SPATIO-TEXTUAL SIMILARITY JOIN

Ch Shylaja and Supreethi K.P

Department of Computer Engineering,
Jawaharlal Nehru Technological University, Hyderabad, India

ABSTRACT

Data mining is the process of discovering interesting patterns and knowledge from large amounts of data. Spatial databases store large space related data, such as maps, preprocessed remote sensing or medical imaging data.

Modern mobile phones and mobile devices are equipped with GPS devices; this is the reason for the Location based services to gain significant attention. These Location based services generate large amounts of spatio- textual data which contain both spatial location and textual description. The spatio-textual objects have different representations because of deviations in GPS or due to different user descriptions. This calls for the need of efficient methods to integrate spatio-textual data. Spatio-textual similarity join meets this need. Spatio-textual similarity join: Given two sets of spatio-textual data, it finds all the similar pairs. Filter and refine framework will be developed to device the algorithms. The prefix filter technique will be extended to generate spatial and textual signatures and inverted indexes will be built on top of these signatures. Candidate pairs will be found using these indexes. Finally the candidate pairs will be refined to get the result. MBR-prefix based signature will be used to prune dissimilar objects. Hybrid signature will be used to support spatial and textual pruning simultaneously.

KEYWORDS

Spatio-textual similarity join, inverted index, candidate pairs, hybrid signature.

1. INTRODUCTION

Location based services have gained significant attention because of the omnipresence of GPS in smart phones and mobile devices. LBS are general class of computer program level services that use location data to control features. LBS are used in a variety of contexts such as health, indoor objects search, entertainment, work, personal life etc.

LBS generate large amounts of spatio-textual data which contain both geographical location and textual description. The spatio-textual objects may have different representations. The difference textual representation may be because of deviations in GPS and different user description. Spatio-textual similarity join correlates the spatio-textual data from different sources. Given two input collections of sets, similarity join identifies all pairs of sets, one from each collection that has high similarity.

Similarity join is an important operation for reconciling different representations of an entity. Two objects are said to be similar if their spatial and textual similarity is greater than the given threshold. There are various ways to quantify the spatial similarity and textual similarity. As an example of a spatial join, consider one data set describing parking lots and other describing movie theaters of a city, using the predicate ‘next to’, a spatial join between these data sets will provide an answer to the query: “find all movie theaters that are adjacent to a parking lot”.

Consider two collections of objects R = {r1, r2 . . . , rn} and S = {s1, s2 . . . , sm}. Each object r (or s) includes a spatial region Mr and textual description Tr. In this paper we use Minimum Bounding Rectangle (MBR) to capture the spatial information, denoted by Mr = [rbl, rtr], where rbl = (rbl.x, rbl.y) is the bottom-left point and rtr = (rtr.x, rtr.y) is the top-right point. We use a set of tokens to capture the textual description, denoted by Tr = {t1, t2, . . . , tv}, which describes an object (e.g., {Hotel, Pizza, Subway}) or users’ interests (e.g., {Seaside, Sandwich, Delivery}). As tokens may have different importance, we assign each token ti with a weight w (ti) (e.g., inverse document frequency idf). To quantify the similarity between two objects, we use the well-known Jaccard as an example to evaluate the spatial similarity (SJac) and textual similarity (TJac). Our techniques can be easily extended to support other similarity functions.

Definition 1 (Spatial Jaccard). Given two objects r and s, their spatial Jaccard similarity (Sjac) is defined as:

$$S_{Jac}(r,s) = \frac{|M_r| \cap |M_s|}{|M_r| + |M_s| - |M_r \cap M_s|} \;\ldots\ldots\; (1)$$

Where | · | is the size of an MBR.

Definition 2 (Textual Jaccard). Given two objects r and s, their textual Jaccard similarity (Tjac) is defined as:

$$T_{Jac}(r,s) = \frac{\sum_{t \in T_r \cap T_s} w(t)}{\sum_{t \in T_r \cup T_s} w(t)} \;\ldots\ldots\; (2)$$

Where w (t) is the weight of token t.

Two objects r and s are similar if they satisfy (1) Spatial constraint: their spatial Jaccard similarity is larger than a spatial similarity threshold τs, i.e., SJac(r, s) > τs; and
(2)Textual constraint: their textual Jaccard similarity is larger than a textual similarity threshold τt, i.e., TJac(r, s) > τt.

2. RELATED WORK

Previous methods for spatial similarity join and textual similarity join are available however spatio-textual similarity join are not. The following are some of the previous methods used for spatial join and textual join.

2.1. Set-similarity joins

Given two input collections of sets, a set-similarity join (SSJoin) [2] identifies all pairs of sets, one from each collection, that have high similarity. Recent work has identified SSJoin as a useful primitive operator in data cleaning. SSJoin algorithms have two important features: They are exact, i.e., they always produce the correct answer, and they carry precise performance guarantees. A general-purpose data cleaning system is faced with the daunting task of supporting a large number of similarity joins with different similarity functions. Recent work [3] has identified set-similarity join (SSJoin) as a powerful primitive for supporting (string-) similarity joins involving many common similarity functions. This algorithm can be implemented on top of a regular DBMS with a very little coding effort.

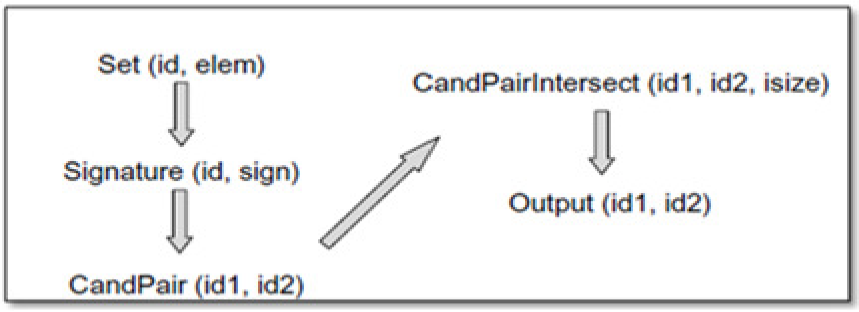

Figure 1: Jaccard SSJoin implementation

2.2. Spatial joins using R-trees

Spatial joins are one of the most important operations for combining spatial objects of several relations. R-trees are very suitable for supporting spatial queries and the R*-tree is one of the most efficient members of the R-tree family [4] . In order to support spatial queries such as “find all objects which intersect a given window”, spatial access methods cluster objects on disks according to their spatial location in space. Consequently, if objects are close together in space, they are stored close together on disk with high probability. An R-tree[5] is a B+-tree like access method that stores multidimensional rectangles as complete objects without clipping them or transforming them to higher dimensional points. An example for an R-tree is given in Figure 1. The tree consists of three data pages and a directory page. Note, that the rectangles of the directory page are the minimum bounding rectangles of those rectangles that are stored in the corresponding child node. The query window is depicted by the gray colored rectangle. First, the query is performed against the root of the R-tree where rectangle r and t intersect the window. Thus, the corresponding two data pages are read into memory and their entries are checked for a common intersection with the window. Eventually, rectangle al is found to be an answer of the window query.

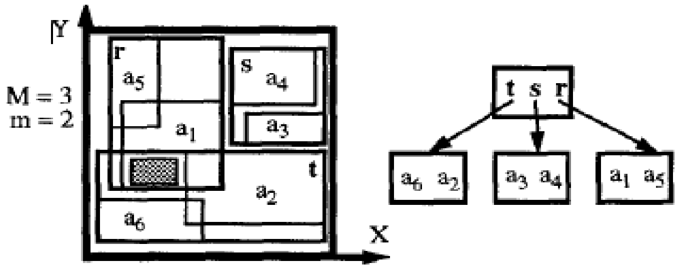

Figure 2: example of an R-tree

2.3. Spatial hash joins

Spatial hash-joins [6] shows how to apply hash-join paradigm to spatial joins, and define a new frame work for spatial hash joins. Spatial partition functions [13] have two components: a set of bucket extents and an assignment function, which may map a data item into multiple buckets. Furthermore, the partition functions for the two input datasets may be different. Relational hash joins yield excellent performance, particularly for relations that are large compared to buffer sizes. Like relational hash-joins, our method partitions its input into buckets during the partition phase and then joins the buckets to produce join results in the join phase. However, unlike relational joins, a partition function in our framework comprises two components: a set of bucket extents and an assignment function. The assignment function may map a data item into multiple buckets, and the partition functions for the two datasets may differ.

2.4. Size separation spatial join

S3 join [7] introduces a new algorithm to compute the spatial join of two or more spatial data sets, when indexes are not available on them. Size Separation Spatial Join (S3 J), is a generalization of the relational Sort Merge Join algorithm. S3 J is designed so that no replication of the spatial entities is necessary, whereas previous approaches have required replication. The algorithm does not rely on statistical information from the data sets involved to efficiently perform the join and for a range of distributions offers a guaranteed worst case performance independent of the spatial statistics of the data sets.

2.5. SEAL: Spatio-textual similarity search:

Many modern LBS applications generate a new kind of spatio-textual data, regions-of-interest (ROIs), containing region-based spatial information and textual description. To satisfy search requirements on ROIs, SEAL [8] introduces a new research problem, called spatio-textual similarity search: Given a set of ROIs and a query ROI, SEAL finds the ROIs which are similar to the query by considering spatial overlap and textual similarity. Spatio-textual similarity search can satisfy users' information needs in various real applications. A filter and verification framework is proposed to prune dissimilar objects and visit small amount of objects that may be similar to the given query.

2.6. Partition based spatial merge join

PBSM (Partition Based Spatial–Merge) [10] is an algorithm for performing spatial join operation. This algorithm is especially effective when neither of the inputs to the join have an index on the joining attribute. Such a situation could arise if both inputs to the join are intermediate results in a complex query or in a parallel environment where the inputs must be dynamically redistributed. The PBSM algorithm partitions the inputs into manageable chunks, and joins them using a computational geometry based plane-sweeping technique.

There are various partitioning strategies for spatio-textual similarity join [14]. One approach is to start with a spatial data structure (either grids or quad trees), traverse regions and apply a below algorithms for identifying similar pairs of textual documents called All-Pairs [11].This is called the local approach. An alternative approach is to construct a global index but partition postings spatially (either by grid or by quadtree, linearized by z-ordering) and modify the All-Pairs algorithm to prune candidates based on distance. This is called as the global approach. Together, this yields four combinations: local grid, local quadtree, global grid, and global quadtree.

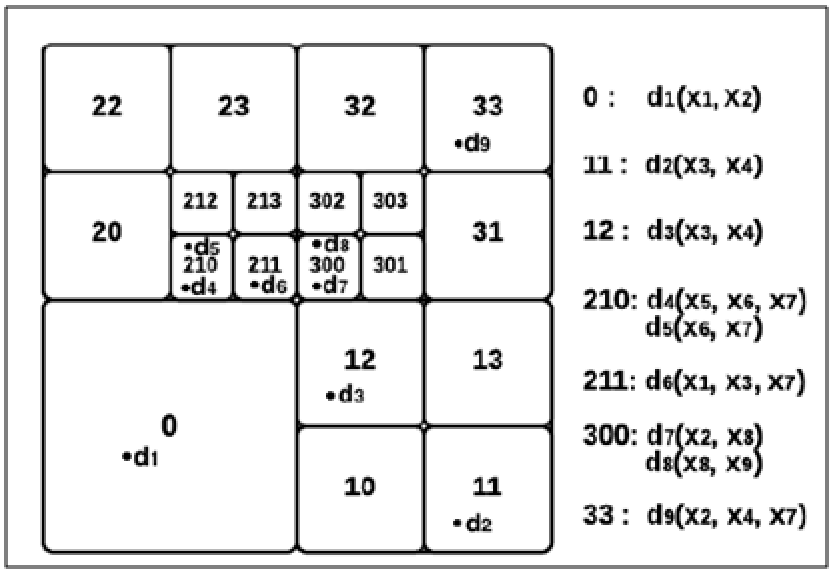

Figure 3: QuadTree Example

The following are the different similarity functions that are been traditionally used in similarity joins. They are Jaccard similarity, Cosine similarity, Edit distance, generalized edit distance. It is well known that no single similarity function is universally applicable across all domains and scenarios.

2.7. Jaccard similarity

Jaccard coefficient [9] is a commonly used measure of overlap of two sets A and B.
Jaccard (A, B) = $\frac{|A\cap B|}{|A\cup B|}$; Jaccard (A, A) =1; Jaccard (A, B) = 0 if A∩B = 0; A and B do not have to be of the same size. Jaccard similarity always assigns a number between 0 and 1. The issues with Jaccard similarity are (1) It does not consider term frequency (2) Rare terms in a collection are more informative than frequent terms.

2.8. Cosine similarity

Cosine similarity measures the angle between two vectors and is calculated by dividing the inner product of two vectors by multiplication of vectors' length. The cosine similarity between documents di and dj is formulated as follows:

$$\mathrm{sim}_{\cos}(d_i, d_j) = \frac{\vec{d_i}\cdot\vec{d_j}}{\|\vec{d_i}\|\cdot\|\vec{d_j}\|} = \frac{\sum_{k=1}^{n} w_{ik}\times w_{jk}}{\sqrt{\sum_{k=1}^{n} w^2{}_{ik}}\times\sqrt{\sum_{k=1}^{n} w^2{}_{jk}}} \quad \ldots\ldots..(3)$$

The cosine similarity values range between (0, 1), where a cosine similarity value of 0 means that the documents are unrelated and a cosine similarity value close to 1 means that the documents are closely related. It is obvious, that in order to have cosine similarity greater than 0, documents should have some words in common. When all of the words and their associated weights are same in two different documents, then the cosine similarity between these two documents is equal to 1.

2.9. Edit distance

Edit distance measures the minimum number of edit operations (insertion, deletion, and substitution) to transform one string to another. Edit distance [3] has two distinctive advantages over alternative distance or similarity measure: (a) it reflects the ordering of tokens in the string; and (b) it allows non-trivial alignment. These properties make edit distance a good measure in many application domains, e.g., to capture typographical errors for text documents, and to capture similarities for Homologous proteins or genes. Similarity joins with edit distance thresholds poses serious algorithmic challenges. Computing edit distance is more expensive (O(n2)) than alternative distance or similarity measure (usually O(n)).

3. PROPOSED SYSTEM

A mark based channel and-refine structure will be built. In the first place spatial and textual marks for the items will be created and altered lists will be manufactured to evade excess reckonings. At that point, these marks will be utilized to discover hopeful sets whose marks are comparative enough. A few calculations will be proposed by arranging spatial and text based marks in diverse ways. Also, two viable systems will be proposed to produce top notch marks. The first chooses a subregion as a signature for each one article. The subregion chose in this technique is minimized. The second technique is a mixture signature which incorporates spatial data and literary portrayals. Additionally compelling pruning methods to enhance the execution will be created.

1) A channel and refine system will be investigated and propose effective calculations which can prune vast quantities of different objects

.

2) MBR-Prefix based mark which utilizes subregions of articles as marks to backing spatial pruning will be added and subregions will be minimized.

3) a cross breed signature will be proposed by incorporating spatial furthermore text based pruning. To streamline the half and half signature, we display the issue as a token part issue and demonstrate it is NP-complete.

Given two sets of spatio-textual objects, signature based similarity join method is used to find the similar objects from the given sets. For this, a signature based filter and refine framework will be developed. First spatial and textual signatures will be developed. Inverted indexes will be built on top of these signatures. Then candidate pairs will be generated using these inverted indexes. Finally the candidate pairs will be refined to get the final result.

3.1 Generating spatial signature:

Prefix filtering technique can be utilized to generate spatial signatures as follows.
Given an object r, girds in G_r will be sorted first. Then signature for r can be developed by deleting last k grids which satisfy $\sum_{i=|G_r|-k+1}^{|G_r|}|M_{gi}| \cap |M_r| \leq \tau_s \cdot |M_r|$ and $\sum_{i=|G_r|-k}^{|G_r|}|M_{gi}| \cap |M_r| > \tau_s \cdot |M_r|$, denoted by $SIG_S(r)$. If r and s are similar, then they at least share one common grid in their spatial signatures.

3.2 Generating textual signature:

The basic idea is that if the similarity of two token sets is larger than a given threshold, they should share enough common tokens. Then the index can be reduced by cutting off some common tokens. For simplicity, suppose *w(ti)* = 1. Consider two token sets Tr and *Ts*. Since $|Tr \cap Ts|$ is an integer, according to [6], if TJac(Tr, Ts) > τt,
then $|T_r \cap T_s| > \tau_t \times |T_r| \geq \lfloor \tau_t \times |T_r| \rfloor + 1$ | Based on this property, we first sort all the tokens according to a global ordering, e.g., idf. Then for each token set T, we generate its prefix by deleting last $\lfloor \tau_t \times |T_r| \rfloor$ tokens. If any two objects have common tokens in their prefixes, we add them into the candidate set. Next we extend the prefix filter technique to the case *w(ti)*≠ 1 (the weight may not be an integer). We first sort tokens in the descending order of their weights. Since $|T_r \cap T_s| > \tau_t \times \sum_{i=1}^{|T_r|} w(t_i)$ then for each token set T, we generate its prefix by deleting the last *k* tokens which satisfy
$\sum_{i=|T|-k+1}^{|T|} w(t_i) \leq \tau_t \times \sum_{i=1}^{|T|} w(t_i)$ and $\sum_{i=|T|-k}^{|T|} w(t_i) > \tau_t \times \sum_{i=1}^{|T|} w(t_i)$

3.3 Generating inverted indexes and candidate pairs

Inverted indexes will be developed on top of these signatures. The generated signatures of the object pairs will be used to get the candidate pairs. The objects will be scanned sequentially and an inverted index for the signatures of the visited objects will be maintained. These inverted indexes will be used to get the candidate pairs. For each grid $g \in SIG_s(r)$, is used to probe the corresponding spatial inverted index and add all the objects in these lists to spatial candidate set C_s. Meanwhile, for each token $t \in SIG_T(r)$, corresponding textual inverted list will be scanned and objects to textual candidate set C_T will be added. Then the intersection of C_S and C_T will be taken as the candidates.

3.4 Algorithms used

Algorithm 1:

Input: R: An object set; τ_s: Spatial threshold; τ_t: Textual threshold
Output: P: a set of similar pairs

```
1. Begin
2. I ← empty index;
3. C ← empty candidate set;
4. For each object r ∈ R do
5.         SIG(r) ← SigGeneration(r, $\tau_s$, $\tau_t$);
6.         C← CandidateFiltering(SIG(r), I);
7.         P ← Refinement(C);
8. End
```

Algorithm 1includes three steps namely filter, Index update and refine. This algorithm can be used to avoid enumerating every object pair.

Algorithm 2:

```
INPUT: Set collections R and S and threshold $\gamma$

1. Begin
2. For each r $\in$ R, generate signature-set Sign(r)
3. For each s $\in$ S, generate signature-set Sign(s)
4. Generate all candidate pairs (r, s), r $\in$ R, and s $\in$ S satisfying Sign(r) $\cap$ Sign $\neq \varphi$
5. Output any candidate pair (r, s) satisfying Sim(r, s) $\geq \gamma$.
6. End
```

Algorithm 2 is used to generate candidate pairs of the signatures build on objects.

4. IMPLEMENTATION

Observe that the form of predicate that is considered here involves multi-set intersection when any R.A (or S.A) group contains multiple values on the R.B attributes. In order to be able to implement them using standard relational operators, these multi-sets will be converted into sets; each value in R:B and S:B will be converted into an ordered pair containing an ordinal number to distinguish it from its duplicates. Thus, for example, the multi-set {1, 1, 2} would be converted to {<1, 1>, <1, 2>, <1, 3>} Since set intersections can be implemented using joins, the conversion enables to perform multi-set intersections using joins.

The two sets of spatio-textual objects that are considered are (1) The twitter users, modeled as spatio-textual objects (2) message set which consists of advertisements. The spatio-textual similarity join is performed between these two sets.

Twitter allows its users to upload their location along with posting tweets. Hence these users are be modeled as spatio-textual objects. Meanwhile twitter messages like advertisements are also modeled as spatio-textual objects. To deliver messages to relevant users, spatio-textual similarity join is used.

The twitter user tweets are collected by creating a widget of respective user. These tweets are parsed and unique terms are identified. The user uploads his location along with posting tweets. The message set consists of advertisements with their spatial location and textual description about the advertisements is considered. Spatio-textual similarity is applied for the spatial data and textual data in both the sets and similar object pairs are found. Based on the similar object pairs the messages are delivered to the relevant users.

To implement this, the following system specifications are required: The hardware requirements are 512MB DD RAM, 40GB hard disk, Pentium IV 2.4 GHz processor and the software requirements are Window7 Operating System, NetBeans 7.4 IDE, JAVA, MySql Database, Apache Tomcat server, TWITTER API and Google API.

5. RESULTS

The objects in the advertisement set that are spatially close to the user and textually similar to the user's tweets are recommended advertisements to the user. These advertisements will be delivered to the user.

6. CONCLUSION

A new research problem called spatio-textual similarity join will be studied in this paper. A prefix-filter based framework will be deviced and several possible solutions will be proposed. To achieve higher performance, an MBRPrefix based filtering technique will be developed which can prune large number of dissimilar objects. Hybrid signature integrating spatial information and textual descriptions will be proposed. A token partition problem will be modeled Jaccard similarity will be used to prune spatial data and Cosine similarity will be used to prune textual data.

REFERENCES

[1] J.Han, M.Kamber, and J.Pei, Data Mining. Waltham, MA: USA, 2012.

[2] A. Arasu, V. Ganti, and R. Kaushik, “Efficient Exact Set-Similarity Joins,” Proc. 32nd Int. Conf. Very Large Data Bases, pp. 918–929, 2006.

[3] S. Chaudhuri, V. Ganti, and R. Kaushik, “A primitive operator for similarity joins in data cleaning,” Proc. - Int. Conf. Data Eng., vol. 2006, p. 5, 2006.

[4] T. Brinkhoff, H.-P. Kriegel, and B. Seeger, “Hans-Peter of Computer Brinkhoff Kriegel 1 Introductiort,” Computer (Long. Beach. Calif)., pp. 237–246, 1993.

[5] A. Guttman, “R-trees: A Dynamic Index Structure for Spatial Searching,” Proc. 1984 ACM SIGMOD Int. Conf. Manag. Data - SIGMOD ’84, pp. 47–57, 1984.

[6] M.-L. Lo and C. V. Ravishankar, “Spatial hash-joins,” ACM SIGMOD Rec., vol. 25, pp. 247–258, 1996.

[7] N. Koudas and K. C. Sevcik, “Size separation spatial join,” ACM SIGMOD Rec., vol. 26, pp. 324–335, 1997.

[8] J. Fan, G. Li, L. Zhou, S. Chen, and J. Hu, “SEAL : Spatio-Textual Similarity Search,” Proc. VLDB Endow., vol. 5, pp. 824–835, 2012.

[9] S. Niwattanakul, J. Singthongchai, E. Naenudorn, and S. Wanapu, “Using of Jaccard Coefficient for Keywords Similarity,” Int. MultiConference Eng. Comput. Sci., vol. I, pp. 380–384, 2013.

[10] Jignesh M, Patel David J. DeWitt, "Partition Based Spatial Merge Join", ACM SIGMOD vol. 25 pp 259-270, 1996.

[11] R. Bayardo, Yiming Ma, Ramakrishnan Srikant,"Scaling Up All Pair Similarity Search", ACM , pp 131-140, 2007

[12] Sitong Liu, Guoliang Li, and Jianhua Feng,"Prefix Filter Based Method For Spatio-Textual Similarity Join",IEEE, vol 26, Issue 10, pp 2354-2367, 2013.

[13] M. Kitsuregawa, H. Tanaka, and T. Moto-Oka, “Application of hash to data base machine and its architecture, ” New Generation Computing, vol. 1, no. 1, pp. 66-74, 1983

[14] Jinfeng Rao, Jimmy Lin, and Hanan Samet,"Partition Strategies for Spatio Textual Similarity Join", ACM SIGSPATIAL InternationalWorkshop on Analytics for Big Geospatial Data (BigSpatial) 2014 ISBN 978-1-4503-3132-6.