An Efficient Block Index Scheme with Segmentation for Spatio-Textual Similarity Join

Yiming Xiang[1], Yi Zhuang[1*], Nan Jiang[2]

[1]College of Computer & Information Engineering, Zhejiang Gongshang University, P.R.China
[e-mail: futuretech@mail.zjgsu.edu.cn, zhuang@zjgsu.edu.cn]
[2] Hangzhou First People's Hospital, Hangzhou, P.R.China
[e-mail: zy158cn@gmail.com]
*Corresponding author: Yi Zhuang

Received September 24, 2016; revised February 20, 2017; accepted March 29, 2017; published July 31, 2017

Abstract

Given two collections of objects that carry both spatial and textual information in the form of tags, a Spatio-Textual-based object Similarity JOIN (*ST-SJOIN*) retrieves the pairs of objects that are textually similar and spatially close. In this paper, we have proposed a *block index-based approach* called BIST-JOIN to facilitate the efficient *ST-SJOIN* processing. In this approach, a dual-feature distance plane (*DFDP*) is first partitioned into some blocks based on four segmentation schemes, and the *ST-SJOIN* is then transformed into searching the object pairs falling in some affected blocks in the *DFDP*. Extensive experiments on real and synthetic datasets demonstrate that our proposed join method outperforms the state-of-the-art solutions.

Keywords: join rectangle, similarity join, near-duplicate image detection

1. Introduction

With the increasing complexity of the data representation over the years, data can be easily 'tagged' with different types of information, such as keywords and spatial locations, etc. For example, webpages contain keywords and they may also be associated to locations; photographs in photo-sharing services, such as Flickr, are assigned descriptive tags and spatial locations [1, 2]; persons in social networks and customer databases have profile entries (keywords) and addresses. The enrichment of objects with multi-source descriptive information allows for more complex queries and analysis over the data.

As one of the important object queries, a Spatio-Textual-based object Similarity JOIN (*ST-SJOIN*) has played a critical role in many applications. Here are two representative examples:

- ***Near-identical object detection***. One of the most important applications of the *ST-SJOIN* is near-identical object detection. For example, consider a database of spatially and textually tagged images (e.g., Flickr). Finding similar image pairs based solely on their tag similarity may not be sufficient, if the tags are not location dependent. Thus, an image tagged as '*house*' is textually similar to other house photos around the world, but can only be actually similar to photographs of the same house (taken from nearby locations). A spatio-textual (self) join can be used to identify pairs of images showing the same subject.
- ***Personalized recommendation***. Another important application is a personalized recommendation. For example, a user wants to obtain some objects for personal needs. That is, the returned objects need to be textually similar to the preference and spatial closely to the preference location. So, the *ST-SJOIN* of the query object with user preferences and the objects in a dataset can return some pairs of objects that satisfy users' needs.

In this paper, we identify and solve the problem of the *ST-SJOIN* for objects that is to retrieve a pair of objects of which the *textual* or *spatial* information are similar. Formally, given two collections of objects R and S that carry both textual and spatial information, the *ST-SJOIN* retrieves the subset J of $R \times S$, such that for every $(r,s) \in J$, two following criteria have to be satisfied:

- r is spatially close to s, based on a distance threshold (i.e., $dis(r,s) \leq \theta_S$, where *dist* denotes distance between locations),
- r is textually similar to s, based on a similarity threshold θ_T (i.e., $tSim(r,s) \leq \theta_T$, where *tSim* denotes textual similarity).

Suppose that there are two object sets R and S. There are six objects (r_1 to r_6) in R and four objects (s_1 and s_4) in S are joined based on their *textual content* (T) and *spatial locations* (S). Assuming qualifying pairs should have textual similarity *tSim* at least θ_T=0.5 and spatial distance dist at most θ_S=0.4, the result of the join is (r_3, s_1), (r_1, s_3) and (r_6, s_4).

To support efficient *ST-SJOIN* processing, in this paper, we have proposed a *block index approach* called BIST-JOIN. In particular, the object pairs in the high-dimensional feature spaces are first mapped into a dual-feature distance plane (*DFDP*), and then the *ST-SJOIN* is transformed into searching the object pairs falling in the join rectangle (*JR*) in this plane without sets intersection processing. To further improve the join efficiency, we also propose a unified index structure called the *block index*.

The primary contributions of this paper are as follows:

1. We present a block index-based method (*BIST-JOIN*) to facilitate the efficient spatio-textual join processing.
2. We design four segmentation schemes to partition the *DFDP* into several blocks.

3. We perform extensive experiments on real and synthetic datasets to evaluate the efficiency of our proposed object similarity join algorithm.

The rest of this paper is organized as follows. In Section 2, we provide a background of our work. Then in Section 3, we give the problem formulation of this work. In Section 4, we introduce an efficient similarity join algorithms based on spatial and textual features. In Section 5, we report the results of extensive experiments that are designed to evaluate the efficiency of the proposed approach. Finally, we conclude in the final section.

2. Related Work

Similarity join processing is a long standing yet challenging research topic that has attracted much attention in several research communities. Existing work includes *spatial join* [6, 7, 14], *textual join* [3, 12, 13, 14] and *set-similarity join* [3, 5, 9, 13], etc.

For spatial joins, efficient algorithms [6] have been developed for data indexed by R-trees. The ST-SJOIN [15] extends the ε-distance join [7]. Given two spatial datasets R and S, the ε-distance join finds the pairs (r, s) such that $r \in R$, $s \in S$, and $dist_l(r, s) \leq \varepsilon$. The ε-distance join processing is similarly to a spatial intersection join: the R-trees that index R and S are concurrently traversed by recursively following pairs of entries for which the minimal bounding rectangles (MBR) have minimum distance at most ε. Further, Chan [7] proposed techniques for minimizing the distance computation cost between objects and MBRs.

For set-similarity join, given a collection D of set-valued data, the problem is to find pairs (x, y) of sets in D, such that $sim_t(x, y) \geq \theta$, where $sim_t(.,.)$ is a similarity function and θ is a threshold. The main application of set-similarity joins is near-duplicate object detection [9]. Set-similarity joins can also be used to facilitate string matching. For example, Gravano et al. [10] showed that the edit distance between two strings can be bounded by set-similarity measures defined on two sets of q-grams, which approximate the strings. The prevalent approach in the past is to solve an approximate version of the problem, i.e., finding most of, if not all, similar objects. Several synopsis-based schemes have been proposed and widely adopted [16, 17, 19].

A recent trend is to investigate algorithms that compute the similarity join exactly. Recent advances include inverted index-based methods [11], prefix filtering-based methods [5, 18], and signature-based methods [3]. Among them, the recently proposed All-Pairs algorithm [5] has been demonstrated to be highly efficient and scalable to tens of millions of records. Nevertheless, the All-Pairs algorithm, as well as other prefix filtering-based methods, usually generates a huge amount of candidate pairs, all of which need to be verified by the similarity function. Computing set-similarity joins based on inverted files [14] was first proposed in [11]: for each object x, the inverted lists that correspond to x's elements are scanned to accumulate the similarity between x and all other objects. Several optimizations over this approach are proposed, including scanning only a smaller subset of x's lists and performing a single pass over the data that constructs the inverted index and computes the join result at the same time. Chaudhuri et al. [8] introduced an efficient filter-refinement framework for set-similarity joins, based on the observation that for two sets x, y to satisfy $sim(x,y) \geq t$, a necessary condition is that the prefixes of x and y should have at least some minimum overlap. Arasu et al. [3] showed that this prefix-based filtering is one of the possible summary schemes that one could use as necessary conditions and provided alternative schemes with theoretical bounds on their effectiveness. Bayardo et al. [5] proposed an efficient framework for evaluating set-similarity joins, which minimizes the necessary elements to add in the inverted file, during join evaluation, based on pre-computed bounds on the element weights in the sets and appropriate orderings for the domain of set elements and the database. Xiao et al. [13] optimized this method by integrating positional filtering

into the All-Pairs algorithm [5], which is called the ppjoin. Based on it, they further employed a suffix filtering to the ppjoin (i.e., ppjoin+) which is complementary to the existing prefix filtering method and can work on tokens both in the prefixes and the suffixes. The technique was later extended to compute top-*k* set-similarity joins [12]. Deng et al. [21] proposed an efficient partition based method for exact set similarity joins. Hu et al. [22] proposed a top-k spatio-textual similarity join scheme called the *TOPK-STJOIN*, which is an improvement of the *ST-SJOIN* [15]. Li et al. [23] proposed an efficient similarity search and join on multi-attribute data. Tao et al. [24] presented an efficient top-k *SimRank*-based similarity join. Deng et al. [27] devised a pivotal prefix based filtering algorithm for string similarity search. Shang et al. [26] presented a knowledge-aware similarity join. Ta et al. [27] presented a signature-based trajectory similarity join.

3. Problem Formulation

First we briefly summarize the notations that will be used in the rest of the paper in **Table 1**.

Table 1. Summary of frequent symbols

Symbol	Description
R, S	two object sets
T	a tag set
O_i	the i-th object
$\lvert\bullet\rvert$	the number of objects in $\bullet$
$\lceil\bullet\rceil$	the integral part of $\bullet$
k	the number of the segmentation for each feature
$dist(I_i,I_j)$	spatial distance between two objects
$tSim(I_i,I_j)$	textual similarity distance between two objects based on Jacard distance
θ_T	a join threshold for textual feature
θ_S	a join threshold for spatial feature

Definition 1 (SPATIO-TEXTUAL OBJECT). *A spatio-textual object O_i can be modeled as a triplet:*

$$O_i =< i, T, S > \quad (1)$$

where i is the ID number of O_i, T refers to the tag set assigned by users to O_i, and S refers to the spatial information of O_i.

Definition 2 (TEXTUAL SIMILARITY DISTANCE). *For each pair of objects (i.e, O_i and O_j), their textual similarity distance (tSim) can be derived in Eq.(2):*

$$tSim(O_i, O_j) = 1 - \frac{|T(O_i) \cap T(O_j)|}{|T(O_i)| + |T(O_j)| - |T(O_i) \cap T(O_j)|} \quad (2)$$

where $T(O_i)$ is the tags assigned for O_i.

Definition 3 (SPATIAL DISTANCE). *For each pair of objects (i.e., O_i and O_j), their spatial distance (dist) can be derived in Eq.(3):*

$$dist(O_i, O_j) = \sqrt{(O_i.x - O_j.x)^2 + (O_i.y - O_j.y)^2} \quad (3)$$

where $O_i.x$, $O_j.x$, $O_i.y$ and $O_j.y$ are the latitude and longitude axis values of O_i and O_j , respectively.

For illustration, based on definition 3, a normalized spatial distance is proposed in the

range of [0, 1].

Definition 4 (NORMALIZED SPATIAL DISTANCE). *Given a spatial distance: dist(O_i,O_j), its corresponding normalized spatial distance (Dist) can be defined in Eq.(4):*

$$Dist(O_i, O_j) = \frac{dist(O_i, O_j)}{\arg\max_{i,j}\{dist(O_i, O_j)\}} \tag{4}$$

such that Dist(O_i, O_j) ∈[0,1], where dist(O_i,O_j) is defined in Eq.(3)

Definition 5 (SPATIO-TEXTUAL SIMILARITY JOIN). *Given two object sets: R and S, their spatio-textual similarity join is a set, which can be formally represented as:*

$$R \underset{\theta_T,\theta_S}{\bowtie} S = \{(r_i, s_j) \in R \times S | tSim(r_i, s_j) \geqslant \theta_T \wedge Dist(r_i, s_j) \leqslant \theta_S\} \tag{5}$$

4. The BIST-JOIN Algorithm

For the *ST-SJOIN* processing, in this section, we propose a *block index-based ST-JOIN* (BIST-JOIN) method.

4.1 Motivations

As the *ST-SJOIN* processing involves CPU and I/O intensive intersection operation, the join performance can be significantly improved if such intersection processing can be reduced or avoided.

Definition 6 (JOIN RECTANGLE). *A join rectangle is a rectangle in which two axis correspond to two features, respectively; and the maximal value of each feature distance are two threshold values (i.e., θ_T and θ_S), respectively. Formally denoted as JR(θ_T,θ_S).*

Based on Definition 6, as shown in **Fig. 1**, given the two join thresholds (i.e., θ_T and θ_S), the *ST-SJOIN* problem can be transformed to a new problem of obtaining the object pairs falling in the corresponding *JR*(θ_T,θ_S). Compared with the above two methods, the object pairs in the *JR* can be easily and efficiently obtained without much CPU-intensive intersection operation.

How to effectively and efficiently obtain the object pairs in the *JR* is a challenging issue. In this subsection, we propose a Block Index-based ST-JOIN algorithm called the *BIST-JOIN*. The basic idea behind it is to segment the range of each feature distance into sub-ranges. In other words, the *DFDP* is partitioned into blocks in which there exists the corresponding object pairs. The *ST-SJOIN* processing is then transformed into searching for the object pairs in the affected blocks of the *JR*(θ_T,θ_S) in the *DFDP*.

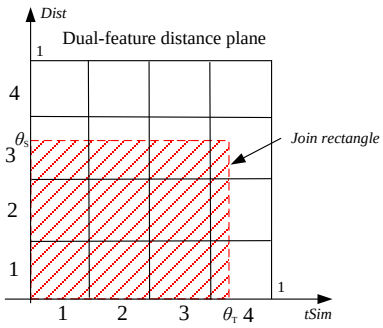


Fig. 1. Join Rectangle of a *ST-SJOIN*

4.2 Segmentation Schemes

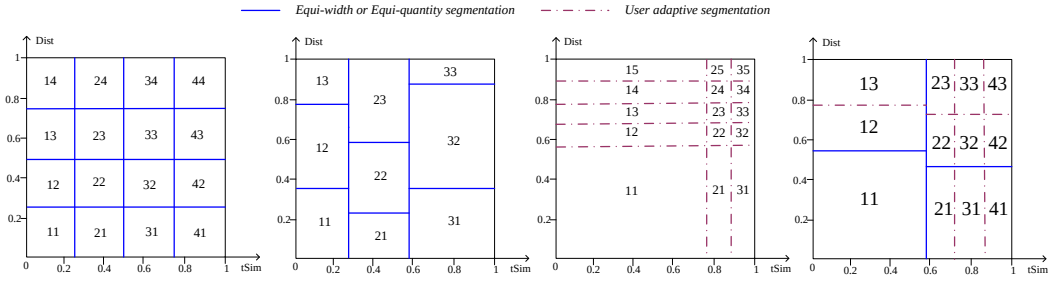
In this subsection, we introduce four segmentation schemes: 1) *equal-width segmentation*, 2) *equal-quantity segmentation*, 3) *user-adaptive segmentation*, and 4) *hybrid segmentation*.

● ***Equal-Width Segmentation***

In the equal-width segmentation-based approach, as shown in **Fig. 2(a)**, we equally segment the range [0, 1] into k segments. For the two features, the number of the blocks is k^2. Due to data skew, the number of object pairs falling in each block is different.

● ***Equal-Quantity Segmentation***

The equal-width segmentation-based approach is more suitable to the feature data that are uniformly distributed. In real applications, the data distribution, however, is skew to some



(a). Equal-width-based (b). Equal-quantity-based (c). User adaptive-based (d). hybrid-based

Fig. 2. An example of the four segmentation schemes

extent. So, this segmentation approach cannot keep a balanced distribution of the number of object pairs in each block. The CPU cost in the refinement process would thus be larger in this case.

To address this problem, we propose an equal-quantity segmentation scheme that enables the number of object pairs in each block to be equal. Specifically, in **Fig. 2(b)**, the object pairs in the feature distance space are first partitioned into k textual segments in terms of the textual distance in which the number of the object pairs in each segment is equal. For each textual segment, its corresponding object pairs are partitioned into k spatial segments in terms of the spatial distance in which the number of the object pairs in each segment is equal.

● ***User-Adaptive Segmentation***

The above two segmentation schemes are based on the distributions of the feature distances themselves. The *ST-SJOIN*, however, involves users' participation and interactions such as thresholds setting, etc. Different users may choose different thresholds (i.e., θ_T and θ_S), which motivates us to segment the feature distance based on the learning of the frequencies of the user–provided thresholds. That is to say, for each feature, the number of segments for a certain range is larger if the user-given threshold is among the range frequently. Based on the above motivations, we present a *user-adaptive* segmentation scheme by which the computation costs in the refinement processing can be reduced effectively from a user's perspective.

Specifically, given t users, assume that each user (U_i) has two thresholds for the object join, which are denoted as θ_T^i and θ_S^i.

Definition 7 (USER THRESHOLD HISTOGRAM). *Given a feature, its corresponding user threshold histogram (UTH) can be modeled as a vector:*

$$UTH = [<Ran_1, Per_1>, <Ran_2, Per_2>, \ldots, <Ran_k, Per_k>] \quad (6)$$

where Ran_i is the i-th sampling range, Per_i is the ratio of the number of thresholds fall in the range Ran_i to that of all the thresholds, formally denoted as: $Per_i = \sum_{\theta^j \in RAN_i} 1 \big/ t$.

Based on Definition 7, assume that the total number of the segments is *k*, then for each range (i.e., Ran_i), the number of segments in this range can be approximately represented as: $\lceil Per_i \times k \rceil + 1$.

- ***Hybrid Segmentation***

Based on the above three segmentation schemes, we propose a hybrid segmentation one that takes advantage of the above three ones.

Fig. 2(d) shows an example of the hybrid segmentation. First, the *DFDP* in this figure is segmented into 4 blocks by the equal-quantity approach. Then, for each block, it can be further segmented according to the frequency of the users' given thresholds falling in the block.

4.3 The Index Scheme

Based on the above four segmentation schemes, the *DFDP* is partitioned into several blocks through segmenting the two feature distances. The block ID (*BID*) can be represented in Eq.(7):

$$BID = \alpha \times sID_T + sID_S \quad (7)$$

where

- sID_T and sID_S are segment IDs for the two features respectively, where they are integers;
- α is a large constant which is used to stretch the bounding segment ID of the textual features (i.e, sID_T).

Definition 8 (BOUNDING SEGMENT ID). *Given a JR(θ_T,θ_S), its corresponding bounding segment ID (bsID)s for the two features are derived as follows:*

$$\begin{cases} bsID_T = i, if\ \theta_T \in \Delta_T^i \\ bsID_S = j, if\ \theta_S \in \Delta_S^j \end{cases} \quad (8)$$

where $bsID_T$ is the bsID for textual feature, $bsID_S$ is the bsID for spatial feature, Δ_T^i is the i-th subrange for the textual distance, and Δ_S^j is the j-th subrange for the spatial distance.

Definition 9 (BOUNDING BLOCK). *Given a JR, its corresponding bounding block is a rectangle for which at least one of the segment IDs for the two features is $bsID_T$ or $bsID_S$.*

Definition 10 (INNER BLOCK). *Given a JR, its corresponding inner block is a rectangle for which all the segment IDs for the two features are less than $bsID_T$ and $bsID_S$.*

Fig. 3(a) shows an example of the *DFDP* in which the two feature distances (i.e., *Dist* and *tMax*) are divided into four segmentations, respectively. Therefore, the 16 blocks are obtained in the *DFDP*. Given a *JR* represented by a grey rectangle, there exist six bounding blocks and six inner blocks.

Definition 11 (PIVOT BOUNDING BLOCK). *A pivot bounding block (PBB) is a bounding block whose BID_P is derived in Eq.(9):*

$$BID_P = \alpha \times bsID_T + bsID_S \quad (9)$$

where $bsID_T$ and $bsID_S$ are equal to that of in Eq.(8).

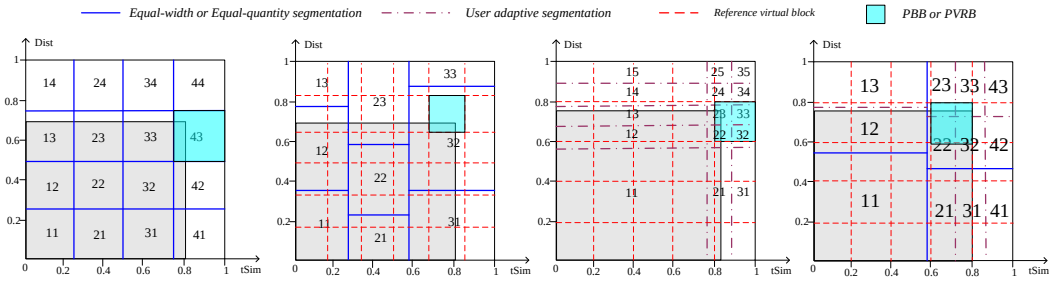
Based on Definition 11, the *PBB*s in **Fig. 3** are represented by the blue rectangles. Next, given a *JR*, we need to identify its corresponding affected inner and bounding blocks in terms of the four cases as described below.

- ***Equal-width segmentation scheme***

For the equal-width segmentation scheme, it is clear that the size of each block is equal. Based on Definition 9, in **Fig. 3(a)**, given a $JR(\theta_T, \theta_S)$, its corresponding *PBB* can be identified with the ID number defined in Eq.(9). So the affected inner blocks are 11, 21, 31, 12, 22, and 32, and the affected bounding blocks are 13, 23, 33, 43, 42, and 41.

- ***Equal-quantity segmentation scheme***

Different from the equal-width segmentation scheme, for the equal-quantity segmentation



(a). Equal-width-based (b). Equal-quantity-based (c). User adaptive-based (d). hybrid

Fig. 3. Four segmentation schemes

scheme, as shown in **Fig. 3(b)**, in most cases, the size of each block represented as blue line rectangle is different. So given a join rectangle represented by a grey rectangle in this figure, it is not trivial to efficiently identify the corresponding bounding blocks and inner ones, as the identification of the bounding blocks involves expensive distance computation especially for the large number of blocks.

To reduce the time cost, we propose a *virtual reference block* (VRB) approach in which the *DFDP* is equally partitioned into some blocks called VRBs represented by red dash rectangles in **Fig. 3(b)**. For each VRB, its corresponding candidate bounding blocks and inner ones are obtained previously. Given a *JR*, similar to the first case (i.e., equal-width), its *pivot virtual reference block* (PVRB) represented by a blue rectangle can be easily identified. As the corresponding candidate bounding and inner blocks of each PVRB has already been obtained in the preprocessing step, so for the PVRB in **Fig. 3(b)**, the bounding blocks become 12, 13, 23, 33, 32, and 31, respectively. The inner ones are 11, 22, and 21.

- ***User adaptive segmentation scheme***

For user adaptive segmentation, similar to the second case, given a *JR* in **Fig. 3(c)**, we use the above VRB approach to identify the affected bounding blocks (i.e., 12, 13, 14, 21, 22, 23, 24, 31, 32, 33, and 34) and inner one (i.e., 11).

- ***Hybrid segmentation scheme***

In the hybrid segmentation, as shown in **Fig. 3(d)**, assume that *DFDP* is first partitioned into some blocks called *parent blocks* using the equal-width or equal-quantity approach. Then, for each parent block, according to the user adaptive scheme, it can be further partitioned into some smaller blocks called *child blocks*.

Given a *JR*, the affected bounding child blocks are 12, 22, 32, 21, and 31, and the affected inner ones are 11.

Based on the above descriptions, for any *JR*, its corresponding bounding blocks and inner ones have been obtained. **Fig. 4** shows a block index structure composed of three levels: 1) *PBB/PVRB level*, 2) *Block level*, and 3) *Data level*.

PBB/PVRB level. For the *PBB/PVRB* level, as shown in the left part of **Fig. 4**, the ID numbers of the all *PBB/PVRB*s are indexed by a B-Tree. The leaf node in the B-Tree consists of two elements: <*PID*, *ptr*>, where *PID* refers to the pivot bounding block ID and *ptr* refers to a pointer that is pointed to the related inner and bounding blocks;

The block level. For the block level shown in the middle part of **Fig. 4**, for each *PBB/PVRB*, its affected blocks (i.e., *bounding blocks* and *inner blocks*) are previously identified and organized by a linked list. For the node in the linked list, it can be represented by a four-tuple *lnode*:= <*BID*, *ptr*, *flag*, *path*>, where *BID* is the ID number of the affected block shown in Eq.(7), *ptr* is a pointer to the next node, *flag*=1 means the node is a bounding block, else the inner one, and *path* refers to the file path of the corresponding block.

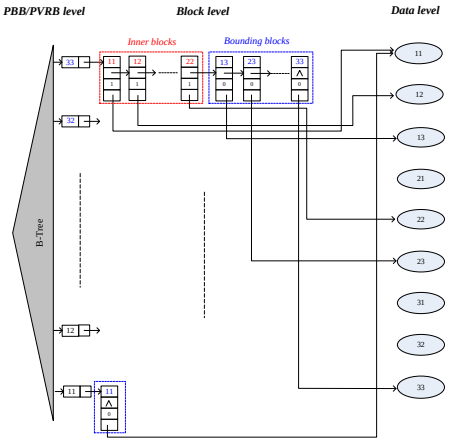


Fig. 4. An example of the BIST index structure

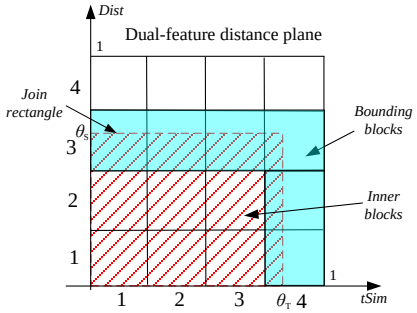


Fig. 5. Join processing

It is worth mentioning that the object pairs in the inner blocks can be part of the result object pairs, while the object pairs in the bounding ones need further refinement processing with the criteria of the thresholds.

The data level. For the data level shown in the right part of **Fig. 4**, the object pairs in each block are recorded and saved by a file whose filename is the block ID (i.e., *BID*). For the candidate object pairs in the affected inner blocks, they can be as part of the result object pairs without any refinement processing. But for the candidate object pairs in affected bounding blocks, the distances of all object pairs need to be sequentially calculated for the refinement processing to verify if they lay in the join block.

Algorithm 1 summarizes the *BIST* index construction procedure.

Algorithm 1. **Index construction**

Input: two object set: *R* and *S*;
Output: BIST index;
1. the *DFDP* is partitioned into blocks according to a segmentation scheme;
2. **for** each block/PVRB **do**
3. the block ID in Eq.(7) is inserted into a B-Tree;

```
4.  end for
5.  for each leaf node in the B-tree do
6.      its corresponding candidate blocks(i.e., bounding blocks and inner blocks) are
        organized by a linked list;
7.      for each node in the linked list do
8.          the corresponding object pairs in the block is pointed by this node;
9.      end for
10. end for
```

4.4 The Join Algorithm

Algorithm 2 summarizes the whole join processing with two steps: 1) *filtering* and 2) *refinement*.

- ***The filtering step***

In the filtering step, as shown in **Fig. 5**, given a join rectangle $JR(\theta_T, \theta_S)$ represented by a blue one, its corresponding inner and bounding blocks are first identified. The object pairs in the inner blocks represented by red grid rectangles in this figure are obtained as part of the result object ones.

- ***The refinement step***

Once the candidate result object pairs in the *bounding blocks* are obtained, in the refinement step, they need to be further calculated in terms of other feature(s) with the restriction of the corresponding thresholds. Finally, the answer object pairs are obtained.

Algorithm 2. Block Index-based ST-JOIN

```
Input: the join thresholds;
Output: the joined object pairs;
1. the PBB/PVRB is first identified according to the two thresholds which
   corresponds to the inner and bounding blocks;
2. the object pairs in the inner blocks are obtained as part of the result object
   ones;
3. for the object pairs in each bounding block do
4.     the similarity of each object pair is further calculated in terms of other
       feature(s) with the restriction of the corresponding thresholds.
5. end for
6. the final answer object pairs are obtained.
```

5. Experimental Evaluation

5.1 Experimental Setup

In this section, we present an extensive performance study on real and synthetic datasets to evaluate the efficiency of our proposed join methods.

The datasets in the experiment can be composed of two real datasets and one synthetic dataset, which are described below:

1) *The Flickr data set*. The first social image data set we used is from NUS-WIDE [28] in which 260k objects are downloaded from the Flickr.com. We used 100k objects in which their textual and spatial features are extracted.
2) *The Panoramio data set*. The second social image data set is from the Panoramio.com [29], which consists of about 50k geo-tagged objects. Similar to the first dataset, the two features of the geo-tagged objects are extracted.

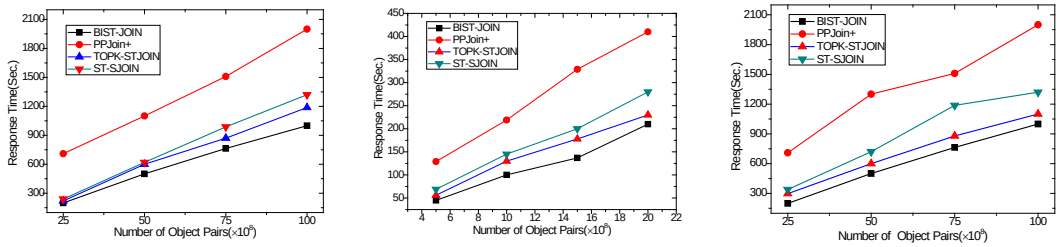
3) *Synthetic data set*. We generated 100k synthetic feature data. Different from the real datasets, the tag and spatial features are generated randomly which are uniformly distributed.

We have implemented the four join methods: *our BIST-JOIN approach* (see Section 4); *the PPJoin+ method* [14], *the ST-SJOIN method* [15], and *the TOPK-STJOIN method* [22] in the *C++* language. The index page size of B-tree is set to 4096 bytes. All the experiments are run on an Intel i5-2400 CPU processor at 3.10GHz with 4 gigabytes memory. In our evaluation, we use the total response time as the performance metric.

5.2 Effect of Data Size

The first experiment tests the effect of data size on the join efficiency. For each type of data set, the numbers of the object pairs are 100×10^8, 20×10^8, and 100×10^8, respectively. The segmentation granularity for the *BIST-JOIN* is $32\times32\times32$.

We have compared the four join methods. As shown in **Fig. 6**, with the increase of the number of object pairs, the total response time of the baseline approach is increasing exponentially since it involves an I/O and CPU intensive sequential scan of the object pairs. The other three methods increase linearly, and the *BIST-JOIN* method is superior to that of other three ones. Compared with the *BIST-JOIN*, the other three competitors (i.e., the *PP-Join*, the *ST-SJOIN* and *TOPK-STJOIN*) involve the refinements of the candidate object pairs which are I/O and CPU intensive. The *BIST-JOIN* method, however, can obtain the candidate object pairs directly without any intersection processing. It is interesting to notice that when the number of the object pairs is small, the *BIST-JOIN* is slightly better than the *TOPK-STJOIN*. As the number of the object pairs increases, however, the performance gap between the two methods becomes larger. This is because: 1) for the *TOPK-STJOIN*, when the number of the objects increases, the computation costs for searching the inverted list, matching the tokens and verification processing of the candidate object pairs are larger than the *BIST-JOIN*; 2) although the pruning capacity of the *TOPK-STJOIN* outperforms the state-of-the-art methods, the number of the candidate object pairs obtained by the *TOPK-STJOIN* may be larger than that of the candidate ones of the *BIST-JOIN* at a reasonable segmentation granularity (e.g., $32\times32\times32$).



(a). Flickr dataset (b). Panoramio dataset (c). Synthetic dataset

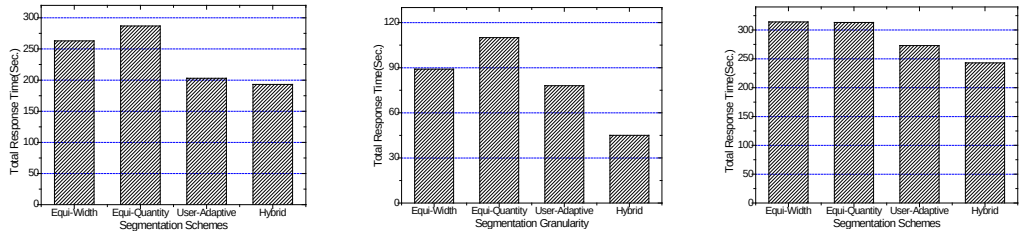
Fig. 6. Effect of data size

5.3 Effect of Segmentation Schemes

In this experiment, we compare the four segmentation schemes with a fixed segmentation granularity (e.g., $32\times32\times32$) on the join performance of the *BIST-JOIN* method by using the three datasets. The numbers of the object pairs for the three datasets are 60×10^8, 8×10^8, and 60×10^8, respectively. For the user-adaptive scheme, the number of the sampling users is 500.

In **Fig. 7**, the hybrid scheme is superior to the other three ones. The reason is that the hybrid one not only considers the object data distribution, but also the frequencies of the user

thresholds. So the distance computation costs of the object pairs in the bounding blocks are smaller than that of the other three ones. In addition, for the real dataset, the number of object pairs falling in the bounding blocks using the equal-width approach is smaller than that of the equal-quantity one due to the skewness of the data distribution. So the join cost of the equal-width is larger than the equal-quantity one accordingly. Moreover, the computation cost of the equal-width approach is much smaller than the equal-quantity one especially when the number of object pairs is large. This is because the equal-quantity approach involves data sorting processing in terms of different feature distances which is a CPU-intensive operation as well. Finally, it is interesting to notice that for the synthetic dataset, the join costs by using the equal-width and equal-quantity methods are approximately equal, since the number of object pairs in each block is almost equal for the uniform distribution of the synthetic data.



(a). Flickr dataset (b). Panoramio dataset (c). Synthetic dataset

Fig. 7. Effect of segment schemes on computation cost

5.4 Effect of Segmentation Granularity

In this experiment, we proceed to study the effect of the segmentation granularity for the *BIST-JOIN* method in which the segmentation granularities are $4\times4\times4$, $8\times8\times8$, $16\times16\times16$, $32\times32\times32$, $64\times64\times64$, and $128\times128\times128$, respectively. The numbers of the object pairs for the Flickr, the Panoramio and the synthetic datasets are 12×10^6, 6×10^6, and 12×10^6, respectively.

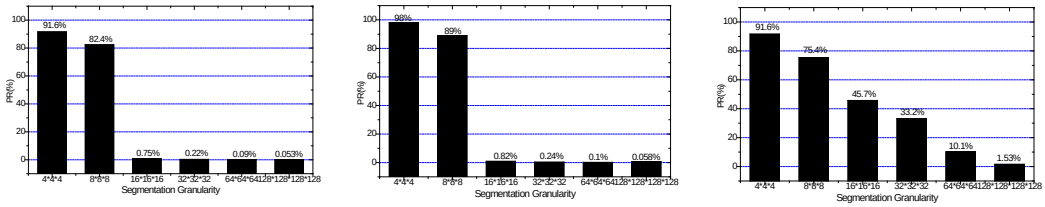
First of all, we test the effect of the segmentation granularity on the pruning ratio (*PR*). As illustrated in **Fig. 8**, the horizontal axis is the segmentation granularity, the vertical axis refers to the *PR*, which is defined in Eq.(10):

$$PR=\frac{NumB}{NumA} \tag{10}$$

where *NumA* refers to the total number of the object pairs in the *DFDP*, *NumB* refers to the total number of the object pairs in the bounding blocks.

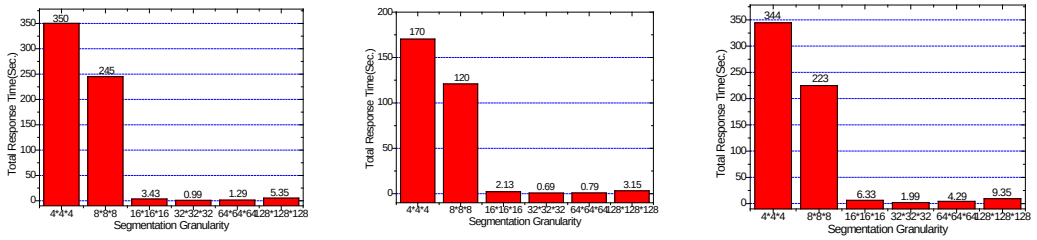
In **Fig. 8**, with the decrease of the segmentation granularities, the *PR* is slowing down dramatically for the real dataset. This is because: 1) the number of the object pairs in the bounding blocks is reduced when the granularity decreases; 2) the data distribution of the real object dataset is skew. When the granularity is beyond $8\times8\times8$, the *PR* reduces gradually. Compared with the two real datasets, for the synthetic dataset, the values of the *PR* are slowing down gradually due to the uniform data distribution.

Next, we proceed to evaluate the effect of the segmentation granularity on the response time. In **Fig. 9**, for the two real datasets, the response time are reducing first with the decrease of the granularity; but when the segmentation granularity is beyond $32\times32\times32$, the response time is gradually increasing. So the granularity $32\times32\times32$ is an optimal one by which the total response time is minimal. The reason behind it is that although the decreasing granularity results in the reduced number of object pairs in bounding blocks, accessing the inner and bounding blocks with increasing number incurs a loss of CPU and I/O costs.



(a). Flickr dataset (b). Panoramio dataset (c). Synthetic dataset

Fig. 8. Effect of segmentation granularity on the PR



(a). Flickr dataset (b). Panoramio dataset (c). Synthetic dataset

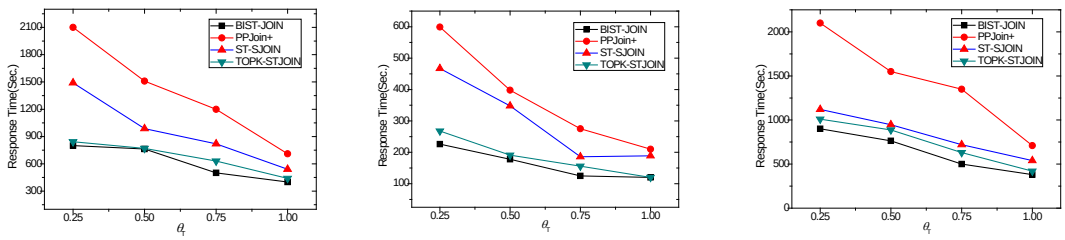
Fig. 9. Effect of segmentation granularity on the response time

5.5 Effect of Thresholds

The experiment tests the effect of thresholds on the join performance. For each type of data set, the numbers of the object pairs are 100×10^8, 20×10^8, and 100×10^8, respectively. The segmentation granularity for the *BIST-JOIN* is $32\times32\times32$.

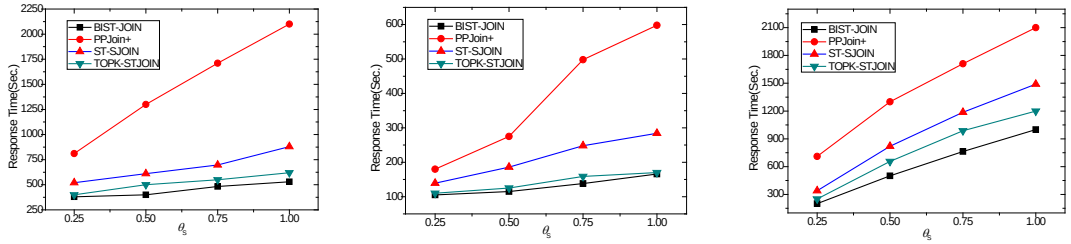
Fig. 10 shows that when θ_T increases, the response time of all methods decreases due to the reduction of the join results. The response time of *BIST-JOIN* and the *ST-SJOIN* methods decrease gradually, and the *BIST-JOIN* method is best among them. The reasons are similar to the above.

As demonstrated in **Fig. 11**, with the increase of θ_S, the join has more results and thus the response time of all methods also increases. We notice that, however, in case of the two real datasets, the time increase of the *TOPK-STJOIN* and the *BIST-JOIN* are smaller compared to the synthetic dataset. This is because the spatial feature is not a dominating one compared with the textual one; the *TOPK-STJOIN* and the *BIST-JOIN* on the real datasets are very close to a *textual* similarity join, since the join performance of the *BIST-JOIN* on the two real data sets are not dominantly affected by θ_T. Therefore, the threshold θ_T is more important than θ_S for the real datasets.



(a). Flickr dataset, θ_S=0.2 (b). Panoramio dataset, θ_S =0.2 (c). Synthetic dataset, θ_S=0.2

Fig. 10. Varying the textual similarity threshold θ_T



(a). Flickr dataset, θ_T=0.3 (b). Panoramio dataset, θ_T=0.3 (c). Synthetic dataset, θ_T=0.3

Fig. 11. Varying the spatial distance threshold θ_S

6. Conclusions and Future Work

In this paper, we proposed an index support spatio-textual similarity join (*ST-SJOIN*) method called the *BIST-JOIN*. Finally, the extensive experiments with real-life and synthetic datasets demonstrate the efficiency of our proposed method outperforms that of the state-of-the-art methods.

In our future work, we will focus on the study of arbitrary-feature-based similarity join and its indexing scheme.

References

[1] A-X. Sun, S. S. Bhowmick, K. Tran Nam Nguyen, G. Bai, "Tag-Based Social Image Retrieval: An Empirical Evaluation," *Journal of the American Society for Information Science and Technology* (JASIST), vol.62, no.12, pp. 2364–2381, 2011. Article (CrossRef Link).

[2] X-R. Li, Cees G. M. Snoek, M. Worring, A. W. M. Smeulders, "Harvesting Social Objects for Bi-Concept Search," *IEEE Transactions on Multimedia*, vol.14, no.4, pp. 1091-1104, 2012. Article (CrossRef Link).

[3] A. Arasu, V. Ganti, and R. Kaushik, "Efficient exact set-similarity joins," in *Proc. of VLDB*, pp. 918-929, 2006. Article (CrossRef Link).

[4] J. Ballesteros, A. Cary, and N. Rishe, "Spsjoin: parallel spatial similarity joins," in *Proc. of* GIS, pp. 481– 484, 2011. Article (CrossRef Link).

[5] R. J. Bayardo, Y. Ma, and R. Srikant, "Scaling up all pairs similarity search," in *Proc. of WWW*, 2007. Article (CrossRef Link).

[6] T. Brinkhoff, H.-P. Kriegel, and B. Seeger, "Efficient processing of spatial joins using r-trees," in *Proc. of SIGMOD Conference*, 1993. Article (CrossRef Link).

[7] E. P. F. Chan. "Buffer queries," *IEEE Transactions on Knowledge and Data Engineering*, vol.15, no.4, pp.895–910, 2003. Article (CrossRef Link).

[8] S. Chaudhuri, V. Ganti, and R. Kaushik, "A primitive operator for similarity joins in data cleaning," in *Proc. of ICDE*, 2006. Article (CrossRef Link).

[9] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios, "Duplicate record detection: A survey," *IEEE Transactions on Knowledge and Data Engineering*, vol.19, no.1, pp.1–16, 2007. Article (CrossRef Link).

[10] L. Gravano, P. G. Ipeirotis, H. V. Jagadish, N. Koudas, S. Muthukrishnan, and D. Srivastava, "Approximate string joins in a database (almost) for free," in *Proc. of VLDB*, pp.491-500, 2001. Article (CrossRef Link).

[11] S. Sarawagi and A. Kirpal, "Efficient set joins on similarity predicates," in *Proc. of SIGMOD*, 2004. Article (CrossRef Link).
[12] C. Xiao, W. Wang, X. Lin, and H. Shang, "Top-k set similarity joins," in *Proc. of ICDE*, 2009. Article (CrossRef Link).
[13] C. Xiao, W. Wang, X. Lin, and J. X. Yu, "Efficient similarity joins for near duplicate detection," in *Proc. of WWW*, 2008. Article (CrossRef Link).
[14] C. Xiao, W. Wang, X. Lin, J. X. Yu, and G. Wang, "Efficient similarity joins for near-duplicate detection," *ACM Transactions on Database Systems*, vol. 36, no.3, 15, 2011. Article (CrossRef Link).
[15] P. Bouros, S. Ge, and N. Mamoulis, "Spatio-Textual Similarity Joins," in *Proc. of VLDB* 2013. Article (CrossRef Link).
[16] R. Baeza-Yates and B. Ribeiro-Neto, "Modern Information Retrieval. Addison Wesley," *1st edition edition*, May 1999. Article (CrossRef Link).
[17] A. Z. Broder, "On the resemblance and containment of documents," in *Proc. of SEQS*, 1997. Article (CrossRef Link).
[18] M. Charikar, "Similarity estimation techniques from rounding algorithms," in *Proc. of STOC*, 2002. Article (CrossRef Link).
[19] S. Chaudhuri, V. Ganti, and R. Kaushik, "A primitive operator for similarity joins in data cleaning," in *Proc. of ICDE*, 2006. Article (CrossRef Link).
[20] A. Chowdhury, O. Frieder, D. A. Grossman, and M. C. McCabe, "Collection statistics for fast duplicate document detection," *ACM Transactions on Information Systems*, vol. 20, no. 2, pp.171 –191, 2002. Article (CrossRef Link).
[21] D. Deng, G-L. Li, J-H. Feng, "A Pivotal Prefix Based Filtering Algorithm for String Similarity Search," in *Proc. of SIGMOD* 2014, 673-684, 2014. Article (CrossRef Link).
[22] H-Q. Hu, G-L. Li, Z-F. Bao, J-H. Feng, Z-G. Gong, "Topk Spatio-Textual Similarity Join," *IEEE Transactions on Knowledge and Data Engineering*, 2015. Article (CrossRef Link).
[23] G-L. Li, J. He, D. Dong, J. Li, J-H. Feng, "Efficient Similarity Search and Join on Multi-Attribute Data," in *Proc. of SIGMOD* 2015, 2015. Article (CrossRef Link).
[24] W-B. Tao, M-H. Yu, G-L. Li, "Efficient Top-K SimRank-based Similarity Join," in *Proc. of VLDB* 2015. Article (CrossRef Link).
[25] D. Deng, G-L. Li, H. Wen, J-H. Feng, "An Efficient Partition Based Method for Exact Set Similarity Joins," in *Proc. of VLDB*, 2016. Article (CrossRef Link).
[26] Z-Y. Shang, Y-X. Liu, G-L. Li and J-H. Feng, "K-Join: Knowledge-Aware Similarity Join," in *Proc. of ICDE* 2017. Article (CrossRef Link).
[27] N. Ta, G-L. Li, J-H. Feng, "Signature-Based Trajectory Similarity Join," *IEEE Transactions on Knowledge and Data Engineering*, 2017. Article (CrossRef Link).
[28] http://lms.comp.nus.edu.sg/research/NUS-WIDE.htm
[29] http://www.panoramio.com/

Dr. Yiming Xiang is currently an associate professor at the College of Management & e-Business in Zhejiang Gongshang University. He obtained his Ph.D. degree from Zhejiang University in 2015. His research interests focus on information processing and management.

Dr. Yi Zhuang is a recipient of the CCF Doctoral Dissertation Award conferred by Chinese Computer Federation in 2008 and IBM Ph.D. Fellowship 2007–2008. He is currently a full professor at the College of Computer & Information Engineering in Zhejiang Gongshang University where he joined as faculty member since May 2008. He obtained his Ph.D. degree in computer science from Zhejiang University in March 2008. From January 2008 to March 2008, supported by IBM Ph.D. Fellowship, Dr. Zhuang has spent 3 months to participate in the study of an optimal hybrid storage model based on DB2 as a research intern in IBM China Research Lab. His research interests mainly focus on multimedia database and cloud computing. He has published 40+ papers in the leading journals and conferences, i.e., ACM TOIT, ACM TALIP, Information Sciences, KAIS, MMSJ, MATP, and Expert System with Applications, etc.

Nan Jiang received the bachelor degree of medical science and the master degree of medical science both from the Zhejiang University in 2004 and 2007, respectively. Her research interests mainly focus on medical image processing. She has published 10+ papers in international conferences and journals.