the signature table for effective similarity search. Section 5 discusses the empirical results, whereas Section 6 is a conclusion and summary.

1.1 Contributions of this paper

This paper discusses a flexible indexing structure for similarity search which is applicable to a wide variety of similarity functions. We would like to be able to perform similarity search when the similarity function between two transactions is a general function of the number of matches and the hamming distance. We achieve this goal under certain very reasonable constraints on the similarity function. The process of building the signature table is independent of the similarity function, which can be specified at query time. Furthermore, the query processing techniques discussed in this paper have the potential to be extended to more complex variations of similarity search queries with multiple similarity functions and/or targets. The (percentage) pruning efficiency of our technique increases with database size. This is a useful property, given the huge sizes of sales transaction databases.

2 The Similarity Function

In this section we will discuss the general properties of the similarity function which are supported by the techniques in this paper. Let x be the number of items in which the transactions T_1 and T_2 match, and let y be the number of items in which T_1 and T_2 differ. Then the similarity between T_1 and T_2 is a function of both x and y. We shall denote this function by $f(x, y)$. For the sake of convention, we will assume that the similarity function is such that higher values imply greater similarity. The index structure proposed in this paper is capable of handling maximization techniques on this general function $f(x, y)$, provided that it satisfies the following two constraints:

$$\frac{\Delta f(x,y)}{\Delta x} \geq 0 \tag{1}$$

$$\frac{\Delta f(x,y)}{\Delta y} \leq 0 \tag{2}$$

The conditions on the function ensure that it is an increasing function in terms of the number of matches and is decreasing in terms of the hamming distance. These are quite reasonable assumptions for similarity functions, since a higher number of matches imply greater similarity, whereas a greater hamming distance implies less similarity. Some examples of functions which satisfy this requirement are as follows:

(1) **Hamming distance (minimization):** When restated in maximization form, we have $f(x, y) = 1/y$.

(2) **Match to hamming distance ratio:** $f(x, y) = x/y$.

(3) **Cosine:** Let S and T denote two transactions, such that $\#S$ and $\#T$ are the number of items in S and T respectively. The cosine of the angle between S and T is defined as follows:

$$cosine(S,T) = \frac{x}{\sqrt{\#S} \cdot \sqrt{\#T}}$$
$$= \frac{x}{\sqrt{(2 \cdot x + y - \#T)} \cdot \sqrt{\#T}}$$

A variant of the above mentioned cosine function is often used to measure similarity in information retrieval applications [11].

A large number of other functions also satisfy the above property. In order to set the ground for further discussion, we shall prove a result which is key to the correctness of our similarity procedure for the function $f(\cdot,\cdot)$.

Lemma 2.1 *Let f be a function satisfying:*

$$\frac{\Delta f(x,y)}{\Delta x} \geq 0$$
$$\frac{\Delta f(x,y)}{\Delta y} \leq 0$$

Let γ be an upper bound on the value of x and θ be a lower bound on the value of y. Then $f(\gamma,\theta)$ is an upper bound on the value of the function $f(x,y)$.

Proof: let (x^0,y^0) be any realization of the value of (x,y). Then, $x^0 \leq \gamma$ and $y^0 \geq \theta$. It follows from the above conditions that:

$$f(x^0,y^0) \leq f(\gamma,y^0) \leq f(\gamma,\theta) \quad (3)$$

The result follows. ■

2.1 The similarity search query and its variations

When stated in simple form, the similarity query is as follows:

For a given target transaction T, find the k transactions which maximize the similarity function $f(x,y)$, where x is the number of matches, and y is the hamming distance with the target transaction.

An interesting variation of this query is one in which we have multiple targets, and we wish to maximize some function of the similarity of the transactions to the different targets. For example, we may wish to maximize the average similarity of the retrieved transactions to n different targets.

Other variations on this query structure include range queries such as those in which we wish to find all the transactions whose similarity to the target is at least equal to a predefined threshold t. More generally, let $f_1(\cdot,\cdot),\ldots f_n(\cdot,\cdot)$ be n similarity functions, and let $t_1,\ldots t_n$ be n threshold values. The range query finds all transactions which satisfy the constraints $f_1(\cdot,\cdot) \geq t_1,\ldots f_n(\cdot,\cdot) \geq t_n$. An example of such a query is one in which we wish to find all transactions which have at least p items in common and at most q items different from the target.

In this paper, our primary focus is on the nearest neighbor query as stated in its purest form of finding the closest transaction to the target. However, since the branch and bound query processing technique is applicable to all optimization functions, the methods discussed in this paper can be easily extended to the above variations. In a later section of the paper, we will briefly discuss the slight modifications required in order to support these queries.

3 The Signature Table

In this section, we will discuss the definition and characterization of the signature table. We shall first discuss some notations and terminology in order to facilitate further discussion.

A *signature* is a set of items. The items in the original data are partitioned into sets of signatures. Consider the case when the items in the original data are $\{1,\ldots,20\}$. Then one possible partition of these items into signatures could be $P = \{1,2,4,6,8,11,18\}$, $Q = \{3,5,7,9,10,16,20\}$, and $R = \{12,13,14,15,17,19\}$.

A transaction T is said to *activate* a signature A at level r if and only if $|A \cap T| \geq r$. This level r is referred to as the *activation threshold*. For example, consider the transaction $T = \{2,6,17,20\}$. Then, the transaction T will activate the signatures P, Q, and R at level $r = 1$, and will activate only the signature P at level $r = 2$.

The set of items are partitioned into K sets of signatures. We shall refer to this value of K as the *signature cardinality*. The supercoordinate of a transaction exists in K-dimensional space such that each dimension of the supercoordinate has a unique correspondence with a particular signature and vice-versa. Each dimension in this K-dimensional supercoordinate is a 0-1 value which indicates whether or not the corresponding signature is activated by that transaction. Thus, if the items are partitioned into K signatures $\{S_1,\ldots S_K\}$, then there are 2^K possible supercoordinates. Each transaction maps on to a unique supercoordinate, though multiple transactions may map into the same supercoordinate. If S_{i_1}, S_{i_2}, $\ldots$ S_{i_l} be the set of signatures which a transaction activates, then the supercoordinates of that transaction are defined by setting the $l \leq K$ dimensions $\{i_1,i_2,\ldots i_l\}$ in this supercoordinate to 1 and the remaining to 0.

The signature table consists of a set of 2^K entries.

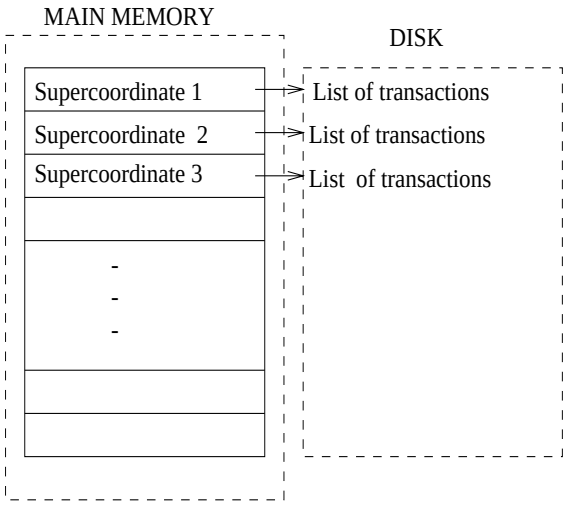

Figure 1: Illustration of the signature table

One entry in the signature table corresponds to each possible supercoordinate. Thus, the entries in the signature table create a partition of the data. This partition will be used for the purpose of similarity search. The signature table is stored in main memory, whereas the actual transactions which are indexed by each entry of the signature table are stored on disk. This constrains the value of the signature cardinality since the 2^K entries of the signature table need to be stored in main memory. Each entry in the signature table points to a list of pages which contain the transactions indexed by that supercoordinate. The signature table is illustrated in Figure 1.

A signature can be understood as a small category of items from the universal set of items U. Thus, if the items in each signature are closely correlated, then a transaction is likely to activate a small number of signatures. These signatures provide an idea of the "approximate" pattern of buying behavior for that transaction. The concept of using approximate representations in order to index data has been explored in earlier work [8, 9, 10]. Our method of representation, data structure and query techniques are very different, being specially designed for the similarity search problem.

3.1 Details of signature table construction

In order to explain the process of signature table construction we shall introduce some additional notation and terminology. An *itemset* I is defined to be a set of items satisfying $I \subseteq U$. Here U is the universal set of items. A *k-itemset* is an itemset which has k items in it. The *support* of an itemset I is defined to be the percentage of transactions in the database which contain I as a subset.

The effective construction of the signature table requires us to partition the universal set of items U into subsets of items which form the signatures. We would like the set of items in each signature to be closely correlated. This is achieved by using the following procedure. We construct a graph such that each node of the graph corresponds to an item. For every pair of items such that the corresponding itemset has a predefined minimum support, we add an edge between the nodes. We also add a weight to each edge which is the inverse of the support of that pair of items. Thus, when the support of a pair of items is high, the weight of the corresponding edge is low, and vice-versa.

We would like to find a partition of the items such that the sum of the supports of the items in each partition is similar and the cross-correlations among the items in the different partitions are as small as possible. In the transformation discussed above this translates to the problem of finding the graph partitioning such that the weight of the edges crossing the partitions of the graph is as small as possible. The desire to keep partitions evenly balanced arises from the need to create the signature table in such a way that the transactions are well distributed over the different entries. The desire to keep the cross-correlations between the different partitions small is motivated by the aim of keeping very closely correlated[1] items in each signature.

Unfortunately, the weighted graph partitioning problem is notoriously difficult to solve. Thus, this approach is too time intensive and impractical for most problems. Instead, we choose to use a variant of the single linkage clustering algorithm [19] in order to find the optimal partitions. The disadvantage of single-linkage algorithms is that they are often likely to lead to long straggly clusters [19]. However, space and time-optimal algorithms are available in order to perform single-linkage clustering, and this is clearly attractive when the total number of items is very large. Briefly stated, the single-linkage clustering algorithm is as follows:

(1) Construct a node for each item.

(2) Let the distance between each pair of nodes be defined by the inverse of the support of the corresponding 2-itemset.

(3) Apply the greedy minimum spanning tree algorithm to the resulting graph in order to generate the single linkage clusters. Add edges one by one to the graph, starting with the null graph which contains only nodes, but no edges. The edges are added in order of increasing distance. This strategy ensures that if the support of a pair of items is high, then they are

[1]Another reason for desiring the correlation property is that we would like only a small number of signatures to be activated by a transaction. This is important in providing efficiency for our similarity search procedure.

likely to belong to the same connected component. The mass of a connected component in this graph is defined as the sum of the supports of the individual items in that component. Remove any component which is such that its mass exceeds a predefined percentage of the sum of the supports of all the individual items. We shall refer to this threshold value of the mass as the *critical mass*. This set of items which satisfies the critical mass criterion forms a signature set. We continue the process of adding edges and removing components of critical mass until each item belongs to some signature set.

Note that the critical mass criterion in step (3) for determining when to remove a component from the graph determines the signature cardinality K. If the critical mass for a signature set is low, then the value of the signature cardinality K is high, and vice-versa. Higher values of K are desirable, since this results in a finer granularity of partitioning of the data. The empirical section will illustrate that partitions of higher granularity result in better pruning performance. On the other hand, it is also necessary to choose low enough values of K such that the signature table can be held in main memory.

Once the signatures have been determined, the partitions of the data may be defined by using the supercoordinates of the transactions. Each transaction belongs to the partition which is defined by its supercoordinate. The transaction lists are built for each supercoordinate (as illustrated in Figure 1) based on this partition.

4 The Branch and Bound Technique

The use of branch and bound methods for nearest neighbor search has been explored by Roussopoulos et. al. [17]. This method was proposed for continuous valued attributes. The same principles are applicable to the market basket problem. The use of signature sets for organizing transactions is the key to determining good estimations on the similarity between the target transaction and the transactions indexed by the signature table entries.

The branch and bound method is a classical technique in combinatorial optimization. It uses an ordered search method on a partitioning of the data in order to avoid searching many of the partitions. Pruning is done by finding good *optimistic* bounds on the distance of a target point to each set representing a partition of the search space. This bound is an optimistic[2] bound (upper bound) on the value of the similarity function between the target and any transaction in this set. If this optimistic bound is less than than the similarity of the target to some solution R which has already been determined, then the corresponding set may be pruned from contention. This is because none of these transactions can provide better similarity to the target than R. This solution R defines a *pessimistic* bound (or lower bound) on the quality of the optimal solution. The overall idea of the branch and bound method is illustrated in Figure 2. The process of pruning reduces the size of the accessed space substantially, and thereby leads to improved running times.

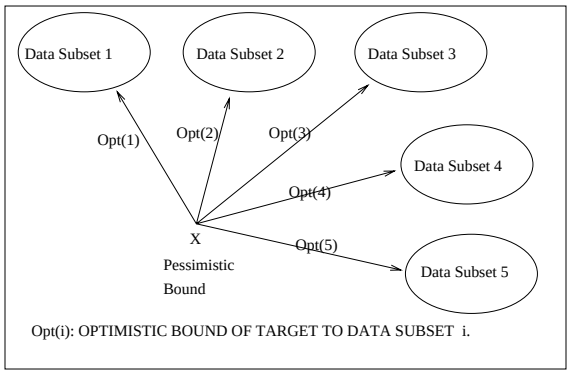

Figure 2: The partitioning techniques in the branch and bound method

We know that each entry in the signature table points to a set of transactions which corresponds to a particular supercoordinate. It is the partitioning created by these supercoordinates which may be used in order to perform the branch and bound method.

Pseudocode for the branch and bound algorithm is given in Figure 3. The description pertains to the case when the single nearest neighbor needs to be found. The modifications for the more general problem are quite simple, and are discussed in Section 4.3. The first step is to compute the optimistic bounds from the target transaction to each entry in the signature table. We perform a main memory sort of the entries in the signature table in decreasing order of the optimistic bounds. This sort provides the order in which the supercoordinates of the signature table should be scanned. The purpose of the sorting is to order the partitions created by the supercoordinates in such a way that the partitions which are most likely to contain the nearest transaction to the target are visited first.

We scan the supercoordinates in the signature table one by one, and compute the similarity between the target transaction and the transactions indexed by that entry. The similarity computation is illustrated in the procedure $EvaluateObjective(\cdot, \cdot)$ of Figure 5. We always keep track of the best candidate which has been found so far. The similarity of this transaction to the

[2]In general, the term "optimistic bound" refers to an upper bound, when the similarity search problem is in maximization form, and a lower bound when the similarity search problem is in minimization form. Recall that our convention indicated that higher values of the function $f(\cdot, \cdot)$ imply greater similarity.

```
Algorithm BranchAndBound(TargetTransaction: T,
                         SignatureTableEntries:{1, ... 2^K})
begin
for each table entry i do
  begin
    Opt(i) = FindOptimisticBound(T, i);
  end
Sort the entries in the table in decreasing order of Opt(i);
BestTransaction = A randomly chosen transaction;
{ BestTransaction keeps track of the transaction with
 the highest similarity value among those scanned so far }
PessimisticBound = EvaluateObjective(BestTransaction, T);
BestValue = EvaluateObjective(BestTransaction, T);
for each table entry i, in decreasing order of Opt(i) do
  begin
  if Opt(i) ≤ PessimisticBound then prune entry i;
  else
    begin
    T(i) = Transactions indexed by entry i;
    for each transaction S ∈ T(i) do
      if EvaluateObjective(S, T) > BestValue then
        begin
        BestValue = EvaluateObjective(S, T);
        BestTransaction = S;
        end
    PessimisticBound = BestValue;
    end
  end;
return(BestTransaction);
end
```

```
Algorithm FindOptimisticBound(TargetTransaction: T,
                              Signature Table Entry: i)
begin
  M_opt = FindOptimisticMatch(T, i);
  D_opt = FindOptimisticDist(T, i);
  return(f(M_opt, D_opt));
end
```

Figure 3: The branch and bound algorithm for indexing market baskets

```
Algorithm FindOptimisticDist(TargetTransaction: T,
                             Signature Table Entry: i)
{ This algorithm finds an optimistic (lower) bound on the
  hamming distance between the target transaction and the
  signature table entry i. A detailed description of this
  technique is provided in Section 4.1 of the paper }

Algorithm FindOptimisticMatch(TargetTransaction: T,
                              Signature Table Entry: i)
{ This algorithm finds an optimistic (upper) bound on the
  matches between the target transaction and the signature
  table entry i. A detailed description of this technique
  is provided in Section 4.1 of the paper }
```

Figure 4: Evaluating the optimistic bounds on the match and hamming distance

```
Algorithm EvaluateObjective(Transaction: S, Transaction: T)
begin
  x = Number of matches between S and T;
  y = Hamming Distance between S and T;
  return(f(x, y));
end
```

Figure 5: Evaluating the objective function between two transactions

target provides a pessimistic bound on the quality of the similarity function between the target transaction and any of the transactions in the database.[3] As the algorithm scans through the different supercoordinates, it continues to update (increase) the pessimistic bound, as better and better candidates are discovered. At the same time, those entries for which the optimistic bound is less than the current value of the pessimistic bound are pruned from contention.

The performance of the algorithm is sensitive to the criterion used in order to sort the signature table entries. This is because finding closely matching transactions (and hence higher pessimistic bounds) earlier on can significantly improve pruning performance. The use of optimistic bounds in order to perform the sorting is only one possible implementation of the algorithm. Another interesting implementation would be to sort the entries in the signature table based on the similarity function between the respective supercoordinates of the signature table entries and the target. In other words, if B' be the supercoordinate of the target, and B_i be the supercoordinate of signature table entry i, then the sort order may be determined by using the similarity function between B' and B_i for each i. This provides a different order of search of the partitions, though the optimistic bounds are still used in order to evaluate the pruning criterion for a given signature table entry. This can improve the performance when the sort criterion is a better indication of the *average case* similarity between the target and the transactions indexed by a signature table entry. For the purposes of this paper, we always use the optimistic bounds in order to perform the sorting. It now remains to explain how the optimistic bounds may be found.

4.1 Finding optimistic bounds

In order to find an optimistic bound on the similarity between the target transaction and a supercoordinate for the similarity function $f(M, D)$, we first need to find optimistic bounds on the values of M and D. It follows from Lemma 2.1 that if M_{opt} and D_{opt}

[3]The correctness of this assertion is quite obvious, since the value of the similarity function of the closest transaction to the target in a subset of the database is a lower bound on the similarity to the closest transaction in the database.

are the respective optimistic bounds on the values of the match and hamming distance between the target and the supercoordinate (all transactions indexed by that supercoordinate), then an optimistic bound on the similarity function may be computed by evaluating $f(M_{opt}, D_{opt})$. Note that D_{opt} is a lower bound, whereas M_{opt} is an upper bound. This is because lower values of the hamming distance imply greater similarity, whereas higher values of the match function imply greater similarity.

We will now discuss how M_{opt} and D_{opt} may be determined. Let $\{S_1, \ldots, S_K\}$ be the sets of signatures. Let r be the activation threshold, and r_i be the number of items common to each of the K signature sets S_i from the target transaction. Let B be an entry in the signature table, and let the K bits representing its supercoordinate be denoted by $\{b_1 \ldots b_K\}$. The bit b_i corresponds to the signature S_i. The variable $Dist$ will contain the optimistic (lower) bound on the hamming distance between the target transaction and the signature table entry B. In the beginning, $Dist$ is initialized to 0. In order to find an optimistic (lower) bound on the hamming distance from a target transaction to all transactions indexed by the supercoordinate B we process each of the signature sets S_j for $j \in \{1, \ldots K\}$ and add an amount to $Dist$, which is calculated in the following way:

(1) For each $j \in \{1, \ldots, K\}$ such that $b_j = 0$, we know that every transaction indexed by that entry must have less than r items from S_j in it. On the other hand, if the target transaction has $r_j \geq r$ items in common with that signature, we know that an optimistic bound on the hamming distance between the target and any transaction indexed by that signature entry must be $r_j - r + 1$. Thus, for each $j \in \{1, \ldots K\}$ such that $b_j = 0$ we add $\max\{0, r_j - r + 1\}$ to $Dist$.

(2) For each $j \in \{1, \ldots, K\}$ such that $b_j = 1$, we know that every transaction indexed by that entry must have at least r items from S_j in it. On the other hand, if the target transaction has $r_j < r$ items common with that signature, we know that an optimistic bound on the similarity between the target and any transaction indexed by that signature entry must be $r - r_j$. Thus, for each such entry we add $\max\{0, r - r_j\}$ to $Dist$.

After all the K signatures have been processed, the value of $Dist$ represents an optimistic (lower) bound on the hamming distance from the target to any transaction indexed by that entry in the signature table. At this stage, D_{opt} may be set to the value of $Dist$. The above description pertains to the *FindOptimisticDist*$(\cdot, \cdot)$ of Figure 4. Correspondingly, the procedure *FindOptimisticMatch*$(\cdot, \cdot)$ of Figure 4 illustrates the method for finding the optimistic (upper) bound on the number of matches. In order to find optimistic bounds on the matches between the target transaction and a signature entry, the method is very similar. The variable $Match$ denotes the optimistic (upper) bound on the number of matches between the target and any transaction indexed by supercoordinate B. In order to determine the optimistic bounds, we update $Match$ in the following way for each signature set:

(1) For each $j \in \{1, \ldots K\}$ such that $b_j = 0$, we add $\min\{r - 1, r_j\}$ to $Match$.

(2) For each $j \in \{1, \ldots K\}$ such that $b_j = 1$, we add r_j to $Match$.

After processing the K signatures, the value of M_{opt} is set to the value of $Match$. Once M_{opt} and D_{opt} have been determined, the optimistic bound on the similarity of the target to a supercoordinate is determined by computing $f(M_{opt}, D_{opt})$. This is achieved by the procedure *FindOptimisticBound*$(\cdot, \cdot)$ of Figure 3.

4.2 Speeding up by early termination

The method can be speeded up significantly using early termination methods. The idea in early termination is that it is possible to terminate the algorithm after searching a predefined percentage of the transactions. The current best solution provides an *approximation* on the nearest neighbor. It is also possible to provide an estimate of how close this solution is to the nearest neighbor distance. Let $\mathcal{S}$ denote the set of table entries which have not been explored till (early) termination. Then an optimistic bound on the quality of the best similarity of the target transaction to any transaction indexed by the remaining signature table entries is given by $\max_{i \in \mathcal{S}}\{Opt(i)\}$ defined in Figure 3. If this value is less than what was obtained at termination, then we know that we have found the true nearest neighbor. Otherwise, this value provides an upper bound on the quality of the best possible solution. Another way in which one could terminate the algorithm is when the best transaction found so far is within a reasonable similarity difference from the optimistic bounds of the unexplored table entries. In this way it is possible to provide a *guarantee* on the quality of the presented solution.

4.3 Generalizations to other similarity and range queries

This technique can be generalized easily to the k-nearest neighbor problem. The only difference is that at each stage of the algorithm, we maintain the k best candidates found so far. The pessimistic bound is determined by the similarity function of the target transaction with the kth best candidate found so far. A

supercoordinate may be pruned if its optimistic bound is less than this value.

Similarity queries for multiple targets can also be supported by the signature table. Consider for example, the case when the similarity of a transaction to a set of targets is defined by the average similarity of the transaction to each of the targets. In this case, the optimistic bound for a signature table entry is obtained by averaging the optimistic bounds of the signature table entry to each of the different targets. The rest of the algorithm is exactly the same.

Although the focus of this paper is on the similarity search problem, we mention in the passing that the techniques discussed in this paper can be easily adapted in order to support range queries such as finding all transactions whose similarity to the target is larger than a predefined threshold t. The only modification required is to the pruning criterion. We need to prune those entries of the signature table such that their optimistic bounds are less than this user specified threshold t. More generally, if $f_1 \ldots f_n$ be n similarity functions, then the range query correponding to $f_1(\cdot,\cdot) \geq t_1 \ldots f_n(\cdot,\cdot) \geq t_n$ can be resolved by pruning those entries in the signature table if the optimistic bounds corresponding to any of the similarity functions are less than the corresponding threshold values.

5 Empirical Results

Synthetic methods for generating market basket data have been discussed earlier by Agrawal et. al. [3]. This method was used in order to test the efficiency of association rule algorithms on market basket data. We used this data generation method in order to test our algorithm.

The first step was to generate $L = 2000$ maximal "potentially large itemsets". These potentially large itemsets capture the consumer tendencies of buying certain items together. All transactions were obtained by combining noisy variations of these potentially large itemsets. We first picked the size of each maximal itemset as a random variable from a Poisson distribution with mean μ_L. Each successive itemset was generated by picking half of its items from the current itemset, and generating the othet half randomly. This ensures that the potentially large itemsets often have common items. Each itemset I has a weight w_I associated with it, which is chosen from an exponential distribution with unit mean. The size of a transaction was drawn from a Poisson distribution with mean μ_T. Each transaction was generated by assigning maximally potentially large itemsets to it in succession. The itemset was assigned to a transaction by rolling an L-sided weighted die such that the weight of the side for itemset I was defined by w_I. If an itemset did not fit exactly, it was assigned to the current transaction half

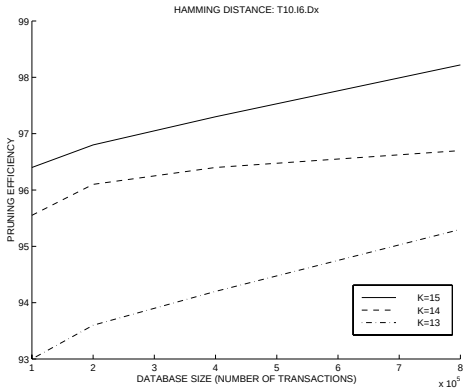

Figure 6: Scaling of pruning performance with database size (hamming distance)

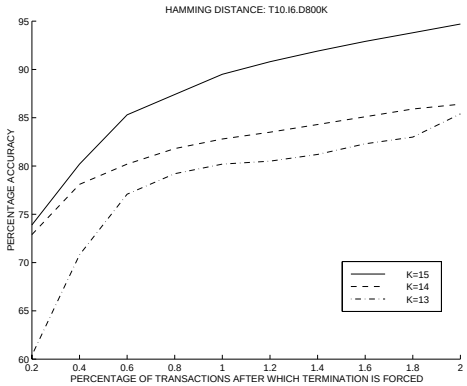

Figure 7: Percentage accuracy scaling with different levels of early termination (hamming distance)

of the time, and was moved to the next transaction the rest of the time. In order to capture the fact that the customers may not often buy all the items in a potentially large itemset together, we added some noise to the potentially large itemsets before adding them to a transaction. For each itemset I, we decided a noise level $n_I \in (0, 1)$. We generated a geometric random variable G with parameter n_I. While adding a potentially large itemset to a transaction, we dropped $\min\{G, |I|\}$ randomly chosen items from the transaction. The noise level n_I for each itemset I was chosen from a normal distribution with mean 0.5 and variance 0.1.

We shall also briefly describe some of the symbols which we have used in order to annotate the data. The three primary factors which vary are the average transaction size μ_T, the size of a maximal potentially large itemset μ_L, and the number of transactions being considered. A data set with $\mu_T = 10$, $\mu_L = 4$ and $100K$ transactions is denoted by T10.I4.D100K.

The algorithm was tested by varying the average transaction size μ_T between 5 to 15 and setting $\mu_L = 6$.

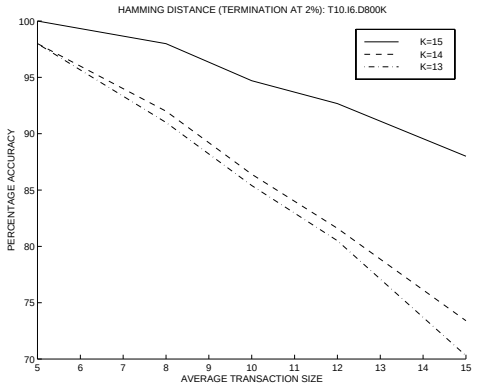

Figure 8: Percentage accuracy scaling with transaction size (hamming distance)

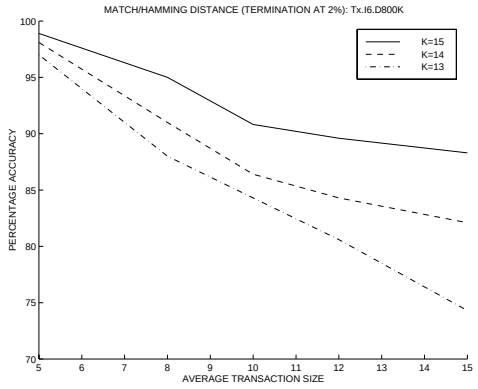

Figure 11: Percentage accuracy scaling with transaction size (match/hamming distance)

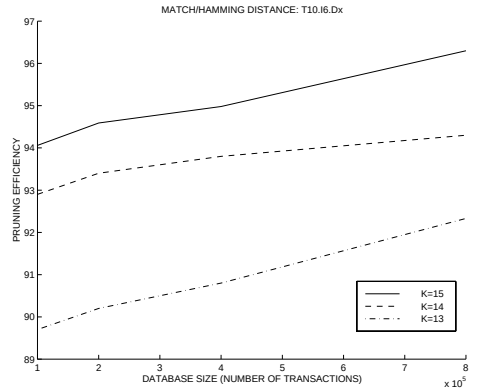

Figure 9: Scaling of pruning performance with database size (match/hamming distance)

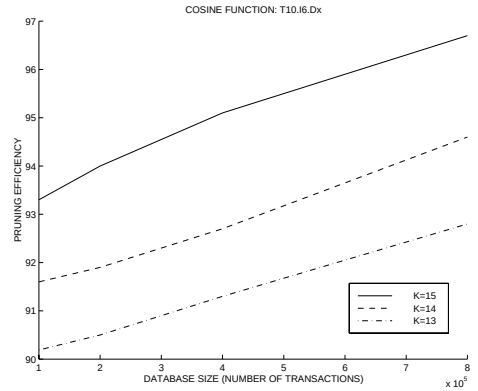

Figure 12: Scaling of pruning performance with database size (cosine)

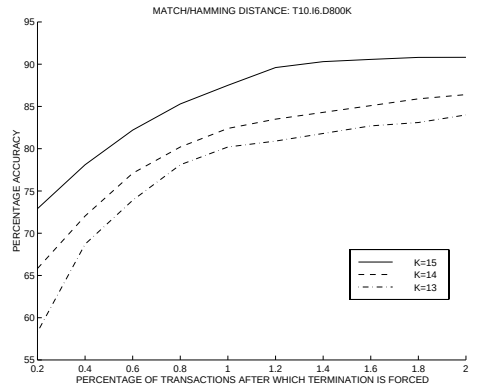

Figure 10: Percentage accuracy scaling with different levels of early termination (match/hamming distance)

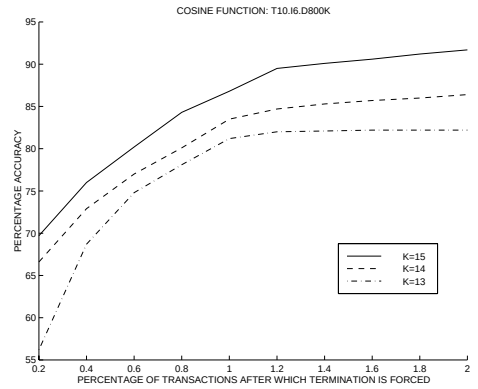

Figure 13: Percentage accuracy scaling with different levels of early termination (cosine)

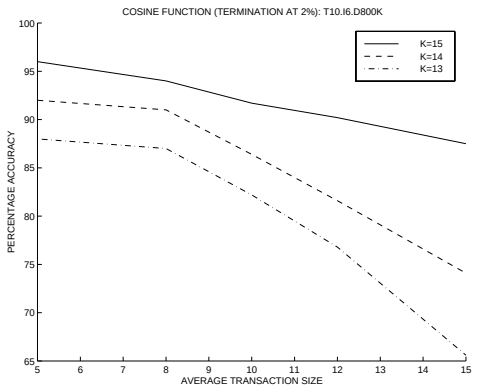

Figure 14: Percentage accuracy scaling with transaction size (cosine)

For all experiments, we fixed the value of the activation[4] threshold to 1. We tested the algorithm for three different objective functions; the hamming distance, the match to hamming distance ratio, and the cosine function.

We tested two performance functions with our algorithm:

(1) Pruning efficiency: The pruning efficiency of the algorithm was defined as the percentage of the data which was pruned by the branch and bound technique when the algorithm was run to completion.

(2) Accuracy: The accuracy was defined as the percentage of the time that the nearest neighbor was found when the algorithm was terminated after accessing a predetermined percentage of the data.

We tested the scalability of the algorithm with respect to three different parameters:

(1) Database Size: We tested how the performance of the algorithm varied with the the number of transactions in the database. We found that *the performance of the algorithm in terms of pruning efficiency improved with database size.* This is highly desirable, because the problem of similarity indexing is most relevant for very large databases.

(2) Transaction Size: As the average transaction size increases, the similarity search problem becomes more difficult because of the increased density of the data. We tested the performance variation of the algorithm with increasing transaction size.

(3) Memory Availability: The size of the signature table is constrained by the amount of available memory. The amount of available memory determines the value of the signature cardinality K. We tested how the performance of the algorithm was affected for differing values of K. Again, we reached the conclusion that the *performance of the algorithm in terms of accuracy and pruning efficiency improved with memory availability.*

The scalability with increasing database size and memory availability are clearly greatly advantageous features, given the current trends.

Since we observed very similar results for all the similarity functions that we tested, we shall restrict ourselves to explaining the performance results for only one of the objective functions. The performance results for the others are very similar, and are provided as confirmation of the flexibility of our index structure in being able to support different similarity functions efficiently at query time.

We tested the variation in pruning performance of the algorithm with increasing database size. The transactions were generated from the distribution T10.I6.Dx, where x denotes the varying database size. The performance results for the hamming distance function are indicated in Figure 6. On the X-axis we have indicated the database size in number of transactions, whereas on the Y-axis we have indicated the pruning performance of the algorithm in terms of the percentage of the transactions pruned. We have illustrated three curves, one each for the values of the signature parameters $K = 13, 14$, and 15. For these values of K, the number of signature table entries are 2^{13}, 2^{14} and 2^{15} respectively. In this range of values for K, the signature table can easily fit into memory. In fact, it is possible to use substantially larger values of K for better query performance.

As is apparent from Figure 6, the algorithm pruned 96% to 99% percent of the transactions when the value of the signature cardinality K was 15. Furthermore, the performance of the algorithm improved substantially when the database sizes were increased. The reason for this is as follows: after evaluating a fixed percentage of the transactions, the value of the pessimistic bound in the case of larger databases is higher. The pessimistic bound is defined by the value of the similarity functions of the closest transaction to the target among those evaluated so far. Hence, for larger database sizes it is statistically expected to be higher. Since a signature table entry is pruned only when its optimistic bound is less than this global pessimistic bound, it is expected that the pruning is more likely for a given signature table entry for larger databases. To understand this point better, let us consider a database which contains only *two* transactions. After evaluating one of the transactions, the other transaction cannot be pruned from contention unless the similarity function for the first transaction is larger than the optimistic bound to the signature table entry for the second transaction. On the other hand, in a database containing a million

[4]We found from our empirical experiments that for larger transaction sizes, higher values of the activation threshold provided better performance. The empirical studies presented in this paper do not take such improvements into account. A detailed study of this performance dependence will be provided in an expanded version of this paper.

transactions, the value of the pessimistic bound is substantially higher after evaluating half a million of the transactions. Therefore, the likelihood of many of the remaining signature table entries getting pruned is also substantially higher. This characteristic of the algorithm is quite desirable, since the entire relevance of the similarity search problem is motivated by the issue of reducing disk I/O in very large databases.

The performance of the algorithm for differing values of the signature cardinality K is because of the fact that higher values of K create a more fine grained partitioning. Therefore the optimistic and pessimistic bounds are more accurate and the probability of pruning is much higher. Note that the values of K which we have illustrated are quite conservative; in real systems we would pick higher values of K (dictated by main memory constraints), and therefore the corresponding pruning performances would also be substantially better.

Figure 7 illustrates the accuracy of the algorithm for the hamming distance function for varying levels of early termination for the data set T10.I6.D800K. We varied the termination criterion from 0.2% of the transactions to a maximum of 2%. As illustrated, the algorithm finds the nearest neighbor more than 90% of the time when $K = 15$ and the early termination level is larger than 1%.

The performances of the algorithm for varying transaction sizes on the hamming distance function are illustrated in Figures 8. The early termination level was fixed at 2%. It is apparent that the accuracy of the algorithm is reduced for longer transaction sizes. The reason for this is that for longer transaction sizes, each transaction activated a larger number of signatures. Therefore, most transactions mapped on to signature table entries which had a larger number of 1 bits. This resulted in a slight reduction of the pruning efficiency. For longer transaction sizes, it is more desirable to choose higher values of the signature cardinality K and also higher values of the activation threshold.

The performances of the algorithm for the same evaluation criteria but a different similarity function (match to hamming distance ratio) are illustrated in Figures 9, 10, and 11; and the performances for the cosine function are illustrated in Figures 12, 13 , and 14 respectively. For a given set of data, exactly the same signature table was used in order to test all the three similarity functions. As illustrated, the trends in these curves are very similar to the corresponding performance curves for the hamming distance function. This is evidence of the robustness and flexibility of our technique for using different similarity functions at query time.

5.1 Performance of inverted indices

Avg. Tr. Size	Percentage Accessed
5	3.32
8	7.45
10	9.37
12	14.50
15	20.10

Table 1: Minimum Percentage of transactions accessed by an inverted index (without scattering)

Since market basket data are high dimensional Boolean data, one possible alternative for this problem is the inverted index. An inverted index for an item stores all the Transaction Identifiers (TIDs), such that the corresponding transactions contain that item. Although the inverted index is vey well suited to queries which are based on the number of matches between the target record and the transactions, they are not so well suited to more sophisticated similarity functions of both the match and the hamming distance, or for complex multi-target variations of similarity queries. For each of the items in the target transaction, it is necessary to access all the transactions indexed by the items in the target, and calculate the similarity between the transactions and the target. Thus, the actual transactions need to be accessed from the database in a two phase process. In the first phase, all the TIDs of the transactions which contain any of the target items are found using the inverted index. In the second phase, the transactions are accessed from the original database, and the similarity of each of these transactions to the target is calculated. As we see from Table 1, the proportion of transactions which need to be accessed is quite considerable when the transaction sizes are larger. Furthermore, we need to go back to the original database in order to calculate the similarity of each such transaction to the target. Since these transactions may not lie on contiguous pages, this may result in a *page-scattering effect* (or random disk access effect), which worsens the performance further. In other words, since the transactions are scattered over many different pages and all database accesses are granular in terms of page sizes, this may result in many unnecessary transactions being retrieved. This effect is such that even if 5% of the transactions in the database need to be accessed, it may be required to access almost the entire database. Table 1 does not take this effect into account. Clearly, the effect of page scattering may result in almost all the transactions in the entire database being accessed for most of the cases illustrated in Table 1. In comparison, the signature table is able to find the best solution a large percentage of time at an early termination level of $0.2 - 2\%$ of the transactions.

6 Conclusions and Summary

In this paper, we discussed a technique for indexing market basket data. The technique can be adapted in order to resolve *approximate* similarity queries. The method for building the index structure is independent of the similarity function which can be specified at query time. The method shows very good pruning and accuracy performance and scales well with database size and memory availability. The query processing techniques discussed in this paper have the potential to be extended to more complex similarity functions with multiple targets, or range queries with multiple similarity functions. Our future research will concentrate on these techniques.

References

[1] C. C. Aggarwal, J. L. Wolf, P. S. Yu, M. Epelman. The S-Tree: An efficient index for multi-dimensional objects. *Proceedings of the International Symposium on Spatial Databases.* pages 350-370, Berlin, Germany, July 1997.

[2] R. Agrawal, T. Imielinski, A. Swami. Mining Association Rules between sets of items in very large databases. *Proceedings of the ACM SIGMOD Conference*, pages 207–216, 1993.

[3] R. Agrawal, R. Srikant. Fast Algorithms for Mining Association Rules in Large Databases. *Proceedings of the 20th VLDB Conference*, pages 478–499, 1994.

[4] S. Arya. Nearest Neighbor Searching and Applications. Ph. D. Thesis, University of Maryland, College Park, MD, 1995.

[5] N. Beckman, H.-P. Kriegel, R. Schneider, B. Seeger. The R*-Tree: An Efficient and Robust Method for Points and Rectangles. *Proceedings of the ACM SIGMOD Conference.* 322–331, 1990.

[6] S. Berchtold, C. Böhm, H.-P. Kriegel. The Pyramid Technique: Towards Breaking the Curse of Dimensionality. *Proceedings of the ACM SIGMOD Conference*, pages 142–153, June 1998.

[7] S. Berchtold, B. Ertl, D. Keim, H.-P. Kriegel., T. Seidl. Fast Nearest Neighbor Search in High Dimensional Spaces. *Proceedings of the 14th International Conference on Data Engineering,* Orlando, 1998.

[8] C. Faloutsos. Signature Files. *Information Retrieval: Data Structures and Algorithms*, W. B. Frakes, R. Baeza-Yates (Ed.), pages 44-65, 1992.

[9] C. Faloutsos, R. Chan. Fast Text Access Methods for Optical and Large Magnetic Disks: Designs and Performance Comparison. *Proceedings of the 14th VLDB Conference*, pages 280-293, 1988.

[10] C. Faloutsos, S. Christodoulakis. Description and Performance Analysis of Signature File Methods. *ACM TOOIS*, 5 (3), pages 237-257, 1987.

[11] W. B. Frakes, R. Baeza-Yates (Editors). Information Retrieval: Data Structures and Algorithms. *Prentice Hall PTR*, New Jersey, 1992.

[12] A. Guttman. R-Trees: A Dynamic Index Structure for Spatial Searching. *Proceedings of the ACM SIGMOD Conference*, 47–57, 1984.

[13] K. Hinrichs, J. Nievergelt. The Grid File: A Data Structure to Support Proximity Queries on Spatial Objects. *Proceedings of the WG'83*, 100–113, 1983.

[14] R. Jain, D. A. White. Similarity Indexing: Algorithms and Performance. *Proceedings of SPIE Storage and Retrieval for Image and Video Databases IV*, Vol. 2670, San Jose, CA, pages 62–75, 1996.

[15] N. Katayama, S. Satoh. The SR-Tree: An Index Structure for High Dimensional Nearest Neighbor Queries. *Proceedings of the ACM SIGMOD Conference.* pages 369–380, 1997.

[16] K.-I. Lin, H. V. Jagadish, C. Faloutsos. The TV-tree: An Index Structure for High Dimensional Data. *VLDB Journal*, 3 (4), pages 517–542, 1992.

[17] N. Roussopoulos, S. Kelley, F. Vincent. Nearest Neighbor Queries. *Proceedings of the ACM SIGMOD Conference*, pages 71–79, 1995.

[18] G. Salton. Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer. *Addison-Wesley Publishing.*

[19] R. Sibson. SLINK: An optimally efficient algorithm for the single link cluster method. *Computer Journal*, Volume 16, pages 30–34, 1973.

[20] T. Seidl, H.-P. Kriegel. Optimal Multi-Step k-Nearest Neighbor Search. *Proceedings of the ACM SIGMOD Conference*, pages 154–165, 1998.

[21] T. Seidl, H.-P. Kriegel. Efficient User-Adaptable Similarity Search in Large Multimedia Databases. *Proceedings of the 23rd VLDB Conference*, 1997.

[22] T. Sellis, N. Roussopoulos, C. Faloutsos. The R+ Tree: A Dynamic Index for Multidimensional Objects. *Proceedings of the 13th VLDB Conference*, pages 507–518, 1987.

[23] D. A. White, R. Jain. Similarity Indexing with the SS-Tree. *Proceedings of the 12th International Conference on Data Engineering*, New Orleans, USA, pages 516–523, February 1996.