The VLDB Journal (2002) 11: 138–152 / Digital Object Identifier (DOI) 10.1007/s00778-002-0068-7

Efficient similarity search for market basket data

Alexandros Nanopoulos, Yannis Manolopoulos

Department of Informatics, Aristotle University of Thessaloniki, Greece; e-mail: {alex,manolopo}@delab.csd.auth.gr

Edited by R. Ng. Received: August 6, 2001 / Accepted: May 21, 2002
Published online: September 25, 2002 – © Springer-Verlag 2002

Abstract. Several organizations have developed very large market basket databases for the maintenance of customer transactions. New applications, e.g., Web recommendation systems, present the requirement for processing similarity queries in market basket databases. In this paper, we propose a novel scheme for similarity search queries in basket data. We develop a new representation method, which, in contrast to existing approaches, is proven to provide correct results. New algorithms are proposed for the processing of similarity queries. Extensive experimental results, for a variety of factors, illustrate the superiority of the proposed scheme over the state-of-the-art method.

Keywords: Market basket data – Similarity search – Nearest-neighbor – Data mining

1 Introduction

Market basket databases organize collections of transactions. Each transaction consists of a set of items from a specified domain (denoted as the *universal set*). Several organizations store very large databases of market basket data, for instance, retail companies. Recently, the emerging popularity of the Web has produced larger collections of basket data, especially in e-commerce sites, where transactions may represent purchased products, visited pages, etc.

In such cases, several applications, such as recommendation systems, present a new need for query processing in large databases of basket data. Recommendation systems find widespread applications in e-commerce sites by making product recommendations during a customer interaction. One of the most successful recommendation system technologies to date is collaborative filtering (CF) [11,20,17] which works by matching customer preferences to those of other customers in making recommendations (collaborative filtering is used in many of the most successful recommendation systems, such as Amazon.com and CDnow.com). In CF- based recommendation systems the input is the collection of purchased transactions of n customer on m products, which comprise the transactions database D [25]. The most important and computational demanding step in the recommendation procedure is finding a similarity-based neighborhood for the target customer C_t. For C_t, the k most similar transactions from D are identified (according to a given similarity measure that denotes similar purchasing preferences). This procedure forms the C_t–neighborhood [26], which can be used either for predicting the probability that C_t will be interested in a given product, or for providing top-N recommendations, i.e., to recommend a list of products that C_t is most likely to purchase.

Example. Assume an e-commerce site, which trades a collection of text books on databases, depicted in the upper part of Fig. 1 (the left column denotes the product id and the right column the book title). In addition, assume a transactions database, depicted in the bottom part of Fig. 1 (the left column denotes the transaction id and the right column the purchased books), which contains the books purchased by each customer so far. Assume a target (i.e., active) customer C_t, who has already purchased books A, D, and E; hence the current transaction T for C_t is $\{A, D, E\}$. Therefore, the C_t–neighborhood is identified by performing a nearest-neighbor query to find the most similar transactions to T, where, for instance, the similarity measure is simply selected to be the number of common purchased books (the similarity measures are explained in more detail in the following). For example, the similarity of T with the 5th transaction is equal to three, whereas the similarity of T with the 2nd transaction is equal to one. Thus, the C_t–neighborhood (assuming that the three most similar transactions are required) consists of the transactions with id equal to 5, 1 and 3, in decreasing order of similarity. Next, based on the identified neighborhood, a simple method to derive a top-3 recommendation can be followed [26] by finding the set of books in each transaction of the C_t–neighborhood that have not yet been purchased by C_t (i.e., not contained in T). For instance, from the 1st transaction, these items are B and F. Therefore, the complete collection of such items is $\langle B, F, C, C\rangle$. By counting the frequency of each item in the latter set [26], a recommendation can be provided to C_t, consisting of books C, B, and F, in decreasing order of frequency.[1]

[1] More complex methods can be followed as well to derive the top-N recommendation (e.g., by combining user ratings on existing products, as done in Amazon.com). Nevertheless, such issues are out of the scope of this work.

p-id	name
A	Database Principles, Programming, Performance
B	Database System Concepts
C	Database Tuning - A Principled Approach
D	Fundamentals of Database Systems
E	Principles of Database and Knowledge-Base Systems
F	Relational Database Theory

t-id	transaction
1	$\{A, B, D, F\}$
2	$\{B, E\}$
3	$\{A, C, D\}$
4	$\{C, D, F\}$
5	$\{A, C, D, E\}$

Fig. 1. A recommendation example. Top: the product database. Bottom: the transactions database

In the above example, also assume that the bookstore decides to place some particular books on sale. It is useful to identify the set of all users whose transactions have a similarity larger than a given threshold with the set of books on sale (a case of range query), so as to send email notifications to them, since it is probable that they will be interested in the sale.

Moreover, new visitors in a Web site can receive indications of links to visit or dynamic advertisements, based on the similarity of their current transaction (set of visited pages) with the past transactions in the log file [10]. In addition, considering the emerging applications of Object Relational Database Systems, transaction data can be supported by non-relational extensions (e.g., set-valued data). Therefore, similarity searching for transaction data can serve as a means for querying the database as is the case for other non-relational data types in ORDBMS, such as similarity searching for time series [1,9]. Finally, query processing techniques for similarity searching over transaction data may comprise primitives for more complex ones, such as the clustering of transaction data so as to identify groups of users that present similar behavior [29]. Although basket data have been studied for other data mining operations, such as association rules mining [2], the query processing technique of similarity searching in databases of basket data present different objectives and requirements [3]. The description of other interesting applications can be found in [3,10].

1.1 Similarity search queries

The problem of similarity searching in market basket databases requires the definition of a measure of similarity. Given two transactions T_1 and T_2, let $x = |T_1 \cap T_2|$ (i.e., the number of common items between T_1 and T_2) and $y = |T_1 - T_2| + |T_2 - T_1|$ (called the hamming distance, i.e., y is the number of items in which T_1 and T_2 differ). Then, based on [3], we consider similarity measures which are functions of x and y; thus, $Sim(T_1, T_2) = f(x, y)$. Moreover, it is assumed that $f(x, y)$ is increasing with respect to x and decreasing with respect to y. This is a reasonable assumption and is in accordance with the intuitive notion of similarity for transaction data. Evidently, a higher number of matches (x) implies greater similarity (because it indicates more common preferences), whereas a greater number of different items (y) implies less similarity (because it indicates more different preferences). An example of such a similarity measure is $f(x, y) = x/y$ [3]. More particularly, in the x/y similarity measure both x and y are considered because two transactions may have a large number of common items x, but also they may differ in a large number of items y. The dependence of the similarity measure on both of them avoids the latter problem. A large number of other functions also satisfies the above assumption, for instance $f(x, y) = \frac{1}{y}$, which is the inverse of the hamming distance, or $cos(T_1, T_2) = f(x, y) = \frac{x}{\sqrt{|T_1| \cdot |T_2|}}$, which is called cosine similarity and is used in applications such as information retrieval.

Consequently, similarity search queries in market basket databases can be stated with the following two definitions:

Definition 1 (k-Nearest-neighbor query). *Given a transactions database D and a query transaction q, find the set $N \subseteq D$ of k transactions ($1 \leq k \leq |D|$) such that for each $t \in N$ and $t' \in D - N$ it holds that $Sim(q, t) \geq Sim(q, t')$.*

Please notice that strictly k transactions most similar to the query are found, i.e., ties are resolved arbitrarily.

Definition 2 (Range search query). *Given a transactions database D, a query transaction q and a threshold r, find the set of transactions $R \subseteq D$ such that for each $t \in R$ it holds that $Sim(q, t) \geq r$.*

1.2 Survey

Although several algorithms have been proposed for similarity searching in high dimensional data [19,5], their crucial characteristic is the *dimensionality curse*, i.e., the rapid degradation in performance for increasing dimensionality. Therefore, multi-dimensional access methods are not a viable solution for the problem of similarity searching in basket databases, due to the very large dimensionality of the latter (in orders of thousands) and also due to their large sparseness (only few dimensions are supported by each transaction) [3].

The *signature-table* index method [3] is a specialized hash-based scheme, which is the first work that was proposed for the nearest-neighbor problem in basket data. Its main characteristic is that once a signature-table has been constructed, several similarity measures can been used for it, i.e., it is not restricted to only one measure. In addition, the underlying search space is not necessarily a metric space (actually, the examined cases in [3] used non-metric spaces). Experimental results in [3] considered the 1-nearest-neighbor query and presented a sufficient index selectivity, by pruning the examination of a large percentage of transactions in the database. Since the signature-table comprises the state-of-the-art in the problem of similarity searching in market basket data, it is described in more detail, along with the motivation, in Sect. 2.

Market basket databases are organized as collections of transactions, which can be represented as sets of items (in non-first normal form). In [10] the closely related problem of similarity searching in set databases is proposed. The only similarity measure considered was $Sim(T_1, T_2) = \frac{x}{|T_1 \cup T_2|}$

(normalized number of common items) and existing hash indexes were used. In general, the indexing scheme in [10] is not independent from the similarity measure, because it is based on embedding to the *hamming space*, and can only address range search queries. Moreover, the provided solution is probabilistic, i.e., the result of a range query may not always contain all the sets within the specified range. Although, in general, sets can represent transactions of basket data, specialized characteristics of the latter are ignored with this representation. For instance, transactions of basket data are highly skewed and correlated (containing patterns) and their lengths are clustered around a small mean value.

The inverted indexes and the signature files [4] are two indexing schemes for set data that have been proposed for the problem of *partial match retrieval*. For the problem of similarity searching in basket data, the inverted index can efficiently handle only the restricted case of nearest-neighbor queries based on the number of common items (i.e., $f(x, y) = x$) and it cannot support general similarity measures (because it has to retrieve all transactions containing *any* item in the query transaction [3]) or the range search query [3]. Experimental results in [3] illustrate the deficiency of the inverted index. Signatures are approximate representations of sets and have been used for the purpose of superset/subset retrieval [18]. However, signatures, as they are presented in existing work, cannot guarantee exact solutions to the problem of similarity searching in basket data (see Sect. 3).

1.3 Paper contribution

In this paper we consider the problem of similarity searching in market basket databases. We are interested in complete, and not approximate, results. We present a new method, $\mathrm{S}^3\mathrm{B}$ (denoting Signature-based Similarity Search for Basket-data), which is based on the representation of transactions with signatures. Signatures comprise a compact and effective coding that captures the similarity between transactions. However, existing signature creation methods do not guarantee the correct finding of all transactions that satisfy a similarity query. We propose a signature representation method which is proved to find the correct results.

On the basis of index structures for signatures, we propose novel algorithms for processing similarity queries. Moreover, we develop enhancements that achieve significant performance improvements, compared to existing approaches. Extensive experimental results, which consider a variety of factors, provide evidence of the superiority of $\mathrm{S}^3\mathrm{B}$.

The rest of this paper is organized as follows. Section 2 briefly describes the main characteristics of the signature-table, and presents the motivation in our work. The proposed approach is given in Sect. 3. Section 4 contains the proposed enhancements and Sect. 5 gives the experimental results. Finally, Sect. 6 provides the conclusions.

2 Motivation

2.1 Signature-table

Given a database, D, of transactions with items from a domain I, the method in [3] first partitions I in K subsets, each denoted

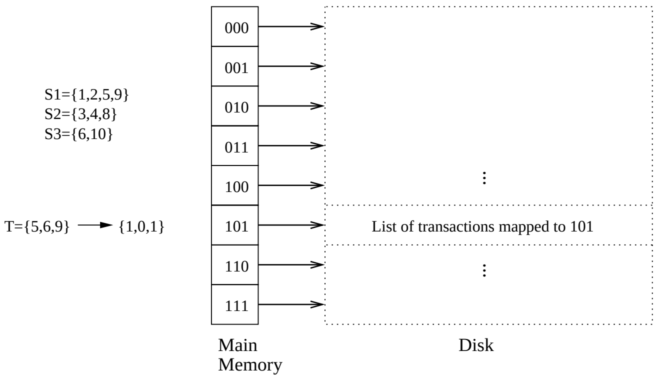

Fig. 2. Example of the signature-table

as S_i, $1 \le i \le K$ (K is called *signature cardinality*). The partitioning is performed with a variant of the single-linkage clustering algorithm [27], which determines K clusters among the items of I. In [3], it is assumed that the database is static and the transactions are known beforehand. The distance between two items, i_1 and i_2, used for the clustering is defined as $\mathrm{dist}(i_1, i_2) = \frac{1}{\mathrm{support}(i_1,i_2)}$, where support$(i_1, i_2)$ denotes the normalized frequency of itemset $\{i_1, i_2\}$ [2]. Thus, correlated items are close (with respect to this distance measure) and are clustered in the same partition. Then, given a user-defined parameter, a (called *activation threshold* – denoted as r in [3]), each transaction T is mapped to a bitmap $\{b_1, \ldots, b_K\}$, called *supercoordinate*, where $b_i = 1$ if $|T \cap S_i| \ge a$, otherwise $b_i = 0$. By getting a table with 2^K entries, one per each possible supercoordinate, each transaction is hashed to the entry of the table that is indicated by its supercoordinate.

An example is depicted in Fig. 2, where $K = 3$ and $a = 1$. Each S_i partition ($1 \le i \le 3$) is also depicted. Transaction $T = \{5, 6, 9\}$ is mapped to supercoordinate $\{1, 0, 1\}$ (because T has two common items with S_1, zero common items with S_2, one common item with S_3, and $a = 1$), thus it is stored in the corresponding list. As described in [3], the table with the supercoordinates is maintained in main memory, whereas the actual transactions, which are indexed by each entry of the signature-table, are stored on disk.

Given a query transaction Q, the nearest-neighbor query operates as a branch-and-bound searching method. For each supercoordinate SC_i ($1 \le i \le 2^K$) of the signature-table, the optimistic (i.e., maximum) value of similarity $Sim_o(Q, SC_i)$ is calculated, called optimistic bound[2]. The optimistic bound is calculated by determining the optimal values x', y' for x and y, respectively (see Sect. 1.1). Let $B = \{b_1, \ldots b_K\}$ be a supercoordinate of the signature-table and $a_i = |Q \cap S_i|$ (the number of common items between Q and the S_i partition of the domain I). Then it holds that [3] ($\overline{b_j}$ denotes the inverse of b_j):

$$x' = \sum_{j=1}^{K} \overline{b_j} \cdot \min\{a-1, a_j\} + \sum_{j=1}^{K} b_j \cdot a_j \quad (1)$$

$$y' = \sum_{j=1}^{K} \overline{b_j} \cdot \max\{0, a_j - a + 1\} + \sum_{j=1}^{K} b_j \cdot \max\{0, a - a_j\} \quad (2)$$

[2] It holds that $Sim_o(Q, SC_i) \ge Sim(Q, SC_i)$, thus no false dismissals are presented [3].

For the previously illustrated example, let a query $Q = \{2, 6, 10\}$. Then, for the supercoordinate $\{1, 0, 1\}$, it is $x' = 3$ and $y' = 0$.

The nearest-neighbor query proceeds by sorting the supercoordinates of the signature-table in decreasing order of the optimistic bound and by testing them sequentially. If a supercoordinate has an optimistic bound larger than the similarity of the currently most similar transaction to Q (assuming the 1-nearest-neighbor query), the transactions indexed by it are retrieved and are examined against Q, and the result is updated. Otherwise, the search to the transactions indexed by the supercoordinate can be pruned.

2.2 Motivating factors

Although [3] addressed the problem of similarity searching for basket data for the first time, it presents limitations which have a significant impact when considering large databases of basket data. These limitations involve several factors that are summarized in the following and comprise the motivation of this work.

Query execution time: in [3], the performance of the signature-table was examined only with respect to its selectivity (percentage of pruned transactions). Regardless of the selectivity of an indexing scheme, the arrangement of the examined transactions (those that are not pruned) is crucial for the I/O overhead, since it may result in a large number of pages that are fetched from secondary storage (page-scattering effect). Furthermore, for each query, the signature-table has to be sorted with respect to the optimistic bounds, and its similarity to all supercoordinates is calculated. Since the size of the signature table is 2^K, which grows rapidly for large values of K, the CPU time can be significantly affected. Nevertheless, the query execution time, which is affected by both the aforementioned factors, was not examined in [3]. Therefore, what is required is an indexing scheme that achieves both high selectivity and low query execution times.

Dynamic data: the signature-table is an index for static data. It requires the knowledge of all transactions beforehand to calculate the support of every 2-itemset and to derive the partitioning of the I domain (domain of items). However, for dynamic data, this restriction can be alleviated if the partitioning found from the portion of existing data can be used also for the new transactions, i.e., after new insertions[3]. Nevertheless, dynamic data require an efficient representation of the transactions on disk (lists of transactions for each supercoordinate), but [3] does not elaborate further on this issue that impacts on the efficiency of the method.

Index design issues: the design of the signature-table is based on a two-level indexing scheme. The supercoordinates comprise the upper part and the transactions are stored at the data level. However, related work on similarity search for multi-dimensional data indicates that a branch-and-bound search procedure (like the one used also in [3]) can better exploit multi-level indexes in order to reduce the examined portion of the database [23].

Characterizing efficiency: the experimental results in [3] were constrained to the 1-nearest-neighbor query. The range query (Definition 2) is briefly mentioned in [3], but no results are presented for it. A clear performance characterization should consider both the k-nearest-neighbor query ($k \geq 1$) and the range query.

[3] Market Basket Data usually are historic, therefore deletions are rare.

3 S^{3}B: signature-based similarity search for basket-data

The proposed method, denoted as *Signature-based Similarity Search for Basket-data* (S^{3}B), consists of two main components: a) the method for the representation of transaction data; b) an indexing scheme along with the algorithms for similarity searching, for the organization of representations. These components are analyzed in the following.

3.1 Representation method

Let U be the universal set (the domain of all items). Then, each transaction is a vector in the $|U|$-dimensional space, where each dimension is either 0 or 1. $|U|$ is significantly large (in the order of thousands of items) and the U-dimensional space is sparse. Thus, the direct representation of transactions in the $|U|$-dimensional space is inefficient. We derive a more compact representation, based on coding the transactions, with signatures.

Assuming bit vectors with F positions ($F << |U|$), each item is mapped to one of the positions by using a hash function. A transaction T can be represented by such a bit vector of length F by hashing all its items. This bit vector is called the signature of T and is denoted as $S(T)$. For instance, $U = \{1, \ldots 1000\}$, $F = 100$ and each item $i \in U$ is hashed to the $H(i) = i \; mod \; F$ position. Then, for a transaction $T = \{128, 235, 728, 987\}$, $S(T)$ has the bits at positions 28, 35, and 87 set to 1, and all remaining bits are set to 0 (a similar signature creation procedure, where each item corresponds to one bit in the signature, has been used in [15].) Therefore, for each $i \in T$, the bit in $H(i)$ position of $S(T)$ is set to 1. For convenience, $S_p(T)$ denotes the bit at pth position of $S(T)$, $0 \leq p \leq F - 1$ (thus, $S_p(T)$ can be either 0 or 1).

Signatures have been used in text databases [4] and databases with set-valued attributes [18] for the partial match problem, e.g., find all transactions that contain a specified set of items. However, similarity search queries (Definitions 1 and 2) present new requirements. For the example of the previous paragraph, let a query transaction $Q = \{128, 329, 728, 829\}$. Then, following the same hash function as in the example, bits at positions 28 and 29 are set. The number of common items between Q and T is two, thus $x = 2$. In addition, Q and T differ in four items, thus $y = 4$. Therefore, assuming the $f(x, y) = x/y$ similarity measure, $Sim(T, Q) = 2/4 = 0.5$. However, using the signature representations instead, $S(T)$ and $S(Q)$ have one bit set at common position (position 28), thus $x = 1$, and they have in total three bits set at different positions (positions 35, 87, and 29), thus $y = 3$. Therefore, $Sim(S(T), S(Q)) = 1/3 = 0.33$. Assume, for instance, a range query with similarity threshold $r = 0.45$ or a nearest-neighbor query where the similarity from the current nearest transaction is 0.4. By using the signature representations, $S(T)$ will be rejected, although $Sim(T, Q) = 0.5$. Thus, T will not be included in the result although its actual similarity fulfills the query. Evidently, false-dismissals are produced. The reason is the existence of collisions during the hashing of items into signatures. For instance, items 128 and 728 of T and Q are hashed in the same position, which contributes one match instead of two.

Consequently, existing representation methods and indexes that use signatures cannot address the similarity queries for basket data. Moreover, for multi-level signature indexes [24,7], each transaction T is represented at each level by the signature of the entry which is the sub-root of the portion of the tree that contains T. Assuming that leaves are at level 0 and contain the signatures generated by the transactions, the signature of a node's entry at level $l > 0$ is derived from the superimposition (i.e., OR-ing of bits at each position) of all signatures of the subtree rooted at the entry. Therefore, even if a hashing function guarantees the avoidance of collisions for the signatures at the leaf level, the signatures at the upper levels of the multi-level index are possible to present collisions at several positions, due to the superimposition. In the following, for a transaction T, $S^l(T)$ denotes the signature in the root of the subtree at level l, which contains T (for simplicity we denote $S^0(T) \equiv S(T)$).

To overcome the problem, we propose the following approach. Given a query signature Q, we determine optimistic bounds for x and y so as not to produce false-dismissals. For the purpose of generality we consider a multi-level index (see Sect. 3.2).

Definition 3 (Optimistic bounds). *Let a query transaction $Q = \{q_1, \ldots, q_n\}$, and a vector C of length F, where $C = \{c_1, \ldots, c_F\}$ and $c_p = |\{q_j | q_j \in Q, H(q_j) = p\}|$, $0 \le p \le F - 1$. For the signature $S^l(T)$ of a transaction T in the index ($l \ge 0$), the optimistic bounds x' and y' for x and y, respectively, are given as follows:*

$$x' = \sum_{p=0}^{F-1} c_p \cdot S_p(Q) \cdot S_p^l(T) \tag{3}$$

$$y' = \sum_{p=0}^{F-1} c_p \cdot S_p(Q) \cdot \overline{S_p^l(T)} \tag{4}$$

We denote OptSim$(Q, S^l(T)) = f(x', y')$.

The formal proof that the optimistic bounds of Definition 3 do not produce false-dismissals is given in the following. The basic intuition in this proof is the consideration (using the C vector) of all items of the query transaction that are hashed at the same position, so as (based also on the properties of super-imposition) to have $x' \ge x$ and $y' \le y$. For the simple example examined earlier (where $T = \{128, 235, 728, 987\}$, $Q = \{128, 329, 728, 829\}$ and $l = 0$), we have that $x' = 2$, because items 128 and 728 increase x' by two (not by one, as in the case of the direct use of $S(Q)$ and $S(T)$). Respectively, $y' = 4$, because items 329 and 829 increase y' by two (not by one, as in the case of the direct use of $S(Q)$ and $S(T)$). The calculated similarity is $x'/y' = 2/4 = 0.5$, which is equal to the actual similarity between Q and T. Hence, in this example, T will not be omitted from the result and no false-dismissal is presented.

Lemma 1. *Given a transaction T in the index, and a query transaction Q, where $x = |T \cap Q|$, the bound x' of Definition 3 is optimistic, i.e., $x' \ge x$ for each $S^l(T), l \ge 0$.*

Proof. For the items of Q and T that are hashed at position p (using a function $H(\cdot)$), let x_p denote the number of those who are common between Q and T. Evidently, $x = \sum_{p=0}^{F-1} x_p$. It is assumed that the level l is considered, where $l \ge 0$. From Eq. 3, we denote $x'_p = c_p \cdot S_p(Q) \cdot S_p^l(T)$, where c_p is the number of items of Q that are hashed at position p in $S(Q)$ (see Definition 3). Therefore, $x' = \sum_{p=0}^{F-1} x'_p$.

For each position p, if $S_p(Q) = 0$ or $S_p^l(T) = 0$, then $x_p = 0$, because Q or T, respectively, contain no items that are hashed at position p (if $l > 0$, this is preserved for $S_p^l(T)$ due to superimposition). From the definition of x'_p it follows that $x'_p = x_p = 0$.

If $S_p(Q) = S_p^l(T) = 1$, then $x_p \le c_p$ (if more than c_p items of T are hashed in p, then only the c_p items of Q can be common between Q and T). From the definition of x'_p it follows that $x'_p = c_p$, hence $x'_p \ge x_p$.

From all above cases, it follows that $\forall\ p$ it is $x'_p \ge x_p \Rightarrow \sum_{p=0}^{F-1} x'_p \ge \sum_{p=0}^{F-1} x_p \Rightarrow x' \ge x$.

Lemma 2. *Given a transaction T in the index, and a query transaction Q, where $y = |T - Q| + |Q - T|$, the bound y' of Definition 3 is optimistic, i.e., $y' \le y$ for each $S^l(T), l \ge 0$.*

Proof. As previously, for all items of Q and T that are hashed at position p, let y_p denote the number of those in which Q and T differ (if i is such an item, then either $i \in Q \wedge i \notin T$, or, $i \notin Q \wedge i \in T$). Evidently, $y = \sum_{p=0}^{F-1} y_p$. It is assumed that the level l ($l \ge 0$) is considered. From Eq. 3, we denote $y'_p = c_p \cdot S_p(Q) \cdot \overline{S_p^l(T)}$. Therefore, $y' = \sum_{p=0}^{F-1} y'_p$.

For each position p, if $S_p(Q) = 0$, then Q does not contain any items hashed at p. In this case, if $S_p^l(T) = 0$, then T also does not contain any items hashed at position p (for $l > 0$ this holds due to superimposition). Therefore, $y_p = 0$. From the definition of y'_p it follows $y'_p = 0$, hence $y'_p = y_p$. However, if $S_p^l(T) = 1$, then $y_p \ge 0$, because each item of T that is hashed at position p is not present in Q ($S_p^l(T) = 1$ may hold due to superimposition, i.e., when $S_p^0(T) = 1$), and this is why $y_p \ge 0$ and not $y_p > 0$). From the definition of y'_p it follows that $y'_p = 0$. Hence, from the above cases, $y'_p \le y_p$.

If $S_p(Q) = 1$ and if also $S_p^l(T) = 1$, then Q and T may have common items that are hashed at position p. If all items hashed at position p are common between Q and T, then $y_p = 0$. Otherwise, $y_p > 0$. From the definition of y'_p, in all these cases, $y'_p = 0$. Consequently, $y'_p \le y_p$.

If $S_p(Q) = 1$ and $S_p^l(T) = 0$, the items of Q that are hashed at position p are not present in T, since T does not contain any item hashed at p (for $l > 0$ this holds due to superimposition). Therefore, $y_p = c_p$ (c_p is the number of such items of Q). From the definition of y'_p it follows that $y'_p = c_p$, hence $y'_p = y_p$.[4]

From all above cases, it follows that $\forall\ p$ it is $y'_p \leq y_p \Rightarrow \sum_{p=0}^{F-1} y'_p \leq \sum_{p=0}^{F-1} y_p \Rightarrow y' \leq y$.

Proposition 1. *The optimistic bounds derived by Eqs. 3 and 4 do not produce false-dismissals.*

Proof. It has to be shown that $OptSim(Q, S^l(T)) \geq Sim(Q,T)$, for each $S^l(T)$ ($l \geq 0$), where T is a transaction in the index and Q is a query transaction. It holds that $OptSim(Q, S^l(T)) = f(x', y')$ (from Definition 3) and that $Sim(Q,T) = f(x,y)$. Since $x' \geq x$ and $y' \leq y$ (from Lemma 1 and 2), it follows that $f(x', y') \geq f(x,y)$, because $f(\cdot,\cdot)$ is increasing in terms of the first argument and decreasing in terms of the second. Therefore, the required inequality holds.

Conclusively, the derived optimistic bounds guarantee that all transactions which satisfy a similarity query, are included in the result. It is worth noticing that other signature generation schemes, from the domain of information retrieval and text databases, which represent each item with more than one bits, cannot present the aforementioned properties so as to guarantee no false-dismissals.

3.2 Indexing method

Several single-level indexes for signature data have been proposed [4,18], which act as filters for the partial retrieval problem. Multi-level indexes for signature data have also been proposed [24,7]. The RD-tree [14] is a generalized tree for set data. One of the options in [14] is to represent the contents of the nodes with signatures. The resulting scheme is equivalent to the S-tree [7].

The S-tree is a height balanced tree. It resembles the paradigm of the R-tree [13] for spatial data. Each node contains a number of entries, each of them being a pair that consists of a signature and a pointer to the child node. The root node can accommodate at least two and at most M entries, whereas the remaining nodes can contain at least m and at most M entries. Signatures at the upper levels (i.e., internal nodes) are formed with superimposition (OR-ing). An example of an S-tree with three index levels is depicted in Fig. 3. In this example, $F = 8$ (the length of the signatures), whereas the universal set is $U = \{1,\ldots,100\}$. The hashing function is $H(x) = x \bmod F$, whereas M (max fanout) is 4 and m (min fanout) is 2. Transactions are depicted at the data level.

The S-tree is an index for dynamic data. Insertions are performed by traversing the tree in a top-down manner and selecting the leaf node N that its signature will require the minimum *weight increase*, denoted as $\epsilon(N)$, to accommodate the new entry. Let s be the signature of N (resulting from the superimposition of N's contents) and s' the signature of the new entry; then $\epsilon(N) = \gamma(s \vee s') - \gamma(s)$, where $\gamma(s)$ denotes the weight of s, i.e., the number of set bits (equal to one) in s. For instance, let $s = \{1,0,1,0\}$ and $s' = \{1,1,0,0\}$, thus $\epsilon(N) = 1$, due to the second bit that is set in s' but not in s. After the insertion in the selected leaf node, the parent-node signature may need to be updated, so as to include the new one (for each set bit in the signature of an entry in a child node, the signature of the parent node has the corresponding bit also set; see Fig. 3). In the case that the selected leaf node N contains the maximum number of entries M, then it has to be split. A new node is created and the contents of N along with the new entry are distributed between N and the newly created node (more details on the split procedure are given in Sect. 4.1). After the split, the signature of the newly created node (i.e., the superimposed signature of all signatures contained in the new node) is inserted in the parent node. Since the latter may also contain M entries, the split may propagate up to the root node. In case the root node splits, a new root is created and the height of the tree is increased by one.

The performance of query processing is affected by the number of examined index nodes, since this factor impacts on both the I/O and CPU time overheads (the fetching of nodes from secondary storage and the processing of their contents). Therefore, in a way that is analogous to the paradigm of the R-tree [13], the following optimization criterion can be stated: *The weight of each node signature should be minimized* (i.e., the minimization of the number of set bits in each node signature). As described earlier, a signature of a node N (which is stored in N's parent node) acts as "bounding signature" for the signatures stored in N and all its descendants. Thus, the minimization of N's weight aims at the tighter enclosure of bounded signatures (this is similar to the minimization of MBR area in the R-tree, which reduces the dead space). Evidently, the weight-minimization criterion tries to reduce the probability that each node will be examined, since a less tight enclosure of bounded signatures will result in a more frequent node activation.

Clearly, the aforementioned optimization criterion pursues the improved clustering of signatures with the objective of good query performance. This targets several types of queries, such as subset- and superset-containment queries [7,28], range and k-nearest-neighbor similarity queries (the algorithmic description of similarity queries is given in the following section). The achieved signature clustering is independent from the query type and moreover, for similarity queries, independent from the used similarity measure (it has to be considered that in the analogous paradigm of the R-tree, the clustering of spatial objects is also achieved independently from the query type; i.e., the R-tree can serve as a versatile index for several types of queries such as the range, k-nearest-neighbor, spatial join, etc.). This is in accordance with what is desired for the case of transactions databases, as also stated in [3], because several types of queries or similarity measures can be served by the index[5].

[4] It should be noticed that if y'_p was considered, as previously, equal to 0, then the required inequality would still hold. However, this would affect the tightness of the bound and the number of false-drops, which affect the performance of the search procedure.

[5] Please notice that the weight increase measure ϵ is intuitively related to similarity measures of the form $f(x,y)$, since ϵ considers both the common and the different bits among the signatures. How-

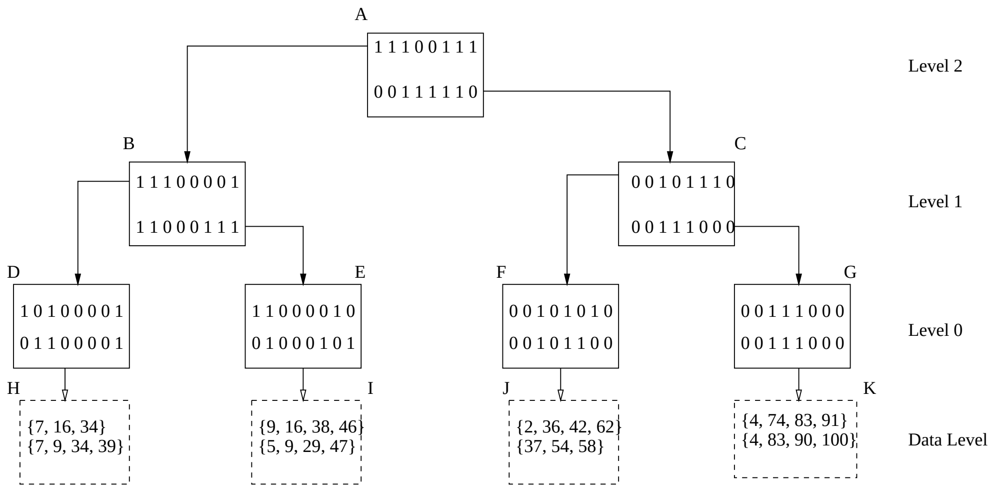

Fig. 3. Example of an S-tree

The fulfilment of the weight-minimization criterion is considered during the insertion procedure, where, as described previously, the leaf selection for the accommodation of the new entry is done with the examination of the weight increase ϵ. Additionally, the criterion is also considered by the split procedure, during the distribution of entries between the two resulting nodes, by assigning each entry e to the node that will present the largest difference in weight increase for the accommodation of e [7,28] (see also Sect. 4.1). In this work, as an initial base, we follow the approach of [28] for organizing collections of transaction in a multi-level signature indexing scheme. Nevertheless, we examine enhancements for the split procedure and the node storage scheme, which are presented in Sect. 4 and aim to the further consideration of the weight-minimization criterion.

3.3 Processing of similarity queries

3.3.1 k-Nearest-neighbor query

The algorithm for the k-nearest-neighbor query is based on an index traversal. By pruning paths according to the *OptSim*$(\cdot,\cdot)$ measure (Definition 3), the traversal reaches the leaf nodes. The transactions indexed by the leaf node's entries determine the actual similarity.

For simplicity reasons, only the case of 1-nearest-neighbor is given (the k-nearest-neighbor algorithm merely has to keep track of all the k-nearest-neighbors during the searching, instead of only the first one). The algorithm is depicted in Fig. 4. It is based on the depth-first search of the index and is analogous to the paradigm of [23] (the paradigm of [16] may also be followed; we have developed the analog of the search procedure of [16], but it did not show a significant performance difference, mainly because the nodes close to the root level produce over-optimistic similarities which affect the breadth-first search procedure of [16] – see Sect. 4.2). However, the only possible pruning criterion is the one of Definition 3 (optimistic bound). Hence, pruning can be applied upwards only (step 4). Initially, the *currNN* variable (that at the end will contain the nearest-neighbor transaction) is arbitrarily set to one of the transactions in the database (e.g., the first one) and function *DFS-NN* is invoked for the root node. The sorting of node entries (step 3) is done in decreasing order of the optimistic bound, because in such a way paths that are more promising to contain the result transactions are searched first (as a tie-breaking criterion, different similarity measures can be used; for instance, for the x/y measure, ties can be resolved with the inverse of hamming distance). The testing of the actual similarity value with the transactions is done at the data level (steps 8-10) by retrieving the corresponding data pages (recall that *Sim*$(\cdot,\cdot)$ calculates the actual similarity between two transactions, not between their signatures).

ever, no strict relation holds between them, because the optimization of $f(x,y)$ requires both the maximization of x and the minimization of y. In contrast, the minimization of ϵ (weight-minimization criterion) may conflict with the maximization of x or minimization of y, because the ordering of the two corresponding signatures is taken into account in the calculation of ϵ (see the definition of ϵ which was given above). For this reason, $f(x,y)$ cannot be directly considered for the weight-minimization criterion.

```
Procedure DFS-NN($Q$, Node, currNN)
begin
1.  foreach entry $e_i \in$ Node
2.      calculate OptSim$(Q, e_i)$
3.  sort $\{e_i | e_i \in$ Node$\}$ in decreasing OptSim
4.  foreach $e_i \in Node$ and OptSim$(Q, e_i) >$ Sim(Q, currNN) {
5.      if Node not leaf
6.          DFS-k-NN($Q$, $e_i$.child, currNN)
7.      else {
8.          $T_i = e_i$.transaction
9.          if Sim(Q, $T_i$) > Sim(Q, currNN)
10.             currNN $= T_i$
11.     }
12. }
end
```

Fig. 4. The algorithm for the nearest-neighbor query

Example. Let $Q = \{4, 74, 83, 87, 91\}$ be a query transaction for the index depicted in the example of Fig. 3. We use the $f(x,y) = x/y$ similarity measure. Then (using the same hash function as in that example), $S(Q) = \{0,0,1,1,1,0,0,1\}$ and $C = \{0,0,1,2,1,0,0,1\}$ (i.e, at position 3 of $S(Q)$ two items

```
Procedure Range($Q$, Node, $r$)
begin
1.  foreach entry $e_i \in$ Node
2.      calculate $OptSim(Q, e_i)$
3.  foreach $e_i \in Node$ and $OptSim(Q, e_i) > r$ {
4.      if Node not leaf
5.          Range($Q$, $e_i$.child, $r$)
6.      else {
7.          $T_i = e_i$.transaction
8.          if Sim($Q$, $T_i$) $> r$
9.              output $T_i$
10.     }
11. }
end
```

Fig. 5. The algorithm for the range query

are hashed, whereas at positions 2, 4, and 7 one item is hashed). DFS-NN starts with node A (root node), where *currNN* is initially set equal to the first transaction, i.e., $\{7, 16, 34\}$ (the similarity between Q and this transaction is 0). The optimal bound for the two entries of A is calculated. The optimal bound between Q and $\{1, 1, 1, 0, 0, 1, 1, 1\}$ is $\frac{x'}{y'} = \frac{2}{3}$, whereas the optimal bound between Q and $\{0, 0, 1, 1, 1, 1, 1, 0\}$ is $\frac{x'}{y'} = \frac{4}{1}$. Thus, node C is visited first. The optimal bound between Q and $\{0, 0, 1, 0, 1, 1, 1, 0\}$ is $\frac{2}{3}$, whereas the optimal bound between Q and $\{0, 0, 1, 1, 1, 0, 0, 0\}$ is $\frac{4}{1}$. According to this, node G is visited next, and the optimal bound for both its entries is similarly calculated as 4. Finally, the transactions in data node K are retrieved and $Sim(Q, \{4, 74, 83, 91\}) = 4$, whereas $Sim(Q, \{4, 83, 90, 100\}) = \frac{2}{5}$. Therefore, *currNN* is set to $\{4, 74, 83, 91\}$. Since the optimal bound of all remaining node entries (that were examined during the DF-search) is lower than the similarity to *currNN*, the procedure terminates, having determined the nearest-neighbor transaction.

3.3.2 Range query

In [3] no algorithm is provided for the range query. We present an algorithm, which, similarly to the one for the nearest-neighbor query, traverses the index and prunes the visiting to subtrees according to the *OptSim* criterion. The algorithm is depicted in Fig. 5.

3.4 Qualitative comparison

Having described the basic scheme of both S^{3}B and signature-table, herein we briefly summarize the rationale in the development of the proposed method and give a qualitative comparison among the two schemes. The presented issues address the earlier described motivation.

The signature-table maintains *every* possible supercoordinate independently from the data, therefore supercoordinates are not derived from the contents of the indexed dataset. Their length (i.e., signature cardinality K) is restricted, because it causes an exponential increase to the size of the signature-table[6]. This can impact on the number of false-drops and to cause an I/O overhead. The reason is that a smaller number of partitions results into more items within each one, thus to an increased probability of conflict. In contrast, S^{3}B uses signatures derived from the data, i.e., each signature is formed from the corresponding transaction. Thus, the signatures can have much larger sizes (hundreds of bits), compared to supercoordinates.

Moreover, the clustering of signatures in the signature-table is achieved through a variation of the single-link algorithm that is applied on the items (see Sect. 2.1). This clustering approach, however, may present the drawback of the single-linkage effect, i.e., the incorrect merging of clusters (see [12] for a description of the problem in the case of multi-dimensional points). In the case of items (in basket data), the single-linkage effect may incorrectly cause the creation of partitions comprising long "chains" of items that have only pairwise correlation. In contrast, S^{3}B takes a different approach for the clustering of signatures, by capitalizing on the weight-minimization criterion, to achieve the activation of as few tree nodes as possible during query execution.

S^{3}B uses a multi-level index, in contrast to the signature-table, which is a single-level index. Thus, S^{3}B can better exploit the branch-and-bound technique so as to increase the selectivity during query processing. Additionally, the multi-level signature index does not involve the costly operation of sorting for the entire index, as in the case of signature-table. The only required CPU cost involves the comparison of the signatures in the index against the query one and the sorting of the former according to the calculated similarity. However, this sorting is a local operation (i.e., within nodes) and is performed only for the tree nodes that are activated by the query.

Finally, S^{3}B can handle dynamic data. The multi-level signature index is a dynamic data structure, whereas the signature-table requires the knowledge of the transaction database beforehand to derive the partitioning.

4 Enhancements for large databases

In this section we present a twofold enhancement for the index used by S^{3}B for the organization of signature representations. It consists of an improved split procedure and a paging scheme, which significantly reduce the query execution times in large transactions databases.

4.1 Split method

The split method for the case of signature data can be stated as follows. Given a node N with M signature entries (M is the maximum allowed number of entries) and a new entry e that has to be inserted in N, the method will produce two new nodes N_1 and N_2 ($|N_1| + |N_2| = M + 1$) and consists of two steps:

A. The initial selection of two seeds, i.e., two entries among $N \cup e$ that will be the first entries of N_1 and N_2, respectively.
B. The iterative selection of the remaining entries, which will be inserted in N_1 and N_2.

As described in [28], the selection of a split method can largely affect the performance of the index, in terms of query

[6] The value of K that is used in [3], is equal to 15.

```
Procedure Split(N, e)
begin
1.  MinMaxWeight $\leftarrow \infty$, $N_1^s \leftarrow \emptyset$, $N_2^s \leftarrow \emptyset$
2.  foreach pair of entries $e_i, e_j \in N \cup \{e\}$ {
3.      $N_1 \leftarrow e_i$, $N_2 \leftarrow e_j$
4.      $R = (N \cup e) - \{e_i, e_j\}$
5.      while $R \neq \emptyset$ {
6.          foreach $e_k \in R$ {
7.              $\epsilon(N_1, e_k) = \gamma(S(N_1 \vee e_k)) - \gamma(S(N_1))$
8.              $\epsilon(N_2, e_k) = \gamma(S(N_2 \vee e_k)) - \gamma(S(N_2))$
9.              $\delta(e_k) = |\epsilon(N_1, e_k) - \epsilon(N_2, e_k)|$
10.         }
11.         select $e_m = \min_{\forall e_k \in R}\{\delta(e_k)\}$
12.         if $\epsilon(N_1, e_m) < \epsilon(N_2, e_m)$
13.             $N_1 \leftarrow N_1 \vee e_m$
14.         else
15.             $N_2 \leftarrow N_2 \vee e_m$
16.         $R \leftarrow R - \{e_m\}$
17.     }
18.     $w = \max\{\gamma(S(N_1)), \gamma(S(N_2))\}$
19.     if $w <$ MinMaxWeight {
20.         MinMaxWeight $\leftarrow w$
21.         $N_1^s \leftarrow N_1$
22.         $N_2^s \leftarrow N_2$
23.     }
24. }
25. return $N_1^s, N_2^s$
end
```

Fig. 6. The proposed split algorithm

execution times. The instantiations of Steps A and B in the above abstract split scheme determine both the complexity of the split method and its efficiency during query execution. A method with a larger complexity can clearly pay-off by presenting improved query processing performance. In [28], several split methods are proposed, which outperform previous ones (such as the method of [7]) for the case of inclusion queries. Nevertheless, due to the scalability requirement in the case of processing similarity queries in large databases of basket data, we present a new split algorithm, which is depicted in Fig. 6.

In this split algorithm, every pair (step 2) of entries is tested as possible seeds (for the Step A of the general split scheme described previously). The two seeds comprise the initial entries of the new nodes N_1 and N_2 (step 3). Then, from the set of remaining ones, denoted as R (step 4), entries are assigned to N_1 and N_2. For each entry e_k in R (step 6), the corresponding weight increases of the signatures of the two nodes (denoted as $\epsilon(N_1, e_k)$ and $\epsilon(N_2, e_k)$, respectively) are calculated (steps 7-8). In addition, for e_k, the difference in weight increase, i.e., the absolute difference value between $\epsilon(N_1, e_k)$ and $\epsilon(N_1, e_k)$, is assigned to $\delta(e_k)$ (step 9). The entry e_m of R with the largest difference in weight increase $\delta(e_m)$, is selected (step 11), is assigned to the corresponding node (steps 12-15) and is removed from R (step 16). When all entries have been assigned to N_1 and N_2, the algorithm examines the maximum weight, w, among $\gamma(S(N_1))$ and $\gamma((N_2))$ (step 18). If w is less than the minimum such weight examined so far (denoted as *MinMaxWeight*), then nodes N_1 and N_2 are selected as the best pair (steps 21-22). Finally, the algorithm returns the selected best pair of nodes (denoted as N_1^s and N_2^s), thus the split is performed.

It can be easily shown that the complexity of the proposed split algorithm is $O(M^4)$, where M is the maximum number of entries in a node. Since M is constrained to take small values (see due to the paging scheme of Sect. 4.2), it can be safely guaranteed that the overall overhead for performing the split is not significant.

It has to be mentioned that in [7,28] less costly split algorithms have been proposed; for instance, split algorithms of linear or quadratic complexity, which use simple heuristics (with linear complexity) for the instantiation of Step A of the abstract split scheme (given in the beginning of this section). However, poor seed selection, due to the simple heuristic, can significantly impact on the split result, since nodes with large weights may occur. In addition, existing split algorithms of linear and cubic complexity [7,28] perform a simple heuristic (with linear complexity) for the instantiation of Step B (i.e., assignment of the remaining entries). Similar to the previous case, a poor assignment, due to the simple heuristic, impacts on the split result, since nodes with large weights may occur. In contrast, in the proposed algorithm, the instantiations of both A and B Steps use heuristics of quadratic complexity (differently from the simple heuristics of linear complexity, which are used in existing algorithms) so as to consider in more detail the weight-minimization criterion (described in Sect. 3.2). For Step A, each possible pair of seeds is examined (at steps 19–22), whereas for Step B, the assignment of each entry is done through the examination of each one, each time, and the selection of the one that incurs the smallest weight-increase. Finally, the selection of the resulting two nodes is performed with respect to the minimization of the maximum weight (through the MinMax value that is used in the algorithm) between the two resulting nodes[7]. The performance gains due to the proposed split algorithm are examined experimentally in Sect. 5.3.5, in terms of query execution time.

4.2 Paging scheme

An important limitation of existing multi-level indexes for signature data is that nodes which are close to the root level tend to have reduced selectivity [7]. This impacts on the query execution time, especially in the case of very large databases. The reason for the aforementioned problem is the accumulation of a large number of set bits (i.e., bits equal to ‘1’) in the signatures of the upper-level nodes, due to the superimposition of the signatures from the lower levels. Hence, in existing indexes, the weight-minimization criterion is not effectively achieved at the nodes residing at levels close the root level.

To avoid this , the node size of the index should be kept small, relative to the capacity of physical pages. More precisely, let a node N and $S(N)$ its signature, which is stored at N’s father node. The bit at the ith position in $S(N)$ is the result of the logical OR between the bits at ith positions in the signatures of all N’s entries. Therefore, the probability that the ith bit is set in $S(N)$, is proportional to the number of N’s

[7] Other possible heuristics have also been examined, e.g., the minimization of the sum of the weights of the two nodes; however, this did not give better results.

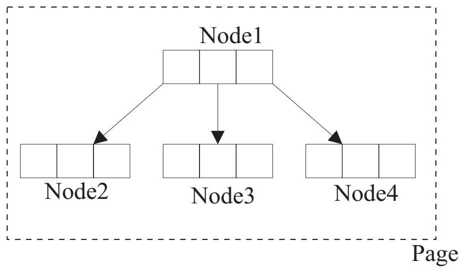

Fig. 7. Example of paging for *Nfr* $= 4$

entries, which is equal to the size of N. Since the logical ORing in $S(N)$ is done independently for each bit position, the expected weight of $S(N)$ (i.e., the expected total number of set bits in $S(N)$) is the sum of the above probability for every bit position. Therefore, the expected weight is proportional to N's size [considering that the total number of bit positions in $S(N)$ is equal to F, which is a constant (Sect. 3.1)]. Hence, the expected node weights, especially for the nodes close to the root level, can be reduced by keeping the node sizes small.

However, the use of small node sizes results in a waste of space, because a whole physical page is reserved to store just a small node. For this reason, the storing of nodes has to be done according to a scheme that maps the contents of several nodes to physical pages. For this purpose, up to *Nfr* (stands for node factor) nodes are stored within each physical page. An example is illustrated in Fig. 7. There is a clear distinction between the logical storage scheme, which uses small nodes so as to reduce weight increase (weight-minimization criterion), and the physical storage scheme, where several nodes are mapped to the same physical page so as to reduce the space overhead.

When a node N is split, according to the above scheme, the resulting new node is stored in the same physical page that contains N. However, if this page is full, i.e., it contains *Nfr* nodes, then the new node is stored in the first page that can accommodate the node (the addresses of such nodes along with the corresponding *Nfr* are kept in a list in main memory). If no such page exists, a new physical page is created to store the node. For static data, a different mapping scheme can be followed. First, the index is constructed and then the nodes are stored in physical pages (*Nfr* nodes in each page) according to their *Breadth-First* order in the index. Thus, nodes that are close are stored in the same physical page.

Consequently, the proposed paging scheme performs a further consideration of the weight-minimization criterion and can produce a very effective storing of the index on disk. Experimental results in Sect. 5.3.5 illustrate its superiority.

5 Performance results

In this section we present the experimental results on the performance of S^{3}B. Since the signature-table comprises the state-of-the-art method for similarity searching in basket data, it is used here for comparison. We use both synthetic and real datasets to examine the performance of the k-nearest-neighbor and the range query with respect to the query size, the size of the database, the length of transaction, and the number of items in the domain. In addition, we examine the impact of factors such as the skewness of item distribution within transactions and the buffer size. Moreover, the efficiency of the proposed enhancements, presented in Sect. 4, is measured against existing approaches.

5.1 Methodology

We have implemented both S^{3}B and signature-table (denoted as ST henceforth) in C. A significant issue for ST is the scheme for the organization of transaction data on disk (see Fig. 2). In [3] no indication is provided for this issue. Nevertheless, the required organization scheme presents similarities with the one for storing data records in traditional indexes (e.g., B$^+$-tree). Therefore, the approach of *posting-lists* [8] can be followed. With such a scheme, ST can handle dynamic data. It can be assumed that patterns (i.e., associations between pairs of items) are valid for some period of time (after the insertion of new transactions). Hence, given a partitioning that has been derived from the existing transactions, new transactions are inserted according to their supercoordinate value, in the available bucket of the corresponding posting-list. During the insertions, in case of bucket overflows, the sizes of new buckets are determined according to [8]. As described, S^{3}B can handle dynamic data without any problem. For S^{3}B, the transactions at the data level are also organized with the scheme of [8].

For static data, we developed an optimization for ST by bulk-loading the transactions in disk pages. Transactions are initially sorted according to their supercoordinate values and then they are inserted in this sorted order. Thus, transactions occupy consecutive ST entries and completely fill disk pages (a disk page may contain transactions with different, but consecutive, supercoordinates). Whenever a disk page is full, the next transaction is inserted in a new page. This way, ST presents reduced space overhead and improved performance during query execution (as will be shown in the following). However, new transactions cannot be inserted, since the insertion, even of few, transactions in arbitrary places requires the shifting of all the remaining transactions, which is an operation with prohibitive cost for data stored on disk. Consequently, this optimization is applied only for static data. Moreover, we apply a further optimization by removing all supercoordinates from the signature-table that do not contain any transactions. This way, the CPU overhead is improved, since ST has to sort fewer supercoordinates during query execution. (Following the approach of [3] for ST, we first compute the similarities of the table entries against the query and then we sort them. We have also tried other variations, such as using a B-tree to get a sorted order of similarities, but they did not produce significantly different results.) Again, this optimization applies for static data, because the removing of supercoordinates cannot be applied for dynamic data (new transactions may be assigned to such supercoordinates).

Finally, the default page size was set to 4 K, the default signature size for S^{3}B was set to 1,000 and *Nfr* to 3. The default signature cardinality, K, for ST was set to 15, as indicated in [3]. We tried several similarity measures, presented in [3]. Here, we use the similarity measure $f(x, y) = x/y$. The other examined measures did not show differences in relative performance between S^{3}B and ST, and are omitted for brevity. The default buffer size was set to 25% of the dataset size (for ST,

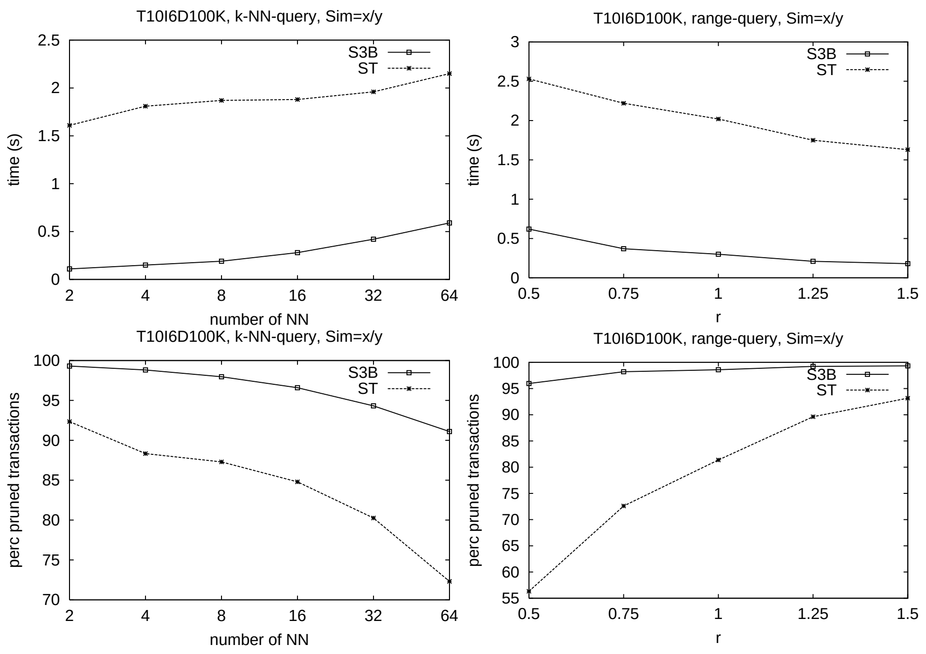

Fig. 8. Performance for similarity queries against query size. *Left*: k-nearest-neighbor query with respect to k. *Right*: range query with respect to r (radius) threshold

the table with the supercoordinates occupies pinned memory space within the buffer).

5.2 Datasets and measures of performance

In order to have better control over the impact of several data characteristics, we used the synthetic data generator for basket market data which is presented in [2] and is also used in [3]. Moreover, we used real data that are obtained from user traces in a web portal. In order to follow the distribution of each dataset, we selected query transactions as transactions from the dataset.

For the synthetic data, the notation followed is TxIyDz, denoting that the mean size, $|T|$, of transaction is x, the mean size, $|I|$, of the potentially large itemsets is y, and that the database contains z transactions. Following [3], we use correlation factor (i.e., the factor of common items between successive transactions during their generation) equal to 0.5. The default values were the ones in [3]: x equal to 10, y equal to 6, z equal to $100K$. The number of total items was equal to 1,000. The other values (noise factor, etc.) were set to the default values of the generator [2].

The characterization of performance is given with the query execution time (wall-clock time). Additionally, for purposes of comparison, we use as performance measure the percentage of pruned transactions (i.e., those not examined by the similarity query), since this is the only measure used in [3].

5.3 Results

5.3.1 Scalability

We examined the impact of the query size and the database size. We use static transaction databases, so as to be able to use the developed optimizations for ST (the case of dynamic data is examined in the following). First, we measured the performance with respect to k for the k-nearest-neighbor query, and to r for the range query. The results for the T10I6D100K dataset are depicted in the left and right parts of Fig. 8, respectively. Both the execution time (in seconds) and the pruning percentage is shown for each case.

Focusing on the k-nearest-neighbor query (upper-left part of Fig. 8), S^{3}B presents an improvement in the execution time, which ranges from a factor of 10, for smaller k, to 4, for larger k. Moreover, S^{3}B achieves a significantly better pruning percentage, as depicted in the lower-left part of Fig. 8. With increasing k, the pruning percentage of ST reduces significantly, whereas for large k, S^{3}B achieves a percentage larger than 90%. Analogous results were obtained for the range query (upper-right part of Fig. 8). The improvement in execution time ranges from a factor of 9, for large r, to 4, for small r. Turning our attention to the measure of pruning percentage (lower-right part of Fig. 8), it is noticeable that the pruning of ST reduces rapidly for smaller values of r (i.e., for larger range queries), whereas S^{3}B attains a pruning percentage that is constantly larger than 95%.

We now move on to examine the scalability to the database size, measured in numbers of transactions. We report results only on the nearest-neighbor query[8], since results on the range query were similar and are omitted. We used analogous datasets to the one of the previous experiment, but with varying numbers of transactions. Thus, they are denoted as T10I6Dx. Figure 9 depicts the results.

As illustrated (left part of Fig. 9), S^{3}B scales linearly to the database size. On the other hand, the execution time for ST also increases with increasing database size, but it does not show a strictly linear behavior. Clearly, S^{3}B outperforms ST in all cases. For large database sizes, the performance im-

[8] Since query transactions are selected from the dataset (to follow its distribution), we use the 2-nearest-neighbor query to avoid the result of the query transaction itself. This approach is followed in the forthcoming experiments as well.

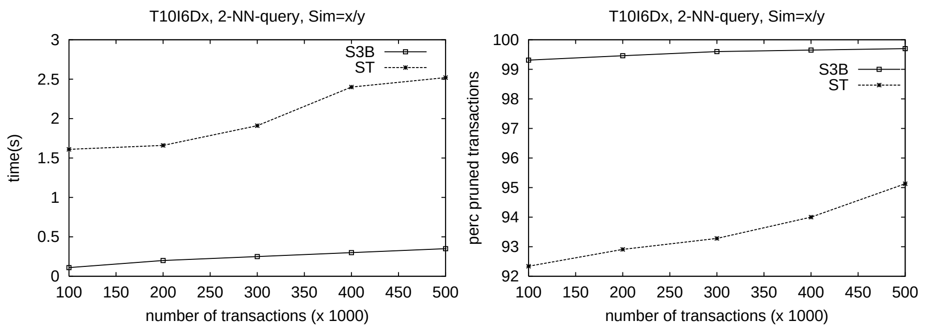

Fig. 9. Scalability results with respect to database size

provement is by a factor of 7 (note that for smaller sizes, the improvement is larger). Therefore, S^{3}B is scalable to large databases. Focusing on the pruning percentage (right part of Fig. 9), we observe that for both methods it increases with increasing database size. This is in accordance with the findings of [3]. The reason is that, for larger databases, a relatively larger percentage of transactions can be pruned by the same optimum similarity bound [3]. Nevertheless, in all cases, S^{3}B presents a significantly better pruning percentage, compared to ST.

5.3.2 Sensitivity

We turn our attention to the sensitivity against the mean transaction size and the total number of items in the database (i.e., the cardinality of the universal domain). For the former case, we used analogous datasets to the ones in previous experiments, but with varying T; therefore, they are denoted as Tx.I6.D100K. For the latter case, we used datasets denoted as T10.I6.D100K, which are analogous to the one used in the first experiment, but with a varying total number of items. The results are depicted in Fig. 10.

As depicted in the upper-left part of Fig. 10, the execution time of ST and S^{3}B increases with increasing T. However, the increase for ST is significantly more noticeable. S^{3}B is much less affected by increased T, and it presents an improvement of a factor of 8 for larger values of T. Focusing on the pruning percentage (lower-left part of Fig. 10), as expected, it reduces with increasing T. The reduction is rapid in the case of ST, whereas S^{3}B achieves a pruning percentage larger than 97% in all cases.

The upper-right part of Fig. 10 depicts the execution times for varying number of items. S^{3}B is not affected by an increasing number of items. The same can be stated for the pruning percentage of S^{3}B (lower-right part of Fig. 10), indicating that the signature-based representation of S^{3}B is effective for transaction databases with large numbers of items. On the other hand, ST presents an increase in the execution time with an increasing number of items. For a larger number of items, the single-link method, upon which the partitioning is based for ST, may not retain the clustering quality (see Sect. 3.4). Although the pruning percentage of ST increases slightly (lower-right part of Fig. 10), the processing of the remaining transactions requires larger time, due to the poor distribution of the signature-table entries, as described above.

Next, we examine the impact of skewness in the distribution of items within transactions. We modified the data generator of [2] accordingly. Based on the approach in [22] and by assuming a mapping of items into consecutive integer numbers, the probability of item i to be included within a transaction is the following (note that the first factor is the normalization constant):

$$p_i = \frac{1}{\sum_{j=1}^{N} 1/j^{1-\theta}} \cdot \frac{1}{i^{1-\theta}}\ , \quad \text{N is the total number of items}$$

Therefore, according to the value of θ, we can control the distribution of items. For $\theta = 0$, p_i follows a pure Zipf distribution, whereas for $\theta = 1$, p_i corresponds to the uniform distribution (note that the generator of [2] uses an exponential item distribution).

We used the T10I6D80K datasets with varying values of θ and we examined the 2-nearest-neighbor query. The results are depicted in Fig. 11. S^{3}B is not significantly affected by large skewness in item distribution, i.e., small values of θ. Its execution time (left part of Fig. 11) presents a slight increase only for very small θ values, mainly because the collisions between frequent items (through the used hash function) result in a small increase in the false-drop probability. This is also shown in the pruning percentage of S^{3}B (right part of Fig. 11), which presents analogous behavior. In contrast, it is worth noting that for ST the execution time reduces with reducing θ. For very small values of θ – which, however, are not realistic for the case of real-world examples[9] but are considered here for comparison purposes – ST achieves lower execution times. In these extreme cases, very few items appear frequently and the others appear in a very small number of transactions. Thus, due to the partitioning of ST, the former items comprise a small number of signatures, which, however, are activated very frequently. Consequently, queries tend to access the transactions indexed by these signatures, whereas the remaining ones are activated rarely. Because of the effective use of buffering, the improvement in the execution time follows (since the corresponding pages of the frequently activated signatures are accessed repetitively, their buffering is effective). Nevertheless, S^{3}B clearly outperforms ST in all cases. Focusing on the pruning percentage, S^{3}B achieves a higher percentage, although ST presents an increase (about 2.7%) in its pruning percentage with reducing θ, since, for reducing θ, each query for ST concentrates on fewer transactions (as described previously).

Evidently, the performance of both S^{3}B and ST depends on the size of the available buffering space. We examined the impact of buffer size, measured as a percentage of the database size. We used the T10I6D100K dataset, and the results for the 2-nearest-neighbor query are depicted in Fig. 12. To clearly

[9] This is in accordance with the description of [22], where only values larger than 0.6 are considered for θ.

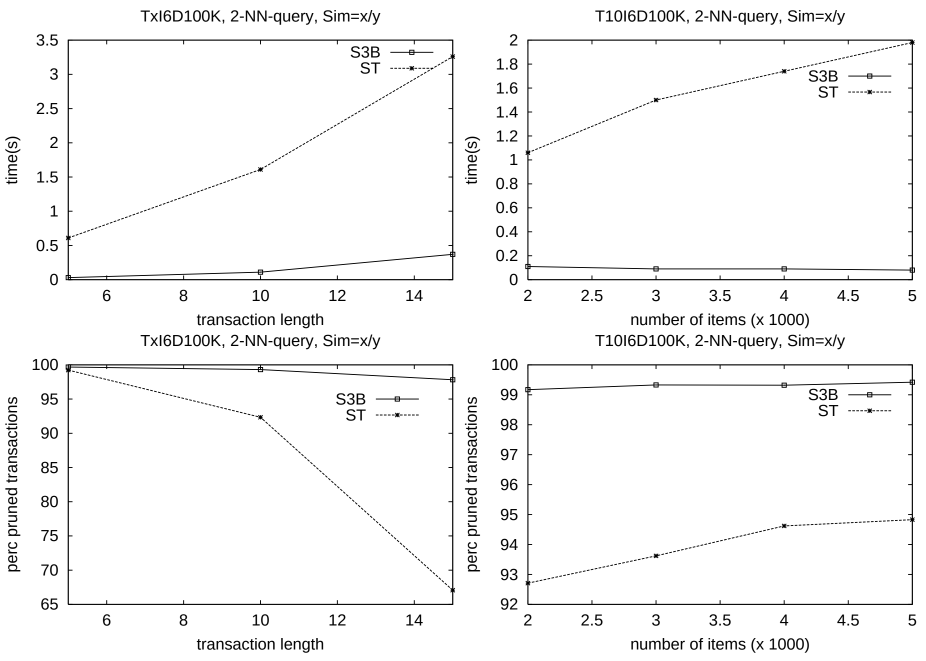

Fig. 10. *Left*: nearest-neighbor query with respect to mean size, T, of transactions. *Right*: nearest-neighbor query with respect to total number of items in database

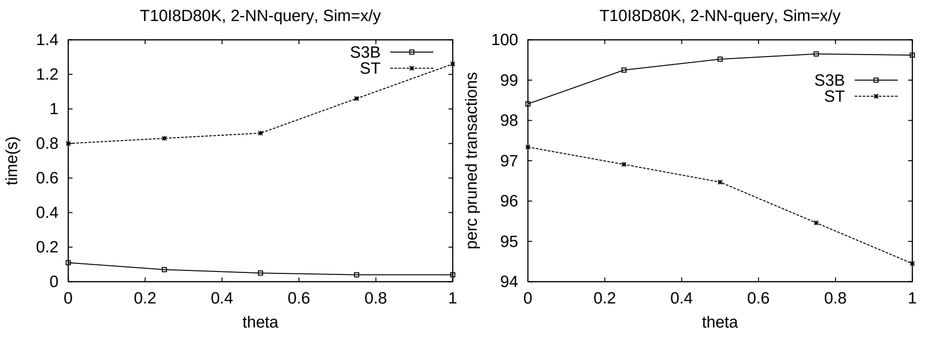

Fig. 11. Sensitivity against item distribution

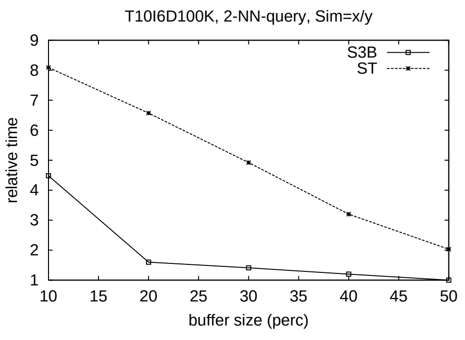

Fig. 12. Sensitivity against buffer size

present the impact of increasing buffer size, the relative execution times are illustrated in Fig. 12 (i.e, for each buffer size, the ratio of execution time against the lowest one, attained by S^{3}B for the largest examined buffer size, is given). As illustrated, ST loses out significantly for small and medium buffer sizes. S^{3}B presents a large improvement between 10% and 20%, due to the overhead presented by the examination (fetching from secondary storage) of a large percentage of upper-level nodes in the case of small buffer sizes (less than 20%). For larger sizes, as expected, both methods converge to a point, since a large part of the contents of the indexes can be accommodated in the buffer.

5.3.3 Dynamic data

We now move on to the comparison between S^{3}B and ST for dynamic data. As described, in this case the developed optimizations cannot be used for ST. This mainly affects the I/O overhead of ST, since, different to static data, the transactions cannot be bulk-loaded into the data pages of ST. More precisely, for a given collection of transactions, the pruning percentage is independent from the method of their loading (either dynamic insertion or bulk-loading), because the same transactions will be pruned for a given query. However, the examination of the remaining transactions requires a different number of disk accesses between the two cases. For this reason, we focus on the number of disk accesses to measure the impact on the factor of I/O overhead. We used the T10I6D100K datasets, and the first 50% of the transactions were used to derive the partitioning of ST (the others were dynamically inserted). We examined the k-nearest-neighbor query for varying k, and the range query for varying r. The results are depicted in the left and right parts of Fig. 13, respectively, (to clearly illustrate the increase in the number of disk accesses with respect to query size, their relative numbers are given, which are normalized to the minimum number of accesses in each measurement that is obtained from the smaller query size). As shown, ST loses out significantly. It presents a large performance degradation for increasing k and decreasing r. This indicates that the perfor-

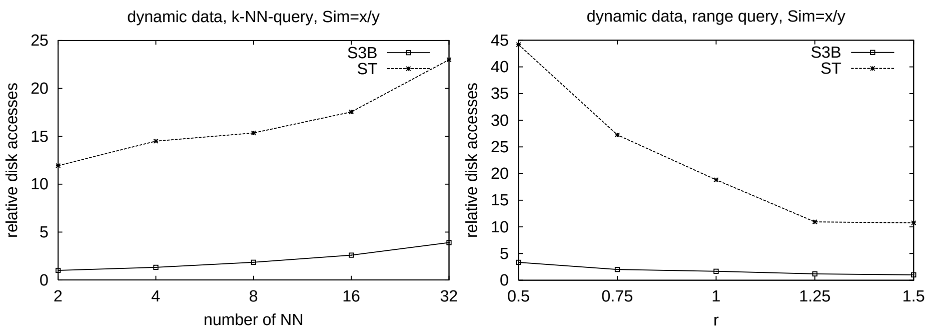

Fig. 13. Comparison for dynamic data

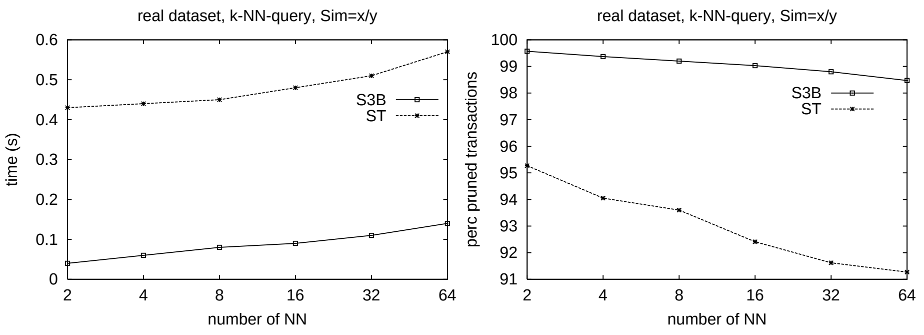

Fig. 14. Performance results for the real dataset and the k-nearest-neighbor with respect to k

mance of ST is not satisfactory for large collections of dynamic transactions.

5.3.4 Real data

We conclude the comparison between S^{3}B and ST by conducting experiments with real data. We present the results on a real Web-log trace from the *ClarkNet* portal, which has been used as benchmark in related work on Web log data mining[10] (see [21] for more details). We cleansed the log (e.g., by removing CGI scripts, stale requests, etc.) and used the *MF* algorithm [6] to create the transactions. The final database contained 133,248 transactions and 2,336 items (i.e., URLs). The results for the k-nearest-neighbor query are depicted in Fig. 14.

As illustrated, both S^{3}B and ST present increasing execution time with respect to k (left part of Fig. 14), but S^{3}B clearly outperforms ST. This is in accordance with the results presented in the previous experiments. Note that the execution times for both algorithms are relatively small, due to the small transaction lengths that Web log data have (the average transaction length was less than 5). This is in accordance with the results presented in Fig. 10, where the execution time reduces with reducing transaction length. Focusing on the pruning percentage (right part of Fig. 14), we can observe that for both S^{3}B and ST it reduces with increasing k. However, the reduction is more noticeable in the case of ST, and S^{3}B presents a better pruning percentage. This result is in accordance with the corresponding ones presented in the previous experiments.

5.3.5 Evaluation of enhancements

We measured the performance improvements due to the proposed enhancements, which were presented in Sect. 4. The

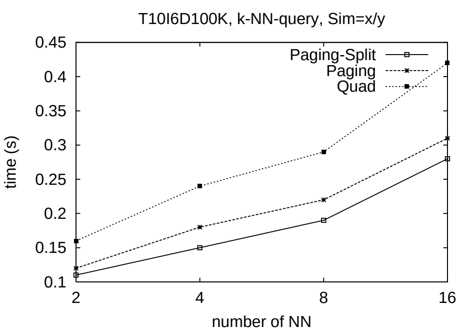

Fig. 15. Results on the proposed enhancements

results for the T10I6D100K dataset and for the k-nearest-neighbor query are depicted in Fig. 15. In this figure, 'Quad' denotes the index which: a) does not use the paging scheme of Sect. 4.2 (i.e., in 'Quad' the logical page size is identical to the physical page size); and b) uses a split algorithm of low (quadratic) complexity, described in [28]. In the same figure, 'Paging' denotes the index which: a) uses the paging scheme of Sect. 4.2 (with $Nfr = 3$); and b) uses the same split algorithm that is used by 'Quad'. Therefore, the performance of 'Paging' indicates the effectiveness of the proposed paging scheme, compared to 'Quad' that does not use this scheme. Evidently, the improvement reaped due to the paging scheme is significant in terms of query execution time.

Moreover, in Fig. 15, 'Paging-Split' denotes the index which uses both the proposed paging scheme and the proposed split algorithm of Sect. 4.1. The comparison between 'Paging-Split' and 'Paging' illustrates the improvement achieved by the proposed split algorithm, compared to the quadratic algorithm used in 'Paging'. Note that the effectiveness of the proposed paging scheme, which is used by both 'Paging-Split' and 'Paging', does not further allow for a large improvement by the split algorithm itself. Nevertheless, 'Paging-Split' clearly achieves the lowest execution times among all methods, indicating the efficiency of the combination of the proposed enhancements.

[10] available at http://ita.ee.lbl.gov/html/traces.html

6 Conclusions

We considered the problem of similarity searching in market basket databases, i.e., the finding of transactions which satisfy the criterion of a similarity query. We are interested in complete results, not approximations and, moreover, in a scheme which is independent from the similarity measure used during query execution. Both the k-nearest-neighbor and the range query are examined. They suit the requirements for new query processing techniques, presented by emerging applications (e.g., Web recommendation systems).

We proposed a novel scheme, namely S^3B, which uses a signature-based representation for transaction data. We addressed the problems of existing signature schemes by proving the correctness of the proposed type of representation. On the basis of index structures for signatures, we developed a twofold enhancement for significantly improving the efficiency of existing indexes. In addition, we presented algorithms for the processing of nearest-neighbor and range queries which use the properties provided by the proposed representation method.

We carried out experimental results for a variety of factors. We examined the performance relative to the query size, the scalability to the database size, the effect of transaction size, and the cardinality of the domain. In addition, we measured the impact of skew in item distribution and the effect of buffering. In all the experiments S^3B significantly outperforms the signature-table, an existing method for similarity searching in market basket databases. In summary, S^3B is an efficient and scalable scheme for large transaction databases.

Future work includes the extension of the proposed approach towards the development of clustering algorithms for market basket databases.

References

1. Agrawal R, Faloutsos C, Swami A (1993) Efficient Similarity Search in Sequence Databases. Proc. International Conference on Foundations of Data Organization and Algorithms (FODO'93), pp 69–84
2. Agrawal R, Srikant R (1994) Fast algorithms for mining association rules in large databases. Proc. International Conference on Very Large Databases (VLDB'94), pp 207–216
3. Aggarwal C, Wolf J, Yu PS (1999) A new method for similarity indexing of market basket data. Proc. ACM International Conference on Management of Data (SIGMOD'99)
4. Baeza-Yates R, Ribeiro-Neto B (1999) Modern information retrieval. Addison-Wesley, reading, Mass., USA
5. Berchtold S, Böhm C, Kriegel HP (1998) The Pyramid Technique: towards breaking the curse of dimensionality. Proc. ACM International Conference on Management of Data (SIGMOD'98), pp 142–153
6. Chen MS, Park J, Yu PS (1998) Efficient data mining for path traversal patterns. IEEE Trans Knowl Data Eng 10(2):209–221
7. Deppish U (1986) S-tree: a dynamic balanced signature index for office retrieval. Proc. ACM International Conference on Information Retrieval (SIGIR'86), pp 77–87
8. Faloutsos C, Jagadish H (1992) On B-tree indices for skewed distributions. Proc. International Conference on Very Large Databases (VLDB'92), pp 363–374
9. Faloutsos C, Ranganathan M, Manolopoulos Y (1994) Fast subsequence matching in time-series databases. Proc. ACM International Conference on Management of Data (SIGMOD'94), pp 419–429
10. Gionis A, Gunopulos D, Koudas N (2001) Efficient and tunable similar set retrieval. Proc. ACM International Conference on Management of Data (SIGMOD'01)
11. Goldberg D, Nichols D, Oki B, Terry D (1992) Using collaborative filtering to weave an information tapestry. Comm ACM 35(12):61–70
12. Guha S, Rastogi R, Shim K (2001) Cure: an efficient clustering algorithm for large databases. Inf Syst 26(1):35–58
13. Guttman A (1984) R-trees: a dynamic index structure for spatial searching. Proc. ACM International Conference on Management of Data (SIGMOD'84), pp 47–57
14. Hellerstein J, Pfeffer A (1994) The RD-tree: an index structure for sets. Tech. Report (1252), University of Wisconsin, Madison, Wis., USA
15. Helmer S, Moerkotte G (1997) Evaluation of main memory join algorithms for joins with set comparison join predicates. Proc. International Conference on Very Large Databases (VLDB'97), pp 386–395
16. Hjaltason G, Samet H (1995) Ranking in spatial databases. Proc. International Symp. on Large Spatial Databases (SSD'95), pp 83–95
17. Hill W, Stead L, Rosenstein M, Furnas G (1995) Recommending and evaluating choices in a virtual community of use. Proc. of Conference on Human factors in Computing Systems (CHI'95), pp 194–201
18. Ishikawa Y, Kitagawa H, Ohbo N (1993) Evaluation of signature files as set access facilities in OODBs. Proc. ACM International Conference on Management of Data (SIGMOD'93), pp 247–256
19. Katayama N, Satoh S (1997) The SR-tree: an index structure for high dimensional nearest neighbor queries. Proc. ACM International Conference on Management of Data (SIGMOD'97), pp 369–380
20. Konstan J, Miller B, Maltz D, Herlocker J, Gordon L, Riedl J (1997) GroupLens: applying collaborative filtering to usenet news. Comm ACM 40(3):77–87
21. Nanopoulos A, Katsaros D, Manolopoulos Y (2002) A data mining algorithm for generalized web prefetching. IEEE Trans Knowl Data Eng (accepted)
22. Park J, Chen MS, Yu P (1997) Using a hash-based method with transaction trimming for mining association rules. Trans Knowl Data Eng 9(5)
23. Roussopoulos N, Kelley S, Vincent F (1995) Nearest neighbor queries. Proc. ACM International Conference on Management of Data (SIGMOD'95), pp 71–79
24. Sacks-Davis R, Ramamohanarao K (1983) A two-level superimposed coding scheme for partial match retrieval. Inf Syst 8(4)
25. Sarwar B, Karypis G, Konstan J, Riedl J (2000) Application of Dimensionality Reduction in Recommender Systems. WebKDD Workshop
26. Sarwar B, Karypis G, Konstan J, Riedl J (2000) Analysis of recommendation algorithms for e-commerce. Proc. ACM Conference on Electronic Commerce 2000 (EC'00), pp 158–167
27. Sibson R (1973) SLINK: an optimally efficient algorithm for the single link cluster method. Comput J 16
28. Tousidou E, Nanopoulos A, Manolopoulos Y (2000) Improved methods for signature tree construction. Comput J 43(4)
29. Wang K, Xu C, Liu B (1999) Clustering transactions using large items. Proc. International Conference on Information and Knowledge Management (CIKM'99), pp 483–490