Scalable Algorithms for Spatial-Textual Data Join

Vivek Gupta

Computer Science

Indraprastha Institute of Information Technology, Delhi (IIIT-D), India

A Thesis Report submitted in partial fulfilment for the degree of

MTech Computer Science

14 May 2015

1.: Prof. Vikram Goyal (Thesis Adviser)

2. Prof. Chetan Arora (Internal Examiner)

2. Dr. Dhaval Patel (External Examiner)

Day of the defense: 14 May 2015

Signature from Post-Graduate Committee (PGC) Chair:

Abstract

Spatio-textual similarity join retrieves a set of pairs of objects wherein objects in each pair are close in spatial as well as textual dimensions. A lot of work has been done in the spatial dimension but no work has been done for spatial-textual joins. However, due to the ubiquity of GPS enabled devices, huge spatial-textual data is being generated which demand new methods to query and perform operations on this new data type. We study join operation for spatial-textual data and incorporate various optimizations/ heuristics such as efficient grid partitioning for spatial dimension, use of a specific prefix length of textual vector and ordering of elements in textual vectors on the basis of their TF-IDF scores. We also design and study algorithms using the above heuristics for spatial-textual data join on MapReduce Framework. Experimental results on two real life datasets, Flickr and Foursquare, show the effectiveness of these optimizations in terms of computation time as well as pruning of non-candidates.

I dedicate my MTech Thesis to my *parents* for their never-ending love, support and blessings. Their kind support in ever moving forward in academic ventures made it possible for me to pursue my M. Tech. I am blessed to have parents like them.

Acknowledgements

I wish to express my sincere gratitude to Prof. Pankaj Jalote, Director at Indraprastha Institute of Information Technology, for providing me an opportunity to do my M.Tech Thesis.

I sincerely thank my Thesis Supervisor Dr. Vikram Goyal, Assistant Professor at Indraprastha Institute of Information Technology, Delhi for his guidance, encouragement and continuous support in carrying out this thesis work. He is one of the best persons, and great mentor. I will always be thankful to him.

I would also like to thank Dr. Dhaval Patel, professor at IIT Roorkee for devoting his time in discussing ideas with me, evaluating my thesis, and helping me at every stage of thesis.

I would like to extend my thanks to Dr. Chetan Arora for evaluating my thesis as internal examiner.

I would also like to thank all people with whom I had many interesting research discussions. I am grateful to the entire computer science department for their words of appreciation and the faith they have shown in me. Many thanks to all my friends for their challenging distractions in my spare time.

Last, but not least, I wish to avail ourselves of this opportunity, express a sense of gratitude and love to my friends and my beloved parents for their manual support, understanding, and simply for everything.

Declaration

This is to certify that the MTech Thesis Report titled **Scalable Algorithms for Spatial-Textual Data Join** submitted by **Vivek Gupta** for the partial fulfillment of the requirements for the degree of *MTech* in *Computer Science* is a record of the bonafide work carried out by him under my guidance and supervision at Indraprastha Institute of Information Technology, Delhi. This work has not been submitted anywhere else for the reward of any other degree.

Professor Vikram Goyal
Indraprastha Institute of Information Technology, New Delhi

Contents

List of Figures

List of Tables

1

Motivation and Introduction

With the proliferation of GPS and Internet technology, the web is now acquiring a spatial dimension along with its prior textual dimension. Nowadays, a lot of spatial textual data is being generated by social web applications like Foursquare, Flicker, etc. which provides opportunities to use these data for use in different applications. However, it also demands for development of efficient techniques to manage and perform various operations over the spatial-textual data. One of the frequently required operations is a join operation over spatial-textual data that can be used to answer spatial-textual similarity queries.

Recently, there is a upsurge in the work on spatio-textual similarity join [1, 2, 3] and different spatio-textual queries [2, 4, 5, 6, 7, 8, 9]. There also exists work for only textual-similarity join and search [10, 11]. Spatial joins [12, 13, 14] and set-similarity joins [11, 15, 16, 17, 18, 19] are also well studied problems in the past. For spatio-textual similarity join, input is a set of tuples with each tuple consisting of a set of textual terms and object location components. The output is a pair of objects which are spatially close and also textually similar to set of terms specified. Each pair of objects in the output satisfy the given spatial and textual similarity threshold constraints. Formally, given two sets of objects M and N (in case of self join $M{=}N$) having both spatial and textual domain features, the join returns the subset $S \subseteq M \times N$, such that for every pair $(m, n) \in S$, m is both spatially close as well as textually similar to n as per the given threshold parameters.

However, computing join operation is an expensive task. Therefore, there is a need for some efficient mechanisms that reduce the number of candidate pairs which need

to be verified finally.

The state of the art algorithms for spatial-textual join like ST-SJOIN [2] performs join in two phases. In the first phase, a set of candidate pairs is generated based on some filtering mechanism. In the second phase, the candidates pairs are actually verified for their final inclusion in result set. To achieve better efficiency for join operation, non-candidate pairs filtering mechanism should be both computationally fast as well as effective in terms of finding the maximum numbers of non-candidate pairs. In addition, the filtering mechanism should be correct in terms of not filtering out any true positive to be present in the result set. In this thesis, we study a number of heuristics for their effectiveness in the context of spatial-textual join. We incorporate them in the state of art algorithm and observe the improvement. We observe that use of heuristics improves performance of ST-SJOIN by a factor of up to 5 [1]. The heuristics that we study are given as follows:

- **Efficient Grid Partitioning:** First heuristic makes an efficient Spatial Grid in order to reduce search space for possible candidates. This reduces search space by a factor of 25
- **Global Ordering of Elements:** Second heuristic orders elements of textual vector so that pruning becomes effective.
- **Prefix Comparison:** Third heuristic defines a prefix length of textual vector on the basis of the given similarity threshold and textual vector length being compared. The prefix length number of terms of textual vector are compared in the first phase instead of all the terms to decide whether the pair should be verified in the second phase.

We study the effectiveness of heuristics over real-life data sets, Foursquare and Flickr, by incorporating the heuristics in already existing state of the art algorithms. We notice that the use of heuristics reduces more non-candidates pairs and performs better than existing algorithms in terms of total computation time to produce a join set.

Then, we implement the same work in MapReduce framework which is a programming model for processing large data sets in parallel and distributed environment. There has been work done in spatial dimension [20, 21] and textual dimension [22]

using MapReduce. In MapReduce, we study 3 different approaches proposed by us and compare those 3 approaches for their efficiency. These approaches are described in section 6 of this thesis.

Applications: Spatio-Textual joins have various applications in fields where GPS systems are used or where spatial and textual data is available. Some of the applications are discussed below.

Matrimony databases: Matrimony databases can be used to pair up couples with similar interests based on the characteristics of people. This also enables a person to search partner at desired location. Today, matrimony websites returns the result with equal probabilities for every result, but using this approach, we can actually rank up results with most likely and relevant results at top. We can also tune either spatial or textual dimension for better results (in case of no or least likely results).

Education System: Education system can incorporate spatio-textual similarity join by recommending educational institutes to students of their interests. Every student have different interest in education field and the quality of education provided by that field differs from place to place. This system can recommend best college based on one's interests. E.g. Suppose a student want to do doctorate in Information Security and he doesn't want to go beyond 2500 KM for his studies, then this system can suggest best college within this range which are good in Information Security field.

Data Redundancy: Data redundancy or data de-duplication is mostly used in set-similarity joins [17, 23]. In set similarity joins, only textual data is considered but considering spatial data further improves removal/detection of redundant data. E.g. In education institute database, there may be labs in many institutes with same tags corresponding to similar labs but we cannot distinguish between them just based on their textual similarity, hence, spatial similarity plays an important role here. Another example can be copy of same data records at two different places. which can't be found for de-duplicacy without spatial dimension.

Academic Collaboration: Spatio-textual joins can be used by Academic people to collaborate with other academicians with similar interests in nearby locations.

Contribution: In past, there has been lot of work done in domain of spatio textual similarity join [1, 2, 3] and also, spatio textual similarity queries [2, 4, 5, 6, 7, 8]. The work is also carried out in variable prefix domain which is a framework for similarity join and search[10]. For spatio textual similarity queries, input is spatial distance and

set of terms (textual terms) and output is pair of objects which are spatially close and also textually similar to set of terms specified for between objects. Proposed heuristic for variable prefix filtering, inputs variable prefix length which gives results in minimum response time as compared to baseline algorithms. Also, spatio joins [12, 13, 14] and set similarity joins [11, 15, 16, 17, 18, 19] are well studied problems in past. Spatial [20, 21] and textual dimension [22] using MapReduce is the work done in recent years for parallelism of computation. We explore techniques of spatio-textual similarity join by applying variable prefix filtering to actually beat prefix filtering [1, 10] and other works done in past. This helps us prune our search space and improve our response time for desired results. We show results by tuning spatial and textual threshold, for pruning of non-candidates. We also study that, if the database is heavily clustered then applying spatial dimension first is fruitful as compared to applying textual dimension first. Also, if the objects of database is far apart from each other, then applying textual dimension first would fetch us good response time.

2

Related Work

The work done in this domain is related to spatial distance joins, similarity join and search, set similarity joins, spatio-textual search, Spatial joins using MapReduce, Textual joins using MapReduce and spatio-textual similarity join. Subsequent sub sections summarizes these works.

2.1 Spatial Distance Join

Spatial distance between two objects is the metric which tells how far apart the two objects are. Spatial data is usually indexed using R-trees [24] which indexes data of spatial objects using minimum bounding rectangles (MBR). Our work, ST-SJOIN-PF further extends ϵ distance join [14] where ϵ is spatial threshold between two objects. This means that given two databases of spatial dimensions, say A and B, ϵ distance returns the pairs of objects (a, b), a∈A, b∈B and $dist_t(a, b) \leq \epsilon$. This can be deployed in MBRs by recursively traversing and intersecting [13] the pairs which have minimum distance at most . We further discuss our approach in Section 5.4.

2.2 Similarity Join and Search

This approach is used to beat prefix filtering[1, 10] in terms of overall response time and do optimizations over textual search space by pruning more pairs of objects than pruned by prefix filtering scheme. This work is done only for textual similarity and search and doesn't consider spatial dimension. In this scheme, we use variable prefix filtering scheme which is actual key for this paper in which we verify for more textual

overlap in initial phase than in simple prefix filtering to prune out more pairs. This means, instead of checking only 1 common element for calculated prefix length, we check for multiple overlapping pairs and discard pairs which do not fulfill this overlap condition. This reduces verification time i.e. time to verify if a pair is actually spatially textually similar or not. This work is further described in section 5.

2.3 Set Similarity Joins

As the name suggests, set similarity finds the pairs of objects which are similar in their textual domain. It finds the pairs of objects which are at least Θ times similar where Θ is any similarity function among given sets. It finds applications in duplicate object detection [25] such as de-duplication of data, plagiarism detection, string matching [23], etc. Several methods have been proposed for this task in past [18] including methods based on lexicographic ordering of text, inverted files [26] approach to store number of objects and object id associated with each text/tag. This approach scans entire lists to check for the pair of candidates, hence optimizations were proposed, studied and implemented in which a small part of $objects'$ list is scanned to construct inverted list to compute join or discard pairs at same time. Chaudhari et al. [17] proposed an efficient filtering technique for set similarity join using prefix filter which means that for two objects to be candidate for join, at least some minimum overlap should be there among two strings which is calculated using prefix length of two objects. Prefix filtering is further improved by Jiannan et al. in [1, 10] where prefix length is taken to be variable than taking it as constant in [2]. Bayrado et al. [16] proposed framework which minimizes necessary elements to be added to inverted files for joining of objects by pre-computing bounds of element weights and also appropriate ordering of set elements in database. Proposed method of Bayrado et al. was further optimized by Xizo et al. [15] by enhancing performance of prefix filtering using positional information of elements in prefixes and partial suffix of objects joined. Section 5.4 discusses the method proposed by [1, 2, 10, 15].

2.4 Spatio Textual Search

Spatio-textual search finds major application and is often used often in applications like extracting spatial information from web pages or other databases which are then tagged using geo-tagging. Geo-data management is also typical application of spatio textual search. Often GPS is used to tag spatial and textual information for any object. Flickr uses geo tagging [27, 28, 29] to tag images into its database. GPS system is also used in tracking any system. E.g. in XIX Commonwealth games, around 1800 buses were provided with GPS systems to track them. Objects are given geotagged footprints which means they are annotated with set of locations which are often approximated by MBR in spatial domain. Also, geotagging enables searching objects/ documents by the text content associated with them along with their spatial information. The SPIRIT [30] search engine provides a test bed for the development of web search technology that is developed for access to geo spatial information which means the user inputs set of keywords and set of spatial predicates and the search engine returns the documents corresponding to user query. This further has capability of query expansion in case the user fails to get desired results. E.g. Find South-Indian restaurants within 5 KM from IIIT Delhi. Many indexing techniques have been proposed in [30, 31, 32, 33] for spatial-textual selection. Complex queries have also been studied for this type of search for matching query keywords [9, 34], prestige-based similarity [16] and also nearest neighbors [23] queries.

To the best of our knowledge, spatio-textual similarity joins have been studied in [1, 2, 3]. The work proposed by [2] have been studied in next subsection and also in section 5. Ballesteros et al. [3] computed SpSJOIn which differs from our work. In SpSJOIN, they combine both spatial and textual similarity together to find spatio-textual similarity between two objects x and y using $sim(x, y) = \frac{sim_{textual}(x,y)}{1+dist_{spatial}(x,y)}$ which combines both spatial distance, $dist_t(x, y)$ and textual similarity, sim(x,y) of objects. Also, SpsJOIN maximizes textual similarity between pairs of objects whereas ST-SJOIN-VPF considers both spatial and textual domain without dominating on either domain and retrieve pairs of objects which fulfills both spatial and textual threshold. Hence, even if two objects are spatially very close but their textual similarity doesn't overlap much, SpSJOIN join may result in 0 pairs of objects, which is improved in ST-SJOIN-VPF.

2.5 Spatial Join Using MapReduce

Lu et al.[20, 21] studied efficient processing of k Nearest Neighbor using MapReduce. They used Voronoi diagram to partition the spatial dimension in K planes. Then Mapper assigns every object, an id/key of plane where it lies. This ends up assigning same key to objects within a plane. Then they calculate lower and upper bounds to prune non-candidates. Every voronoi cell checks for its neighboring cell only if it fulfills lower and upper bounds and then send them to same reducer. Finally, reducer actually verifies candidates.

2.6 Textual Join Using MapReduce

Vernica et al.[22] studied efficient parallel set-similarity joins using MapReduce. They join two different files of records of textual vector. They perform MapReduce task in 3 stages. Stage 1 includes token ordering which is divided into two phases. Phase 1 computes frequency of every token. Phase 2 orders the tokens based on their frequencies. Then, stage 2 extracts the prefix length and filters out non-candidates whose terms in prefix length do not overlap. Stage 3 is Reduce stage where candidates are verified for actual join.

2.7 Spatio Textual Similarity Join

Recent work done by Panagiotus et al.[2] and Vivek et al.[1] enabled joining of spatio textual data including self-join on them. The proposed methods considers Euclidean distance for spatial similarity and Jaccard threshold for textual similarity. Also, they did some optimizations both on spatial and textual dimensions. The work done is really interesting and enables for further optimizations in this domain. For spatial dimension optimizations, they constructed dynamic grid and partitioned it in order to prune pairs with their spatial distance more than their spatial threshold. This reduces our calculation significantly when database is sparse. They also had some optimization in textual space in which they used prefix filtering to prune pairs of objects which do not satisfy minimum overlap condition. Detailed discussion about these are given in section 5.4 where we discuss baseline approach and also our approach over present prefix filtering approach to further reduce our search space.

3

PROBLEM DEFINITION

Given a database of spatio-textual objects, where each object in the database is represented as triple, denoted as $\langle obj.id,\ obj.loc, obj.text\rangle$ specifying the identity, location, and textual description of an object. Our goal is to find pairs of objects which are close in space and their overlaps in terms of text according to the given thresholds for spatial and textual dimensions, respectively. *obj.loc* takes values from 2-D geographic space and *obj.text* takes a set of terms of the global dictionary of terms. Each term in the dictionary can be weighted or unweighted based on relevance of term to that object. In this thesis, we use TF-IDF combination. The size of object x, denoted as $|x|$ as the numbers of terms contained in the object. The size of the object is used in defining the prefix length of objects.

Now, we denote the spatial distance for every pair of spatio-textual objects as $dist(x, y)$, as the distance between object x and y, and their textual similarity, $sim(x, y)$ as the set similarity between textual vector of terms associated with objects x and y. Spatial distance is measured as Euclidean distance, i.e., $dist(x, y) = dist(x.loc, y.loc)$. Textual similarity is measured as Jaccard similarity, i.e., $sim(x, y) = \frac{(|x.text\ \cap\ y.text|)}{(|x.text\ \cup\ y.text|)}$.

Given a database of spatio-textual objects O, the algorithm finds all pairs of objects (x, y) which are spatially close and textually similar i.e. $\forall x, y \in O$, a pair (x, y) appears in output if $dist(x, y) \leq \epsilon$ and $sim(x, y) \geq \Theta$ where ϵ and Θ are spatial and textual thresholds, respectively.

Table 3.1: Textual Similarity

θ	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8
x_1	1	0.2	0	0.4	0.33	0	0	0
x_2	0.2	1	0	0.17	0.14	0.33	0	0
x_3	0	0	1	0	0	0	0.6	0.5
x_4	0.4	0.17	0	1	0.8	0.13	0.13	0.11
x_5	0.33	0.14	0	0.8	1	0.11	0.2	0.18
x_6	0	0.33	0	0.13	0.11	1	0	0
x_7	0	0	0.6	0.13	0.2	0	1	0.83
x_8	0	0	0.5	0.11	0.18	0	0.83	1

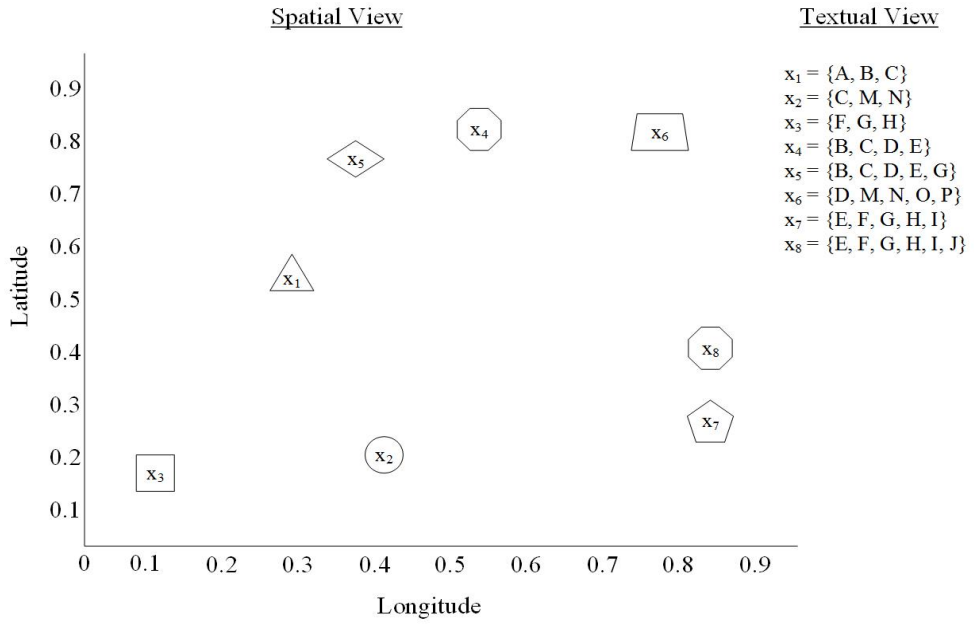

Figure 3.1: Example of 8 spatio-textual objects

Example 3.1 Consider a collection of 8 spatio-textual objects, O=$\{x_1, x_2, ..., x_8\}$ in Fig. 3.1 above. Every object have $\langle obj.id, obj.loc, obj.text\rangle$ like $\langle x_1, (0.3, 0.55), (A, B, C)\rangle$. Our algorithm, ST-SJOIN-VPF(O, 0.2, 0.8, 1) retrieves pairs $(x_4, x_5), (x_7, x_8)$. Table 3.1 and 3.2 shows spatial and textual similarities scores for all pairs of objects in the example. Object pair (x_7, x_8) has maximum textual similarity of 0.83 and similarly pair (x_1, x_5) has minimum spatial distance of 0.19.

Table 3.2: Spatial Distance

ϵ	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8
x_1	0	0.52	0.49	0.24	0.19	0.59	0.63	0.58
x_2	0.52	0	0.42	0.63	0.67	0.73	0.41	0.51
x_3	0.49	0.42	0	0.82	0.73	0.91	0.79	0.85
x_4	0.24	0.63	0.82	0	0.08	0.21	0.51	0.45
x_5	0.19	0.67	0.73	0.08	0	0.41	0.57	0.51
x_6	0.59	0.73	0.91	0.21	0.41	0	0.53	0.47
x_7	0.63	0.41	0.79	0.51	0.57	0.53	0	0.09
x_8	0.58	0.51	0.85	0.45	0.51	0.47	0.09	0

4

Prior Spatio Textual Similarity Join Methods

The section describes the methods for computing spatio-textual similarity joins. The section starts with standard PPJOIN algorithm modified with variable prefix which is extension to ALL PAIRS [27] algorithm and concludes with all spatio-textual join algorithms. In prefix filtering techniques, joins are calculated over inverted list of objects, which also makes sense since it is most efficient way of computing textual joins. An inverted index [26], also known as postings lists or inverted file is made for collection of objects and it is an index data structure which stores mapping from content, such as text associated with each object to its location in the database i.e. object.id. In simple terms, an inverted index is of form of (termId, docId) where termId is every term in dictionary and docId is object Id which contains that term. Now, in order to compute join, for each object x associated with O, we scan the posting list L_t of every term of object x, i.e. x.text and calculates 'n' length overlap of object x with every other objects say y $\in L_t$ which belongs to postings list of terms of object x and keep them in set $O_x[y]$. This way we find candidate pairs for object (x,y) associated with object x. We then compute actual similarity between every object (x,y) added as candidate pairs to check if they satisfy $sim(x, y) \geq \Theta$ to be actual pairs for join. This suffers from problem of double checking an object for candidate and join pairs i.e. pairs (x, y) and (y, x) are checked which produce same results but doubles computation. Hence, some optimizations are built over inverted index in which inverted index is built incrementally and join is processed on the fly. This means that while checking the

posting list of any object say x, we add an entry for x specifying that x have already been processed and we need not check it again. This helps us processing every object just once and hence computation reduces to half over baseline approach.

4.1 PPJOIN Algorithm

PPJOIN [15] Algorithm is an extension to All-Pairs algorithm which is further extended to combine positional filtering with prefix filtering. It takes as input a collection of Canonical objects already sorted in increasing order of their sizes. Then it sequentially scans each object x, finds candidate pairs that intersects $x's$ prefix and accumulates the overlap in a hash map A. The candidate sets so generated are verified to be actual pairs for joins in verification phase. Given two objects, the basic idea of positional filter is to compute an upper bound $\overline{O}$ of their maximum overlap O (x, y). If $\overline{O}$ is lower than minimum overlap required for candidate for join as required by the textual threshold, we can safely discard pair (x, y).

4.2 PPJ-I

PPJ-I constructs a dynamic grid partition on the spatial dimension of size, spatial threshold, ϵ, before performing joins. It defines a neighborhood for every cell to generate a spatial vector. These neighbor cells are in at most 3×3 grids around the corresponding cell. In dynamic grid, a cell index is maintained on top of every posting list in order to have direct access to initial entry/ entries in posting list. Every cell index contains an entry for $< cellid, ptr_{cellid} >$ for objects in posting list, L_t having first entry $\langle obj,\ pos_{obj} \rangle$ with obj.cellid=cellid. This helps eliminate efforts for linear scan of grid for a particular cell id and we can directly get entries for each cell. We have used variable prefix index in this approach (VPF-PPJOIN-VI) and shown efficient results as compared to baseline algorithms.

4.3 PPJ-C

Similar to PPJ-I, this algorithm also constructs a dynamic grid. It then orders the objects of database in increasing order of their cell-ids and does not use any neighbor-

hood interval concept. Therefore, this approach is named as Cell partitioning and so the suffix - C. This means that the algorithm orders the objects both by their cell ids and their sizes but prefers cell ids before their sizes. We have used variable prefix index in this approach (VPF-PPJOIN-C) and shown efficient results as compared to this algorithm. In this algorithm, we need to compute 1 self-join and at most 4 non-self join. For non-self join, [15] suggests merging contents of joined sets and then finding global ordering and again applying VPF-PPJOIN. In order to join two cells say c and c', we perform size based ordering of the content of each cell and adopts a merge sort strategy which calculates smallest distance between cells c and c'. If object $x \in c, y \in c'$, we scan an inverted index of c' and insert object x in the inverted index of $c(c')$.

4.4 PPJ-R

Spatial domain provides a powerful algorithm to define a data structure, namely R-Tree which is used to index multi-dimensional information including Geo-locations. We build upon this structure by assuming spatial indexing by R-Tree [24]. The algorithm works on spatial threshold, ϵ. It takes input in the forms of nodes for which it takes two nodes N_x, Ny as inputs and performs iterations in R-Tree. Initial call results in root of tree i.e. $N_x = N_y$. Now, if N_x and N_y are non-leaf nodes, then the algorithm identifies every pair of entries $(e_x, e_y) \in N_x \times N_y$such that the minimum distance between minimum bounding rectangles of r tree i.e. MBR(e_x) and MBR(e_y) is at most equal to the spatial threshold value ϵ. This gives us the pairs of objects that qualify spatial threshold and the algorithm runs recursively for all such nodes and all entries within nodes. We have used variable prefix index in this approach (VPF-PPJOIN-R) and shown efficient results as compared to this algorithm.

5

Proposed Spatio Textual Similarity Join Methods

This section describes the heuristics and improvements over prior algorithms studied in this thesis. We discuss the efficient grid partitioning scheme, followed by global order heuristic[1], further followed by prefix length heuristic[1]. We first construct Spatial Grid for spatial filtering in which we assign cell id's to each object as shown in line 1 of Algorithm 1. The algorithm takes as input a collection of Canonical objects O already sorted in increasing order of their their TF-IDF values, textual threshold ϵ, spatial threshold θ and variable prefix length n. Then, we compare a candidate with it's neighbor cells only in order to prune most of the non-candidates in spatial dimension. Our approach is built on top of PPJOIN algorithm which is modified in order to filter objects based on variable length prefixes. Then it sequentially scans each object x, finds candidate pairs that intersect the prefix of x and accumulates the overlap in a hash map. Line 1 indexes all the objects in spatial grid. Line 3 finds all the neighbors of an object. Line 4 sequentially scans every object in obtained from line 3 and then line 5 checks for spatial threshold between pairs of objects. If spatial threshold is satisfied in line 5, then it finds probe prefix length in line 6 and checks if the candidates fulfills probe prefix overlap in line 7. If they satisfies probe prefix overlap, then the candidate sets so generated are verified to be actually paired for joins in Line 8.

Algorithm 1: VPF-PPJOIN(O,Θ, ϵ, n)

Input : O is a collection of records sorted by the increasing order of TF-IDF values; each record has been canonicalized by a global ordering O_G a textual similarity threshold Θ *and a spatial distance threshold* ϵ. Also, we take Neighborhood Cell C for every object as input in order to prune many non-candidates.

Output : Set S of all pairs of objects (x, y), such that x,$y \in$ O, sim(x, y) $\geq \Theta$ *and* $dist(x, y) \leq \epsilon$.

$G_S \leftarrow Construct\_Spatial\_Grid(O, \epsilon)$ //Index all the objects in Spatial Grid.
forall the *objects* $x \in O$ **do**
Obtain N_x, neighborhood cells of x using G_x
for all *objects* $y \in N_x$ *not processed yet* **do**
  if $dist(x, y) \leq \epsilon$
    $p = |x| - (\Theta.|x|) + n.$ //Probe Prefix length
    if (prefix_sim(x,y)) $\geq n$
      if (sim(x,y)) $\geq \theta$
        $S = S \bigcup$ Join-Pairs(x,y)
    else
      discard pair(x, y)
  else
    discard pair(x, y)
return S

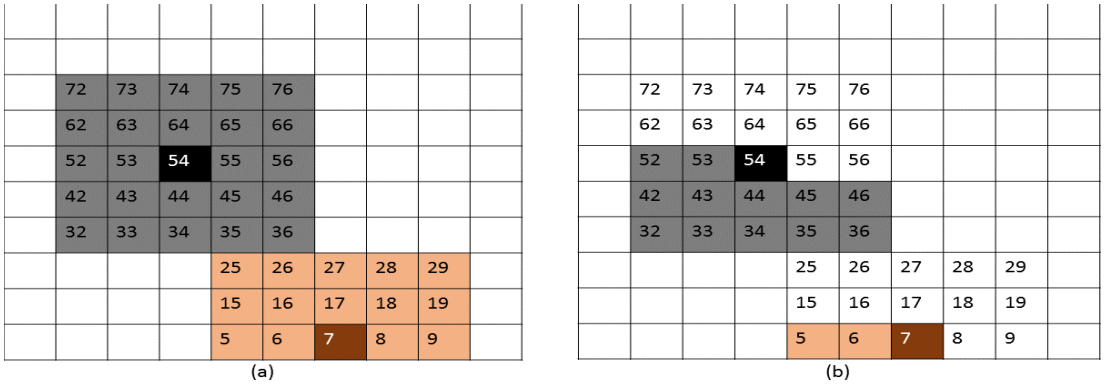

Figure 5.1: An example of efficient grid based partitioning (a) Interval Approach (b) Cell Approach

5.1 Efficient Grid Partitioning

We partition spatial dimension using Spatial Grid in order to prune non-candidates which are far away. Baseline approaches makes spatial grid of width ϵ in order to remove candidates in non-neighborhood cells. We further optimize this approach by saving 25% of search space by making spatial Grid of width $\epsilon/2$. Using this, we check for two neighborhood cells (instead of one in previous case), but this saves $\epsilon/2$ cell for every neighbor. Fig. 5.1 shows an example of Spatial Grid with width= $\epsilon/2$.

5.2 Global Ordering of elements on the basis of TF-IDF Values[1]

We order terms of textual vector. The motivation of this ordering is to make the pruning more effective by using more selective/ less frequent terms. A term which is more selective will appear only in very few objects and hence would become more effective in reducing the number of candidate pairs. Therefore we define a global order on terms using TF-IDF values. Lower the TF-IDF value, more is the probability of non-matching of terms (with most of other objects) in the database.

5.3 Prefix length for textual filtering[1]

In prefix filtering[1, 10] techniques, joins are calculated over using the prefix of inverted list of objects. An inverted index [26] is an index data structure which stores map-

O

x_1	$\{ t_1, t_2, t_4, t_6, t_8 \}$
x_2	$\{ t_2, t_3, t_5, t_7, t_9 \}$
x_3	$\{ t_1, t_2, t_3, t_4, t_8 \}$
x_4	$\{ t_1, t_5, t_7, t_8, t_9 \}$
x_5	$\{ t_2, t_6, t_7, t_8, t_9 \}$

(a)

O

x_1	$\{ t_4, t_6, t_1, t_2, t_8 \}$
x_2	$\{ t_3, t_5, t_7, t_9, t_2 \}$
x_3	$\{ t_3, t_4, t_1, t_2, t_8 \}$
x_4	$\{ t_5, t_1, t_7, t_9, t_8 \}$
x_5	$\{ t_6, t_7, t_9, t_2, t_8 \}$

(b)

Figure 5.2: (a) Object with 5 elements each containing 5 term, (b) Object ordered with TF-IDF scores

ping of the form of ($termId$, $docId$) from content, such as text associated with each object to its object id. In order to compute join, we check for minimum number of overlapping elements using prefix filtering to prune out most of the non-candidates. After all candidate pairs are generated, we check for actual pairs for join which satisfy $sim(x, y) \geq \Theta$. This suffers from problem of double checking an object for candidate and join pairs i.e. pairs (x, y) and (y, x) are checked which produce the same results but doubles computation. This optimization is already discussed in section 4. This helps us to process every object over once and hence, the computation reduces to half over the baseline approach.

In order to find variable prefix length, we order the terms according to global ordering O_G which rearranges the terms with increasing TF-IDF values in objects. We calculate the probe prefix of an object, denoted by ppref (x) as $p' = |x| - (\Theta.|x|) + n$, where the value of n can vary from 1 to $|x - 1|$. However, it has been observed from experiments that the algorithm attains better results when n lies in range of $|x|/3$ to $|x|/2$ where x is length of the object. In variable prefix filtering method, at least n terms must be shared by two objects in common to be candidates to join. We present an example demonstrating effect of variable length prefix filtering below.

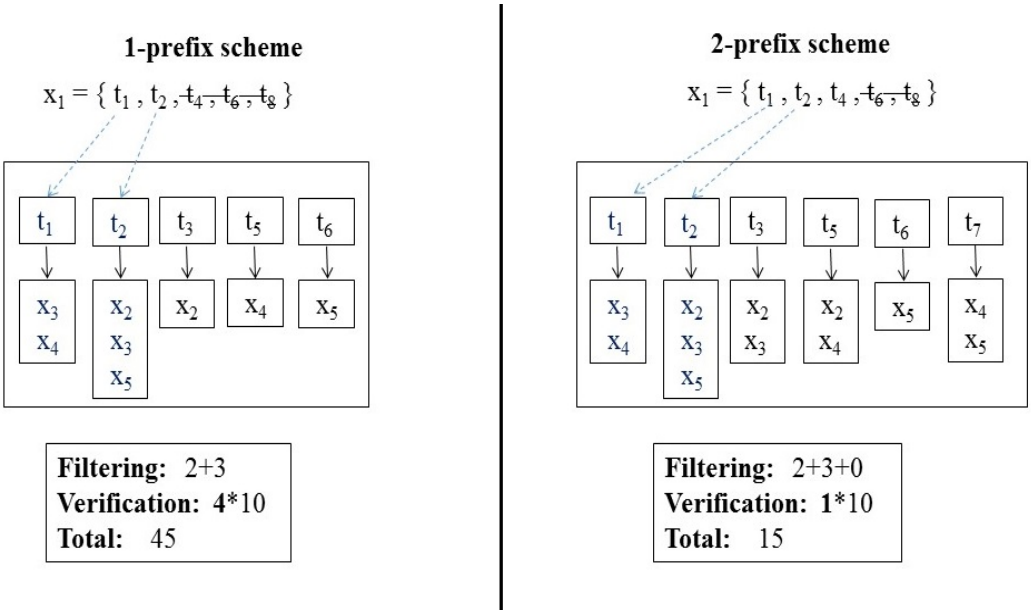

Figure 5.3: Cost Evaluation of 1-Prefix and 2-Prefix

Table 5.1: Cost Analysis of variable Prefix Scheme

S. No.	1-Prefix	2-Prefix	3-Prefix	4-Prefix
x_1	45	15	17	20
x_2	33	15	7	9
x_3	45	26	17	20
x_4	22	4	6	9
x_5	33	4	7	11

Example 5.1 Consider the collection of objects in Fig. 5.2 (a) and let textual threshold be $\Theta = 0.8$. Then, prefix filtering algorithm returns three pairs (x_1, x_5), (x_2, x_5) and (x_4, x_5) in the candidate set of x_5 with ppref $(x_5) = B, C$ whereas, 2-prefix filtering scheme returns just two pairs (x_1, x_5) and (x_4, x_5) and 3-Prefix filtering scheme returns just one pair i.e. (x_4, x_5). However, further increasing prefix scheme i.e. 3 and 4-prefix scheme also returns just one set (x_4, x_5) which results in poor performance of results as it moves towards brute force approach.

Example 5.2 Table 5.1 calculates the cost for textual join over object set O of 5 objects as shown in Fig. 5.2 (a). Fig. 5.2 (b) shows Global Ordering (Heuristic 1) for elements of objects based on their TF-IDF scores. Objects are joined using 1-Prefix,

2-Prefix, 3-Prefix and 4-Prefix methods with textual similarity threshold 0.8 i.e. $\Theta = 0.8$ (Heuristic 2). We calculate probe prefix length of object x_1 which comes out to be 2 and 3 in case of 1-Prefix and 2-Prefix respectively as shown in Fig. 5.3. It means we have to find at least 1 common element from initial two elements in case of 1-Prefix and at least 2 common elements from initial 3 elements in case of 2-Prefix. Moreover, it can be observed from Fig. 5.3, that as prefix length increases, filtering time increases. On the other hand, verification time reduces considerably, and hence overall response time decreases. Table for all prefix lengths is shown in Table 5.1 for all the objects which shows that as prefix lengths increases, total response time decreases, but increasing prefix length more than n/2 increases filtering time and hence overall response time increases.

5.4 Improvements over prior algorithms

The section deals with improvements over baseline algorithms. We study the effect of our proposed methods on top of prior algorithms and discuss the improvements in experimental section. For this, we make a dynamic grid partition (as discussed in section 5.1) on the fly which prunes out many pairs not fulfilling spatial threshold without extra efforts.

Dynamic grid partitioning approach is described by constructing a 2-D dynamic grid partition in which every square spans over a distance of spatial threshold, $\epsilon/2$ (Fig. 5.1) and hence the name dynamic grid partition as the grid is computed when ϵ is specified and span of grid changes with changing value of ϵ. In this approach, every object can pair up with objects in its neighbors where neighboring cells vary from 5 to 24 with 5 in case of object with cell id in corner of grid (e.g. cell id 1 in Fig. 5.1 (a)) and 24 in case of any middle cell id (e.g. cell id 54 in Fig. 5.1(a))). Cell ids are assigned to objects based on their latitude and longitude values and are assigned from top to bottom and left to right. Now, we represent object as quintuple (obj.id, ob.loc, obj.text, n-prefix, obj.cellid). We describe two approaches of dynamic grid partitioning proposed by [2] and includes additional functionality of variable prefix filtering to perform optimization to improve response time over their algorithms. We then explain these approaches with the help of example and then show experimental results in next

section.

5.4.1 VPF-PPJOIN-VI Algorithm

VPF-PPJOIN-VI algorithm is an extension over VPF-PPJOIN algorithm. VPF-PPJOIN considers spatial filter to be applied directly before textual threshold check whereas VPF-PPJOIN-VI further prunes out some entries by constructing a grid and discarding further pairs which are sure to fail spatial threshold condition. It differs from PPJ-I algorithm [2] in sense that it includes variable prefix filtering and efficient gird partitioning. Originally PPJ-I was studied in [2] which we are extending for our paper. It works as follows.

- VPF-PPJOIN-VI constructs a dynamic grid partition before performing join based in spatial threshold, . It defines the intervals for every cell id constructed during grid formation and then assigns at least 3 and at most 5 intervals to every cell id, hence the name variable interval- VI. These intervals corresponds to the neighbors of cell id in just two below row, same row and just two above rows, i.e. at most 5*5 grid intervals with defined cell id in between of all three intervals. As shown in Fig. 5.1(a) above, intervals for cell id 7 are [5, 9], [15, 19], [25,29] and similarly, intervals for cell id 54 is [32,36], [42, 46], [52, 56], [62, 66], [72,76].

- The algorithm keeps the entries of every posting list in ascending order of cell id for every entry.

- In dynamic grid, a cell index is maintained on top position of every posting list in order to have direct access to initial entry/ entries in posting list. Every cell index contains an entry for $< cellid, ptr_{cellid} >$ for objects in posting list, L_t having first entry $< obj, pos_{obj} >$ with obj.cellid=cellid. This helps eliminating efforts for linear scan of grid for a particular cell id and we can directly get entries for each cell.

- Now to check for spatial threshold in dynamic grid for a term, let us suppose term t, which is contained in probe prefix of x, VPF-PPJOIN-VI access entries $< obj, pos_{obj} >$ for each cell id which has objects in posting list, L_t and also falls in one of the intervals defined for corresponding cell id. Then in every interval

ranging from $[i_{min}, i_{max}]$, each entry, e is checked sequentially until $dist_t(obj, e) \leq \Theta$ and the remaining entries are discarded in those intervals.

5.4.2 VPF-PPJ-C Algorithm

VPF-PPJOIN-C algorithm is designed to further improve performance attained by VPF-PPJOIN and VPF-PPJOIN-VI. This also considers the ordering of objects in increasing order of their sizes and textual data in order of non-frequent terms first. It differs from PPJ-C algorithm [2] in sense that it includes variable prefix filtering and efficient gird partitioning. Originally PPJ-C was studied in [2] which we are extending for our purpose. The distinguishing features of VPF-PPJOIN-C are listed below.

- The algorithm constructs a dynamic grid as done in VPF-PPJOIN-VI (Fig. 5.1 (b)). It then orders the objects of database O in increasing order of their cell ids and not defining any interval and hence, it is named Cell partitioning and so the suffix C. This means that the algorithm orders the objects both by their cell ids and their sizes but prefers cell ids before their sizes.

- Then, it defines a set of cells S[c] which contains the objects y which can be joined with every object x in that cell id c. In such definition, S[c] includes cell c itself and other cell ids with distance of 1 unit distance in grid and cell ids smaller than c. E.g. In Fig. 5.1 (b), cell id 7 is checked with only 3 cells, i.e. S[7] = {5, 6, 7} and similarly, cell id 54 is checked with itself and adjacent cells with cell id smaller than 54, hence S[54]= {32, 33, 34, 35, 36, 42, 43, 44, 45, 46, 52, 53, 54}. This eliminates the cost of checking objects in cell ids {55, 56, 62, 63, 64, 65, 66, 72, 73, 74, 75, 76} which by intuition makes sense as later, these cells checks cell id 54. Hence, every cell is double checked in VPF-PPJOIN-VI which is eliminated in this algorithm and so the algorithm is cost effective.

- This algorithm is space effective as well as we need to store less cell ids as compared to variable interval algorithm as the algorithm discards cell ids with higher value than current cell id.

- In this algorithm, we need to compute 1 self-join and at most 12 non-self join.

5.4.3 VPF-PPJOIN-R Algorithm (R-Tree Approach)

Spatial domain provides a powerful algorithm to define a data structure namely R-Tree which is used to index multi-dimensional information including geo-locations and provide them with minimum bounding rectangle to define spacing between them in space. This enables us to apply R-Tree algorithm in order to prune candidates in spatial dimension. We build an algorithm for VPF-PPJOIN-R by assuming spatial indexing by R-Tree [24]. The algorithm works on spatial threshold, ϵ. It takes input in the forms of tree nodes. Initially, it takes two nodes N_x, Ny as inputs and performs iterations in R-Tree. Initial call results in root of tree i.e. $N_x = N_y$. Now, if N_x and N_y are non-leaf nodes, then the algorithm identifies every pair of entries $(e_x, e_y) \in N_x * N_y$ such that minimum distance between minimum bounding rectangles of r tree i.e. MBR(e_x) and MBR(e_y) is at most equal to spatial threshold value ϵ. This gives us the pairs of objects that qualify spatial threshold and the algorithm runs recursively for all such nodes and all entries within nodes.

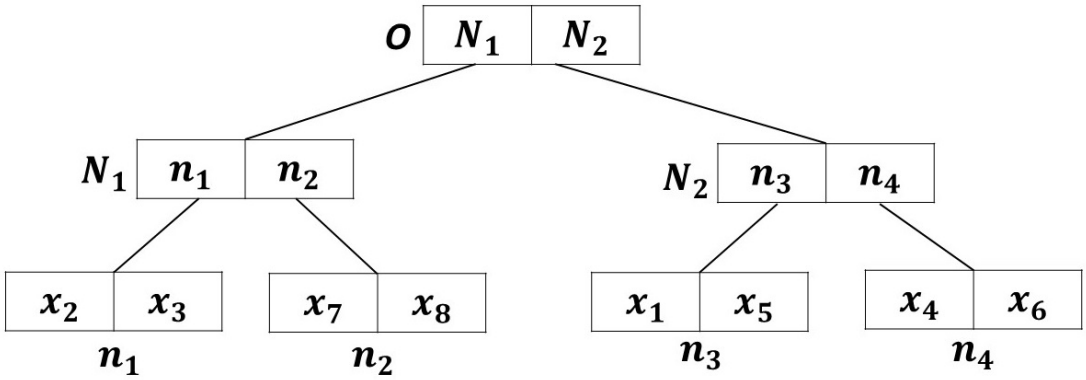

Figure 5.4: R Tree for Collection of Objects in figure 2

Further optimizations are proposed on this approach for computing spatial ϵ-distance joins [13, 14]. The algorithm extends MBR(N_x) and MBR(N_y) in all dimensions with threshold ϵ and computes their intersection area A. Every entry in N_x not intersecting in A with every other entry in N_y is discarded. Further, remaining entries (which are not processed yet) are sorted in their lower x-dimension in which entries in N_y are extended by ϵ and then their intersection join , I is computed using plane sweep based heuristic described in [13]. I contains all pairs of objects which fulfill spatial threshold

condition.

Table 5.2: Iterations for VPF-PPJOIN-R filtering phase for ST-SJOIN-VPF(O,0.1,0.9,1) for figure 2

Node Join	**Obj.**	**Checking Pair**	**Indexing**
$n_1,\ n_2 \bowtie$ n_1, n_2	x_2 x_3 x_7 x_8	$O_{x_8}[x_7] = 1$	$n_1.L_C = \{(x_2, 1)\}$ $n_1.L_F = \{(x_3, 1)\}$ $n_2.L_E = \{(x_7, 1)\}$ $n_2.L_E \cup = \{(x_8, 1)\}$
$n_3,\ n_4 \bowtie$ n_3, n_4	x_1 x_4 x_5 x_6	(x_5, x_4), *pruned* ϵ (x_5, x_1), *pruned* ϵ $O_{x_5}[x_4] = 1$	$n_3.L_A = \{(x_1, 1)\}$ $n_4.L_B = \{(x_4, 1)\}$ $n_3.L_B \cup = \{(x_5, 1)\}$ $n_4.L_D = \{(x_6, 1)\}$

E.g. Table 5.2 demonstrates the VPF-PPJOIN-R's iterations for ϵ = 0.2 , Θ =0.8, n=1 for Fig. 5.4. It shows overlapping and pruned pairs in every iteration.

The advantages over dynamic grid partition is that we need not compute dynamic grid partitioning and also need not assign cell id. In practice, R-Tree approach found to be expensive and less flexible because of tree parameters and drilling down to leave nodes for computing joins.

6

MapReduce implementation of spatio-textual joins

We propose 3 different ways of MapReduce implementation of spatio-textual similarity join. Two methods considers spatial dimension before textual dimension whereas one method considers textual dimension before spatial dimension. Before performing joins using MapReduuce, we do a preprocessing step which includes partitioning spatial grid and ordering terms of textual vector in TF-IDF order. The three approaches are described in subsequent sections.

Cost Analysis of MapReduce Algorithms We find lower and upper bound [35] for MapReduce tasks in order to compute cost of the three approaches we study in this paper. We study following inferences from [35] and correspondingly find lower and upper bounds for our approaches.

- **Lower Bound:** It is the function of maximum inputs assigned to a reducer.
- **Replication Rate:** It is the average number of Key-Value pairs to which each Input is mapped by Mappers.
- **Upper bound:** It can be defined as maximum number of inputs that one reducer can receive. We limit a reducer to no more input than can be processed in main memory (which is fair estimate of upper bound).

Based on above terminologies, we study formulae for Upper and lower bounds (again studied in [35]) as follows.

- **Upper Bound:** Let q be input to each reducer. We want to find output which a reducer can cover.
  Total I/P: $|I|$
  Total Output: $|O|$
  Input to each Reducer: q
  Output from each Reducer: g(q)

- **For p Reducers,**
  $\sum_{i=1}^{p} g(q_i) \geq |O|$

- **Lower bound on Replication Rate:** $\sum_{i=1}^{p} q_i/|I|$
  $=> \sum_{i=1}^{p} (q_i * g(q))/q \geq |O|$

- **Replication rate, r:** $\sum_{i=1}^{p} q_i/|I| \geq (q * |O|)/(g(q) * |I|)$

We incorporate these formulae to evaluate cost for our approaches.

6.1 Spatio-Textual Using Prefix Filtering (ST-PF)

This is 3 phase MapReduce implementation of spatio-textual similarity joins. In this approach, we consider spatial dimension in 1st phase (Mapping), then we consider textual dimension in 2nd phase. Final phase is Reduce phase where we check for actual pairs of objects to be candidates. Three phases can be summed up as follows.

6.1.1 Phase 1: Spatial Dimension Key-Value pairs:

In this phase, we assign a key to each cell. Then, for each cell, we assign it's neighbor cell id as it's value. This helps us generate key-value pairs in spatial dimension and pruning of far away candidates. E.g. In Fig. 5.1 (b), Key Value pairs for cell id 54 (with Dataset D) can be generated as- $(C_{54}, (D, C_{32}))$, $(C_{54}, (D, C_{33}))$, $(C_{54}, (D, C_{34}))$, $(C_{54}, (D, C_{35}))$, $(C_{54}, (D, C_{36}))$, $(C_{54}, (D, C_{42}))$, $(C_{54}, (D, C_{43}))$, $(C_{54}, (D, C_{44}))$, $(C_{54}, (D, C_{45}))$, $(C_{54}, (D, C_{46}))$, $(C_{54}, (D, C_{52}))$, $(C_{54}, (D, C_{53}))$ and $(C_{54}, (D, C_{54}))$

6.1.2 Phase 2: Textual Dimension Key-Value pairs (First Reduce Phase):

In this phase, we take key value pairs with same Key at reducer, then we find prefix length of objects lying in cell id which was key (E.g. Cell id 54 in our case). Then we, find prefix overlap with other candidates (with all other cell ids) and prunes out non-candidates which do not overlap with required terms prefix length. Then, without changing Key-Values (only discarding some objects), we send pairs to final reduce phase.

6.1.3 Phase 3: Candidate Verification (Second Reduce Phase):

In this phase, we take as input Key value pairs with same key (at same reducer). This phase then verifies for actual candidates by finding Jaccard similarity for textual dimension and Euclidean distance for spatial dimension. Finally, it returns pairs of objects fulfilling spatial and textual threshold criteria.

6.1.4 Cost Evaluation of ST-PF Algorithm:

As we implemented this algorithm in two MapReduce iterations, we will find lower and upper bounds and two phases and combine them to form single cost for lower and upper bound.

- **Input objects:** n
- **Output pairs:** nC2
- **First MapReduce Iteration:**
  - Input to each reducer (Average): $n * \epsilon^2$
  - Output from each reducer: $13 * n * \epsilon^2$
  - Lower Bound: $(n * \epsilon^2 * n^2)/(13 * n * \epsilon^2 * 2 * n) = n/26$
  - Upper Bound: $13 * n * \epsilon^2$
- **Second MapReduce Iteration:**
  - Input to each reducer (Average): $13 * n * \epsilon^2$

  - Output from each reducer: $(13 * n * \epsilon^2)/2$
  - Lower Bound: $(13 * n * \epsilon^2 * n^2)/(13 * n * \epsilon^2 * 2 * p * n) = n/p$
  - Upper Bound: $(13 * n * \epsilon^2)/2$

- **Overall Cost**
  - Lower Bound: $n(1/p + 1/26)$
  - Upper Bound: $(39 * n * \epsilon^2)/2$

6.2 Spatio-Textual Using Prefix Filtering with Global Token Ordering (ST-TO)

This is 3 phase MapReduce implementation of spatio-textual similarity joins. In this approach, we consider spatial dimension in 1st phase (Mapping), then we consider textual dimension in 2nd phase. Final phase is Reduce phase where we check for actual pairs of objects to be candidates. Three phases can be summed up as follows.

6.2.1 Phase 1: Spatial Dimension Key-Value pairs:

This phase is same as Spatial dimension phase of section 6.1.1.

6.2.2 Phase 2: Textual Dimension Key-Value pairs with Global Token Ordering (First Reduce Phase):

In this phase, we take key value pairs with same Key at reducer, then we find prefix length of objects lying in cell id which was key (E.g. Cell id 54 in our case). Then, it further generates Key Value pairs based on prefix terms. It takes prefix terms as key, and the objects, in which it is lying as it's value. E.g. Let cell id 54 and it's neighbor contains objects shown in Fig. 6.1, let $\theta = 0.8$. Then, $Prefix_{length}(x_8)$=2. Then, $Prefix_{terms}(x_8$=(E, F). Now, key value pairs can be generated as (E, (x_2, x_6, x_7)), (F, (x_1, x_3, x_4, x_7)). Similar task is performed for all pairs of objects. This prunes non-candidates which fail prefix filtering criteria. Finally, these pairs are send to Reduce phase for verification.

DataSet D

$x_1 = \{F, G, H\}$
$x_2 = \{E, M, N\}$
$x_3 = \{F, G, H\}$
$x_4 = \{F, G, H, I\}$
$x_5 = \{B, C, D, E, G\}$
$x_6 = \{D, E, N, O, P\}$
$x_7 = \{E, F, G, H, I\}$
$x_8 = \{E, F, G, H, I, J\}$

Figure 6.1: Example of 8 textual objects

6.2.3 Phase 3: Candidate Verification (Second Reduce Phase):

This phase is same as Reduce phase of section 6.1.3.

6.2.4 Cost Evaluation of ST-TO Algorithm:

As we implemented this algorithm in two MapReduce iterations, we will find lower and upper bounds and two phases and combine them to form single cost for lower and upper bound.

- **Input objects:** n
- **Output pairs:** nC2
- **First MapReduce Iteration:**
  - Input to each reducer (Average): $n * \epsilon^2$
  - Output from each reducer: $13 * n * \epsilon^2$
  - Lower Bound: $(n * \epsilon^2 * n^2)/(13 * n * \epsilon^2 * 2 * n) = n/26$
  - Upper Bound: $13 * n * \epsilon^2$
- **Second MapReduce Iteration (For Prefix Length K):**
  - Input to each reducer (Average): $13 * n * \epsilon^2$

  - Output from each reducer: $(13 * n * k * \epsilon^2)/2$
  - Lower Bound: $(13 * n * \epsilon^2 * k * n^2 * k)/(13 * n * k * \epsilon^2 * 2 * p * n) = (n * k)/p$
  - Upper Bound: $(13 * n * k * \epsilon^2)/2$

- **Overall Cost**
  - Lower Bound: $n(k/p + 1/26)$
  - Upper Bound: $13 * n * \epsilon^2(1 + k/2)$

6.3 Textual-Spatio Using Prefix Filtering with Global Token Ordering (TS-TO)

This is 3 phase MapReduce implementation of spatio-textual similarity joins. In this approach, we consider textual dimension in 1st phase (Mapping), then we consider spatial dimension in 2nd phase. Final phase is Reduce phase where we check for actual pairs of objects to be candidates. Three phases can be summed up as follows.

6.3.1 Phase 1: Textual Dimension Key-Value pairs with Global Token Ordering:

This phase is same as Map phase of section 6.2.2.

6.3.2 Phase 2: Spatial Dimension Key-Value pairs (First Reduce Phase):

In this phase, key-value pairs with same key are sent to same reducer. Then, the object set in values are checked if they lie in neighborhood cells. If they lie in neighborhood cells, they they are sent to final phase for verification, else the pair is discarded. E.g. Suppose, after 1st phase, we get one Key-Value pair as (F, (x_1, x_3, x_4, x_7)). Then, in this phase, we form object pairs (x_1, x_3), (x_1, x_4), (x_1, x_7), (x_3, x_4), (x_3, x_7), and (x_4, x_7). Now, we check if these pairs lies in neighborhood cells of each other. E.g. object x_3, x_7 lies in neighborhood cells of object x_1. The, we make another Key-Value pair as (x_1,(D,(x_3,x_7))) and send them for final verification. This way, spatial pruning is done in this phase.

6.3.3 Phase 3: Candidate Verification (Second Reduce Phase):

This phase is same as Reduce phase of section 6.1.3.

6.3.4 Cost Evaluation of TS-TO Algorithm:

As we implemented this algorithm in two MapReduce iterations, we will find lower and upper bounds and two phases and combine them to form single cost for lower and upper bound.

- **Input objects:** n
- **Output pairs:** nC2
- **First MapReduce Iteration (For Prefix Length K):**
  - Input to each reducer (Average): n/p
  - Output from each reducer: $(k * n/p)C2$
  - Lower Bound: $(n * 2 * p^2 * n^2 * k)/(p * n * n) = 2 * p * k * n$
  - Upper Bound: $(n^2 * k)/(2 * p^2)$
- **Second MapReduce Iteration :**
  - Input to each reducer (Average): $(n^2 * k)/(2 * p^2)$
  - Output from each reducer: $(13 * n^2 * k)/(2 * p^2)$
  - Lower Bound: $(n^2 * k * p)/(2 * p^2 * n * \epsilon^2) = (n * k)/(2 * p * \epsilon^2)$
  - Upper Bound:$(13 * n^2 * k)/(2 * p^2)$
- **Overall Cost**
  - Lower Bound: $(n * k) * (2 * p + 1/(2 * p * \epsilon^2))$
  - Upper Bound: $(7 * n^2 * k)/p$

6.4 Comparison of MapReduce Approaches

From cost analysis of different MapReduce Methods studied, we find that ST-PF performs better than ST-TO which further performs better than TS-TO. ST-PF incurs minimal cost in finding all pairs of candidates and hence outperforms other methods. Experimental section shows effect of all three approaches for varying values of different parameters.

7

Experimental Evaluation

The section describes the results obtained after experimental study. Section 7.1 describes the experimental setup in terms of hardware and the dataset used, Section 7.2 discusses the performance of the baseline and proposed methods, and Section 7.3 describes the performance of synthetic dataset with varying values of ϵ and Θ, etc.

7.1 Setup

Our experiments were performed on the Inspiron I5 processor with 6 GB RAM, 500 GB HDD. For MapReduce, we used Hadoop Cluster-1.0.3, 8 Nodes with 4 cores, 6 GB RAM in each Node and Ubuntu Server 12.04. Datasets comprise real and synthetic spatio-textual collections. For real dataset we crawl data from Flickr and FourSquare characteristics of which are described below.

- Flickr [1] dataset is comprised of text and location information associated with photographs in Flickr that we crawl using Flickr API. It contains 3,680,269 data objects with total dictionary size of 12,450,395 terms. Other EXIF details related to photographs were added to its textual description part. The average size (weight of an element is the number of characters in that object) for a textual description vector is 9.2.

- Foursquare [2] dataset consists of history of users check-ins over a span of around 10 months. It contains 1.2 million objects with global dictionary size of 532,943

[1] Flickr garden, https://www.flickr.com/services/api/

[2] Refer [36]

terms. The average size of textual description for every object came out to be 5.1.

For generating smaller size datasets, we sample objects randomly from our real data set on the basis of (i) number of objects , $|O|$ and (ii) Size of global dictionary $|T|$ as given in Table 7.1.

Table 7.1: Synthetic Dataset Statistics

Parameter	**Values**
$\|O\|$	$50,000$ $250,000$ **500,000** $1,000,000$ $5,000,000$
$\|T\|$	$5,000$ $25,000$ **50,000** $100,000$ $500,000$
ϵ	**0.001** 0.005 0.01 0.05 0.1
Θ	0.5 0.6 0.7 **0.8** 0.9
n	1 2 $\frac{\|n\|}{2}$ $\|n\|$

We measure the effect of spatial threshold and the textual threshold on computation time. Table 7.1 shows range of values for these thresholds used for experiments. We vary ϵ from 0.001 to 0.1 and Θ from 0.5 to 0.95. We vary 'n' prefix from 1 to $\frac{|obj|}{2}$ where $|obj|$ is size of object . The tightest values, i.e. ϵ=0.0001 and Θ=0.95 model de-duplication of objects while intermediate values, e.g., $\epsilon = 0.1$ and $\Theta = 0.6$ corresponds to the application of a recommendation system/ matrimony system. For R-Tree algorithm, we pre-construct an R-Tree and doesn't take into account construction the time for creating R-Tree. Indexes are stored in main memory. We implement original PPJOIN algorithm and modify it by adapting textual-similarity heuristics. Then we implement MapReduce version of same algorithms.

7.2 Improvement over baseline algorithms

The section compares our proposed methods, namely ST-SJOIN-VPF and ST-SJOIN-MR against baseline methods for spatial and textual joins dealt separately. For spatial dimension, R Tree methods indexes the objects inside MBRs using spatial distance. To perform join, it constructs MBRs with spatial threshold ϵ and for each qualifying

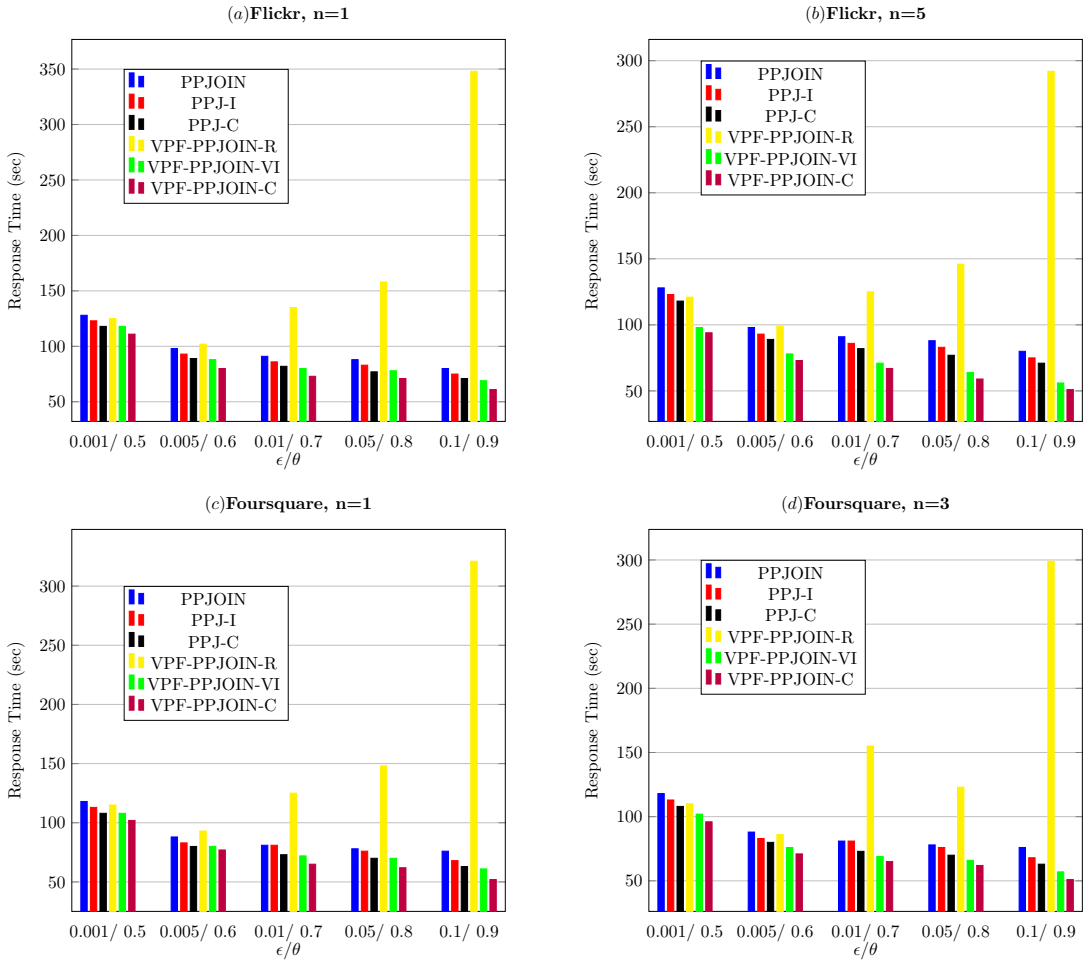

Figure 7.1: Comparison of running time of different Algorithms

pair (x, y) such that $dist(x, y) \leq \Theta$, it computes $sim(x, y)$ and verifies if their textual similarity satisfies threshold condition i.e. $\text{sim}(x, y) \geq \Theta$. This differs from PPJOIN algorithm in a way that this considers a spatial filter during the verification phase to check if $dist(x, y) \leq \Theta$ for pair (x, y). It differs from VPF-PPJOIN in two ways- (i) VPF-PPJOIN checks for spatial threshold during the filtering phase. (ii) VPF-PPJOIN employs variable prefix than 1-Prefix in PPJOIN algorithm.

Fig. 7.1 and Fig. 7.2 shows the experimental results for response time of all defined methods in terms of their response times for non-MapReduce and MapReduce implementation respectively. We vary ϵ/Θ range from 0.001/0.5 to 0.1/0.95 to study the

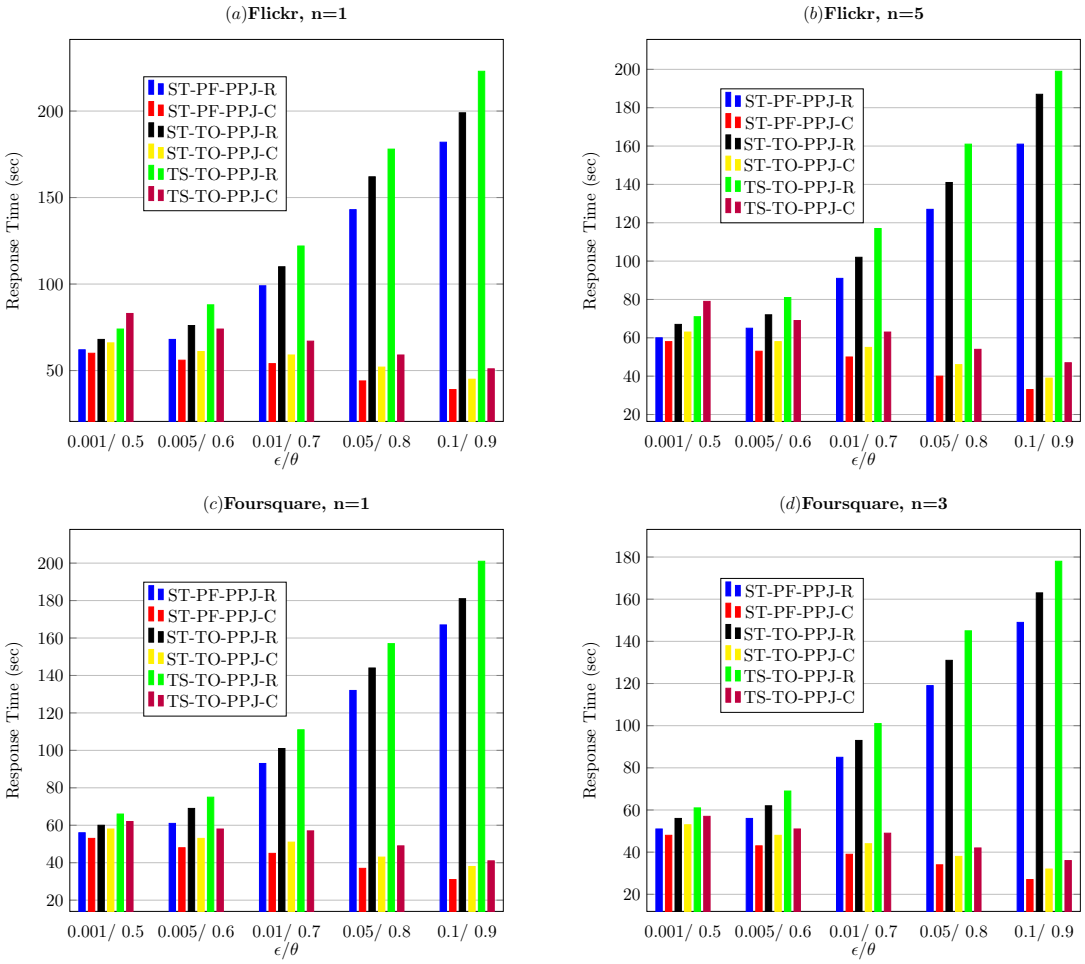

Figure 7.2: Comparison of running time of different Algorithms using MapReduce

effect of varying parameters. As ϵ is kept constant and Θ varied, the join is dominated to spatial distance join and as Θ is kept constant and ϵ varied, the join is dominated to textual join. With loose spatial threshold, response time for R-Tree method increases, but the response time of PPJOIN decreases. Both databases show with different behavior. Flickr has an average length of object as 9.2 compared to Foursquare with an average length of 5.1. This means that for Flickr, textual threshold will play an effective role as compared to spatial threshold and for Foursquare, spatial threshold would prune out pair of objects easily as compared to textual threshold.

R-Tree algorithms shows poor performance with all sets of data because of large

number of leaf nodes with much smaller extent as compared to a spatial threshold which makes it join a large number of pairs at leaves. On the other hand, in a dynamic grid partitioning, cell id is defined in which an object is checked with a limited number of pairs (to its adjacent neighbors) and hence response time improves.
In synthetic dataset, where spatial threshold is extremely small and textual threshold is extremely loose, say $\epsilon = 0.001$ and $\Theta = 0.5$ or so, R-tree approach outperforms other algorithms as the numbers of pairs qualifying spatial filter almost equals both spatial and textual filter threshold and textual filter doesn't play much role here and R-Tree has less leaf in this case.

7.3 Comparison of STS-JOIN-VPF Method

Previous sections dealt with superiority of our proposed approach over PPJOIN or R-Tree approach. In this section, we emphasize on experimental analysis of our algorithms to study the behavior of our algorithms, namely VPF-PPJOIN, VPF-PPJOIN-VI, VPF-PPJOIN-C, VPF-PPJOIN-R. Then we study improvements attained by MapReduce approach on these data sets. We study different algorithms namely ST-PF-PPJ-R and ST-PF-PPJ-C for R Tree and Cell based approach of section 6.1, ST-TO-PPJ-R and ST-TO-PPJ-C for R Tree and Cell based approach of section 6.2, and TS-TO-PPJ-R and TS-TO-PPJ-C for R Tree and Cell based approach of section 6.3. The variations are done on synthetic data to observe effects of changing parameter values.

7.3.1 Varying the number of objects $|O|$:

Effect of variation of number of objects is studied in Fig. 7.3, 7.4 and 7.5. Intuitively and experimentally, when number of objects increases, response time of every algorithm increases. Experimentally, VPF-PPJOIN-C and ST-PF-PPJ-C dominates all algorithms specified (in non-MapReduce and MapReduce versions respectively) as there is pruning in all dimensions. VPF-PPJOIN-R turns out to have higher response time because it doesn't have effective spatial filtering optimizations in order to prune objects in spatial domain.

7.3.2 Varying the size of dictionary $|T|$:

Effect of variation of the size of the dictionary is studied in Fig. 7.3, 7.4 and 7.5. As the size of dictionary increases, response time increases. But, sometimes response time decreases when prefix filtering approach prunes out most of the non-candidate pairs. All methods are affected by increasing the size of the dictionary, still VPF-PPJOIN-C and ST-PF-PPJ-C outperforms other methods in non-MapReduce and MapReduce versions respectively.

7.3.3 Varying the length of prefix 'n':

Effect of variation of length is studied in Fig. 7.6, 7.7 and 7.8. As n varies from 1 to half the size of object, filtering time increases, but verification time decreases considerably which dominates the overall response time.It can be seen that further increasing 'n' leads to brute force checking as this checks a lot of entries to be common in both the objects. Hence, approximately n=$|size|/3$ to n=$|size|/2$ gives better results, after which response time increases considerably.

7.3.4 Varying the spatial distance threshold ϵ:

Effect of variation of spatial distance threshold is studied in Fig. 7.9, 7.10 and 7.11. As ϵ increases, number of results in join pairs increases and hence the running time of all algorithms increases. In such cases, R-Tree algorithm's approach suffers from very high response time as the number of leaf nodes increases significantly. Also, a database with a higher average length of an object is not much affected as such databases are mainly textual threshold dominated. In our experimental setup, VPF-PPJOIN-C and ST-PF-PPJ-C showed the best results among all algorithms (in non-MapReduce and MapReduce versions respectively), and VPF-PPJOIN-R suffered significantly.

7.3.5 Varying the textual similarity threshold Θ:

Effect of variation of textual similarity threshold is studied in Fig. 7.12, 7.13 and 7.14. With increase in Θ, number of results in join pairs decreases and hence, the response time of all algorithms increases. If the average length of objects is small, then this doesn't have much impact on response time as response time is dominated by spatial threshold in such cases.

7.3.6 Number of objects pruned:

Effect of filtering power of proposed algorithm is studied in Fig. 7.15, 7.16 and 7.17. Number of objects pruned in case of Interval and Cell approaches are same as it Interval method only increases computation time because of double checking of objects. Huge impact is being shown in terms of pruning of candidates.

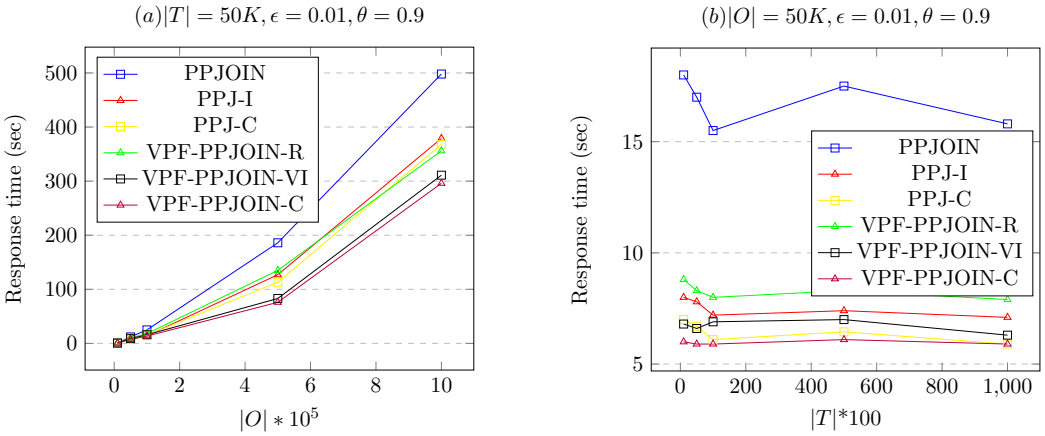

Figure 7.3: Varying number of objects/ size of dictionary without TF-IFD Ordering.

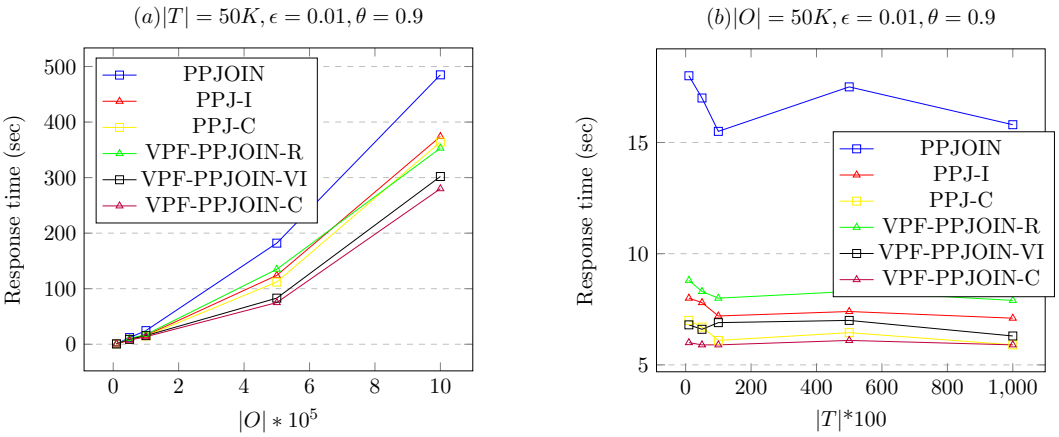

Figure 7.4: Varying number of objects/ size of dictionary with TF-IFD Ordering.

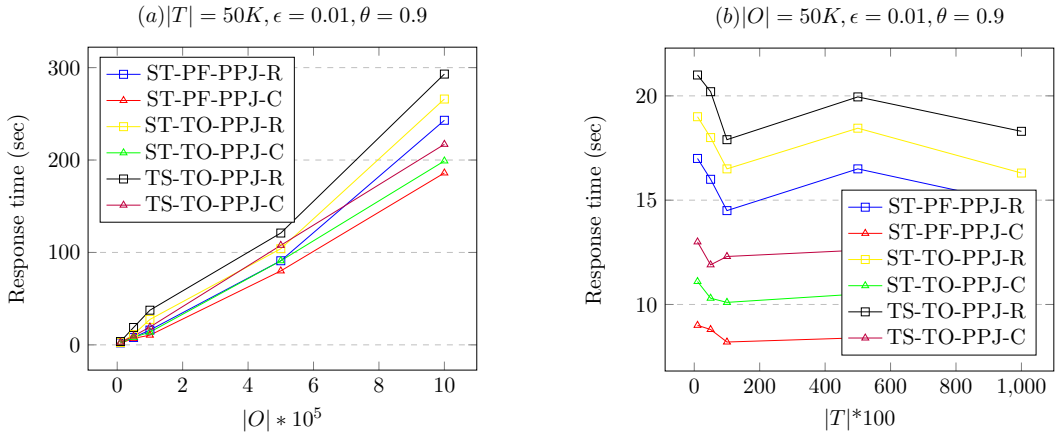

Figure 7.5: Varying number of objects/ size of dictionary using MapReduce

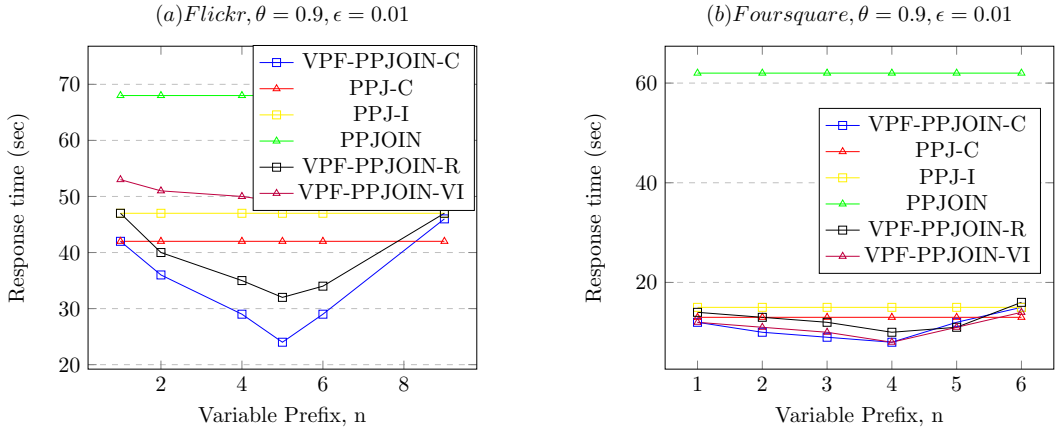

Figure 7.6: Varying the prefix length, n, without TF-IDF Ordering

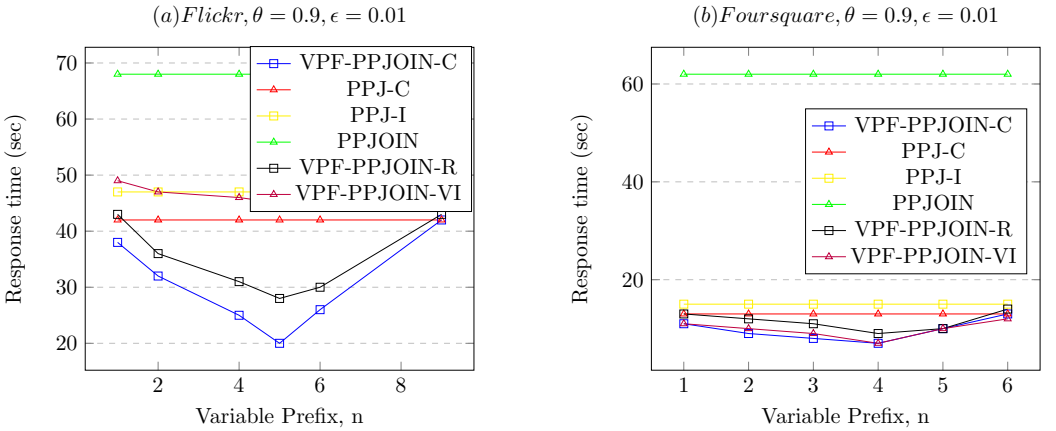

Figure 7.7: Varying the prefix length, n, with TF-IDF Ordering

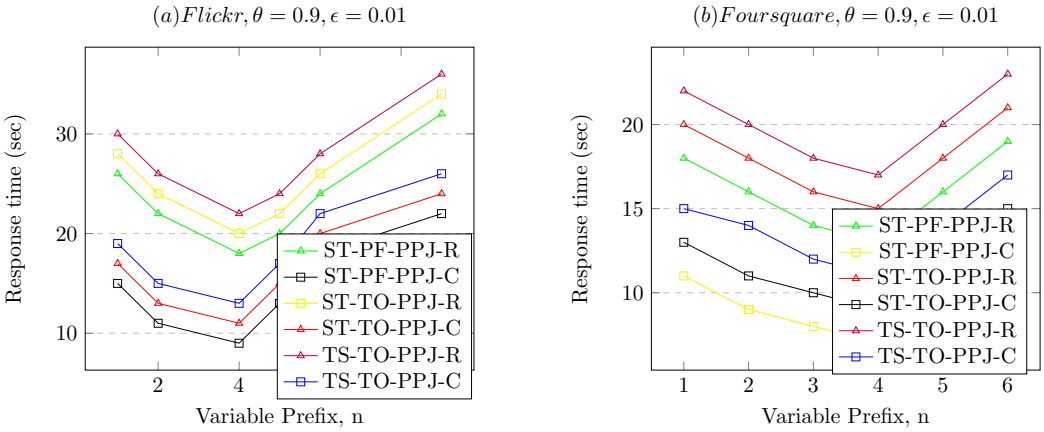

Figure 7.8: Varying the prefix length, n, using MapReduce

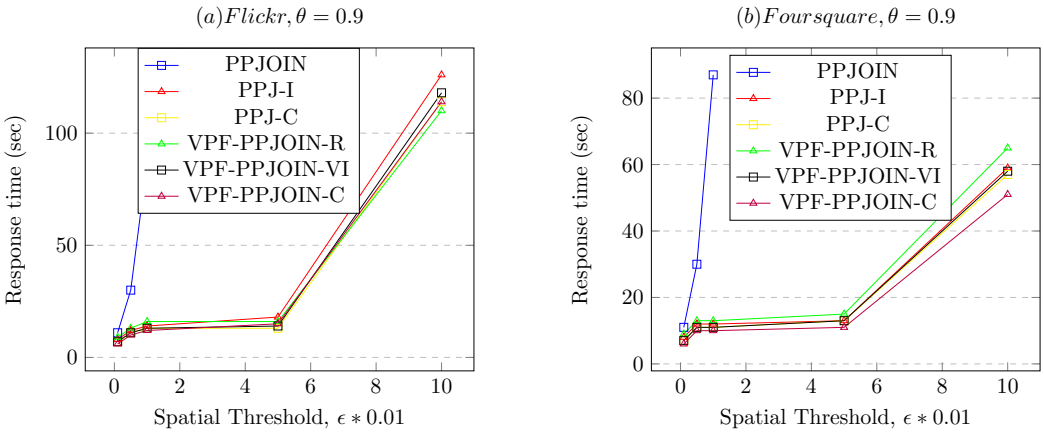

Figure 7.9: Varying the spatial threshold, ϵ, without TF-IDF Ordering

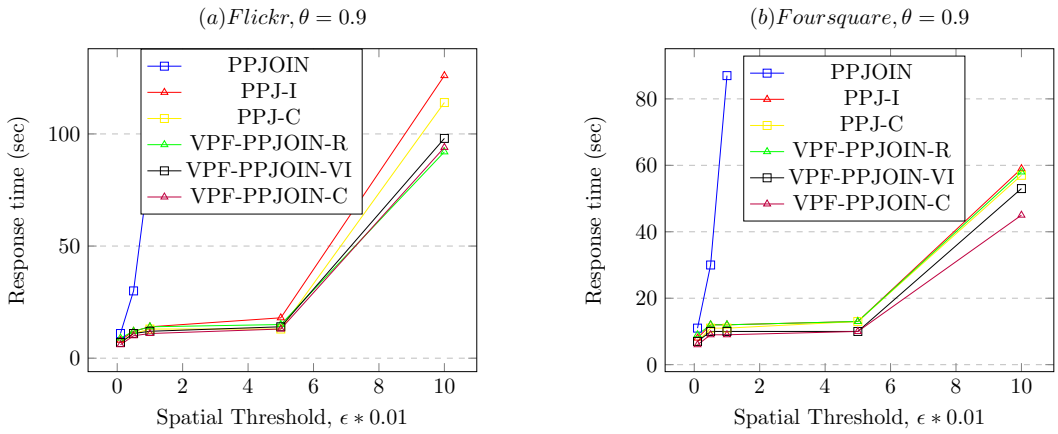

Figure 7.10: Varying the spatial threshold, ϵ, with TF-IDF Ordering

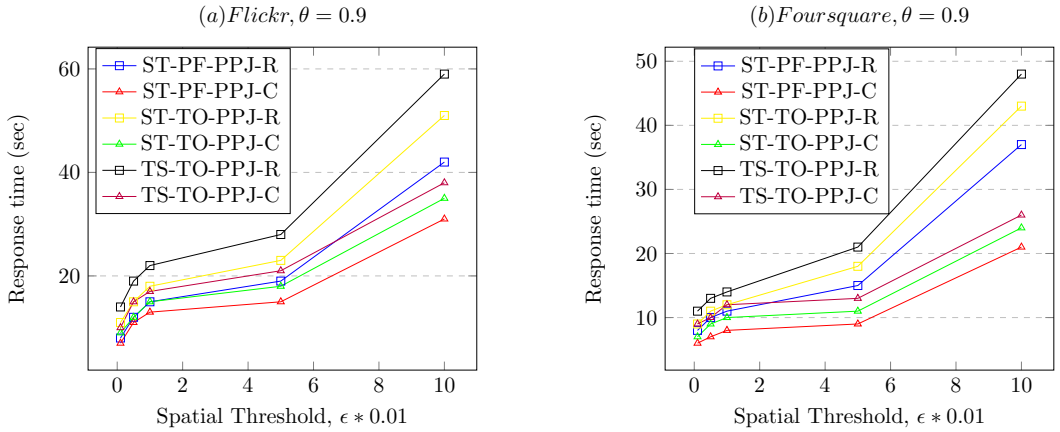

Figure 7.11: Varying the spatial threshold, ϵ, using MapReduce

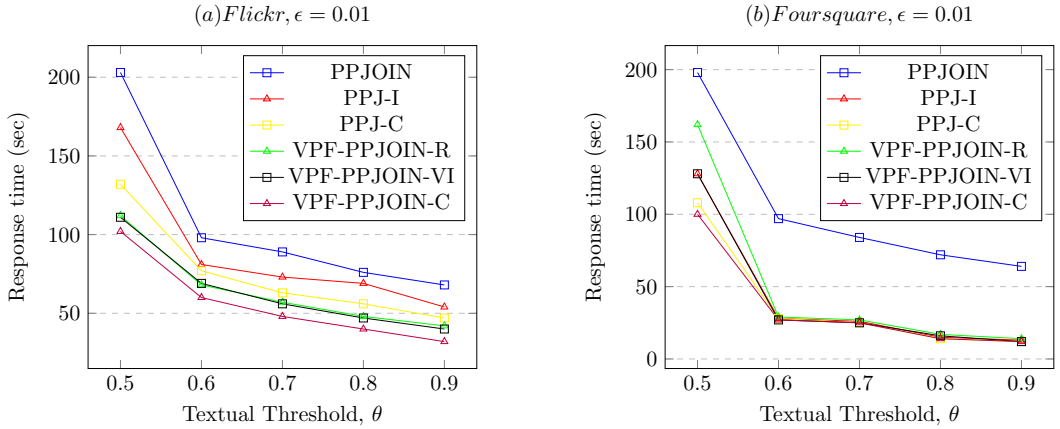

Figure 7.12: Varying the textual threshold, Θ, without TF-IDF Ordering

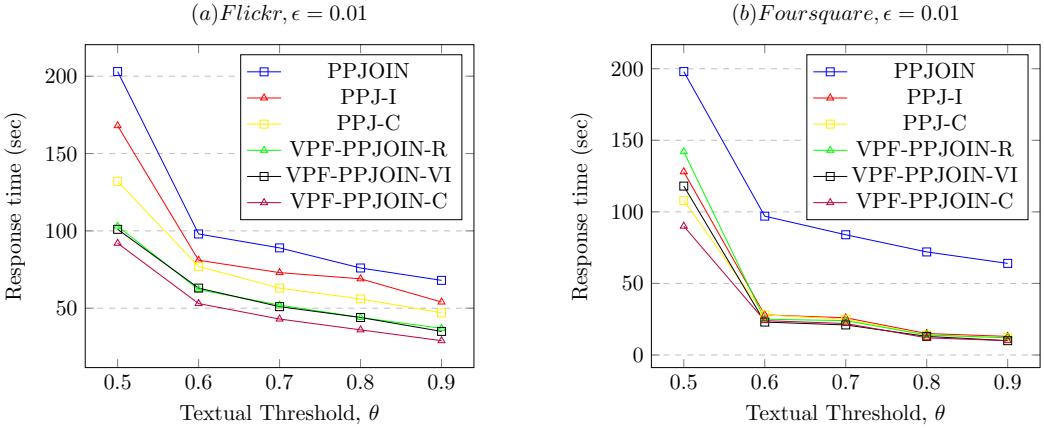

Figure 7.13: Varying the textual threshold, Θ, with TF-IDF Ordering

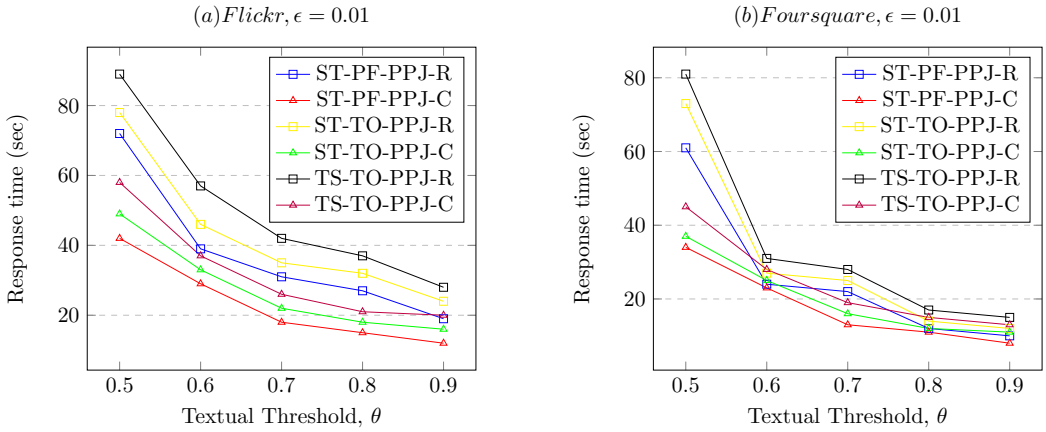

Figure 7.14: Varying the textual threshold, Θ, using MapReduce

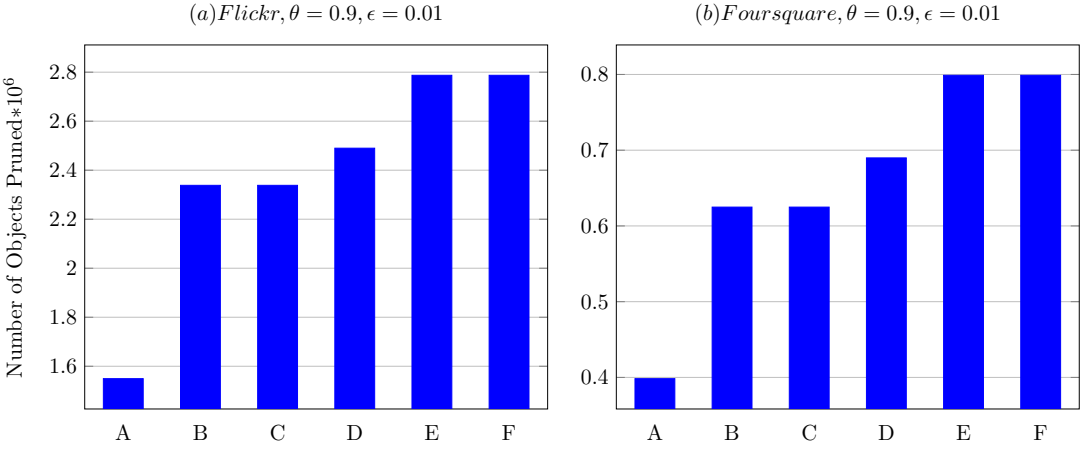

Figure 7.15: Number of objects pruned without TF-IDF Ordering{A= PPJOIN, B= PPJ-I, C= PPJ-C, D= VPF-PPJOIN-R, E= VPF-PPJOIN-VI, F= VPF-PPJOIN-C}

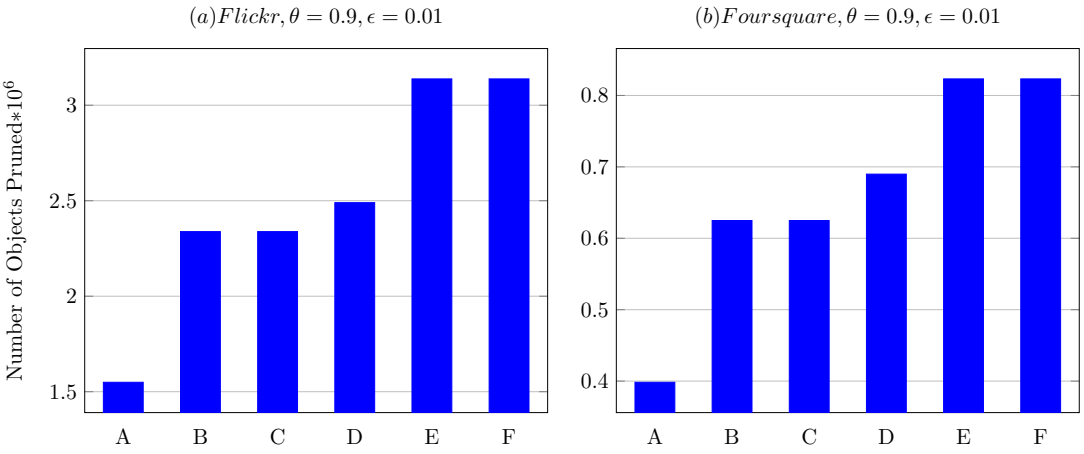

Figure 7.16: Number of objects pruned with TF-IDF Ordering {A= PPJOIN, B= PPJ-I, C= PPJ-C, D= VPF-PPJOIN-R, E= VPF-PPJOIN-VI, F= VPF-PPJOIN-C}

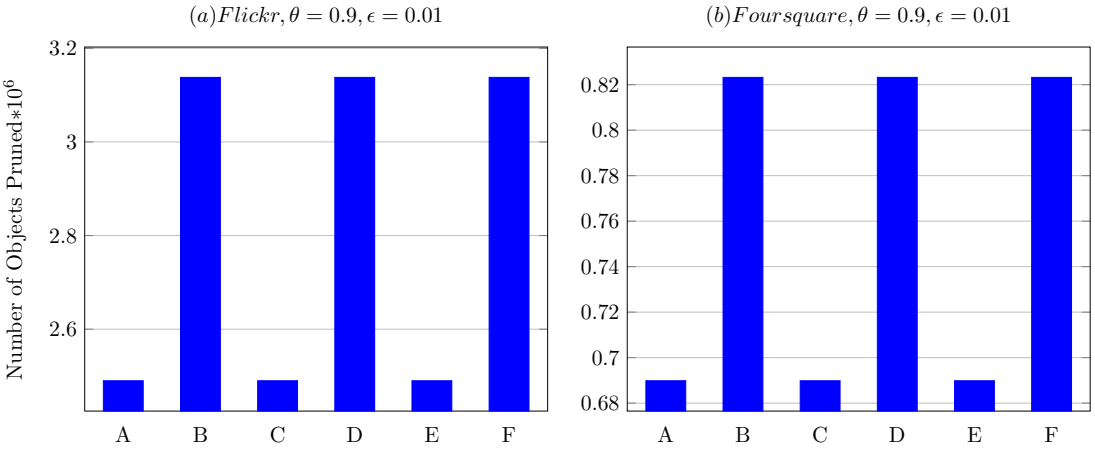

Figure 7.17: Number of objects pruned using MapReduce {A= ST-PF-PPJ-R, B= ST-PF-PPJ-C, C= ST-TO-PPJ-R, D= ST-TO-PPJ-C, E= TS-TO-PPJ-R, F= TS-TO-PPJ-C}

8

Conclusion and Future Work

The paper deals with spatio-textual similarity joins which have a wide range of applications. We studied a number of heuristics, namely efficient grid partitioning, prefix filtering and TF-IDF ordering proposed earlier in different contexts, for the problem of spatial-textual join. We find that incorporation of these heuristics improves the performance of spatial-textual join algorithm. We performed extensive experimental study and observed that approximate prefix length of $\frac{n}{3}$ to $\frac{n}{2}$ with term ordering achieves best performance most of the times. We also implemented this work through MapReduce framework and studied substantial improvements over our methods defined. As a future work, the proposed approach can be extended by incorporating other textual-similarity measures like Hybrid Jaccard similarity and Extended Jaccard similarity. Another direction can be combining spatial and dimension simultaneously during Mapping phase (in MapReduce) for single iteration MapReduce implementation. Also, we can take spatial threshold as actual value then relative value (E.g. $\epsilon = 5$ KM instead of 0.1).

References

[1] Vivek Gupta and Vikram Goyal. **Spatio-textual Similarity Joins Using Variable Prefix Filtering**. In *Proceedings of the Second ACM IKDD Conference on Data Sciences*, CoDS '15, pages 120–121, New York, NY, USA, 2015. ACM. vii, 1, 2, 3, 4, 5, 6, 7, 8, 15, 17, 18, 19

[2] Panagiotis Bouros, Shen Ge, and Nikos Mamoulis. **Spatio-textual Similarity Joins**. *Proc. VLDB Endow.*, **6**(1):1–12, November 2012. 1, 2, 3, 6, 7, 8, 20, 21, 22

[3] Jaime Ballesteros, Ariel Cary, and Naphtali Rishe. **SpSJoin: Parallel Spatial Similarity Joins**. In *Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, GIS '11, pages 481–484, New York, NY, USA, 2011. ACM. 1, 3, 7

[4] Xin Cao, Gao Cong, and Christian S. Jensen. **Retrieving Top-k Prestige-based Relevant Spatial Web Objects**. *Proc. VLDB Endow.*, **3**(1-2):373–384, September 2010. 1, 3

[5] Gao Cong, Christian S. Jensen, and Dingming Wu. **Efficient Retrieval of the Top-k Most Relevant Spatial Web Objects**. *Proc. VLDB Endow.*, **2**(1):337–348, August 2009. 1, 3

[6] Ian De Felipe, Vagelis Hristidis, and Naphtali Rishe. **Keyword Search on Spatial Databases**. In *Proceedings of the 2008 IEEE 24th International Conference on Data Engineering*, ICDE '08, pages 656–665, Washington, DC, USA, 2008. IEEE Computer Society. 1, 3

[7] Zhisheng Li, Ken C. K. Lee, Baihua Zheng, Wang-Chien Lee, Dik Lee, and Xufa Wang. **IR-Tree: An Efficient Index for Geographic Document Search**. *IEEE Trans. on Knowl. and Data Eng.*, **23**(4):585–599, April 2011. 1, 3

[8] João B. Rocha-Junior, Orestis Gkorgkas, Simon Jonassen, and Kjetil Nørvåg. **Efficient Processing of Top-k Spatial Keyword Queries**. In *Proceedings of the 12th International Conference on Advances in Spatial and Temporal Databases*, SSTD'11, pages 205–222, Berlin, Heidelberg, 2011. Springer-Verlag. 1, 3

[9] Dongxiang Zhang, Yeow Meng Chee, Anirban Mondal, Anthony K. H. Tung, and Masaru Kitsuregawa. **Keyword Search in Spatial Databases: Towards Searching by Document**. In *Proceedings of the 2009 IEEE International Conference on Data Engineering*, ICDE '09, pages 688–699, Washington, DC, USA, 2009. IEEE Computer Society. 1, 7

[10] Jiannan Wang, Guoliang Li, and Jianhua Feng. **Can We Beat the Prefix Filtering?: An Adaptive Framework for Similarity Join and Search**. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, SIGMOD '12, pages 85–96, New York, NY, USA, 2012. ACM. 1, 3, 4, 5, 6, 17

[11] Arvind Arasu, Venkatesh Ganti, and Raghav Kaushik. **Efficient Exact Set-similarity Joins**. In *Proceedings of the 32Nd International Conference on Very Large Data Bases*, VLDB '06, pages 918–929. VLDB Endowment, 2006. 1, 4

[12] Christian Böhm, Bernhard Braunmüller, Florian Krebs, and Hans-Peter Kriegel. **Epsilon Grid Order: An Algorithm for the Similarity Join on Massive High-dimensional Data**. In *Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data*, SIGMOD '01, pages 379–388, New York, NY, USA, 2001. ACM. 1, 4

[13] Thomas Brinkhoff, Hans-Peter Kriegel, and Bernhard Seeger. **Efficient Processing of Spatial Joins Using R-trees**. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, SIGMOD '93, pages 237–246, New York, NY, USA, 1993. ACM. 1, 4, 5, 23

[14] E.P.F. Chan. **Buffer queries**. *Knowledge and Data Engineering, IEEE Transactions on*, **15**(4):895–910, July 2003. 1, 4, 5, 23

[15] Chuan Xiao, Wei Wang, Xuemin Lin, Jeffrey Xu Yu, and Guoren Wang. **Efficient Similarity Joins for Near-duplicate Detection**. *ACM Trans. Database Syst.*, **36**(3):15:1–15:41, August 2011. 1, 4, 6, 13, 14

[16] Roberto J. Bayardo, Yiming Ma, and Ramakrishnan Srikant. **Scaling Up All Pairs Similarity Search**. In *Proceedings of the 16th International Conference on World Wide Web*, WWW '07, pages 131–140, New York, NY, USA, 2007. ACM. 1, 4, 6, 7

[17] Surajit Chaudhuri, Venkatesh Ganti, and Raghav Kaushik. **A Primitive Operator for Similarity Joins in Data Cleaning**. In *Proceedings of the 22Nd International Conference on Data Engineering*, ICDE '06, pages 5–, Washington, DC, USA, 2006. IEEE Computer Society. 1, 3, 4, 6

[18] Sunita Sarawagi and Alok Kirpal. **Efficient Set Joins on Similarity Predicates**. In *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*, SIGMOD '04, pages 743–754, New York, NY, USA, 2004. ACM. 1, 4, 6

[19] Chuan Xiao, Wei Wang, Xuemin Lin, and Haichuan Shang. **Top-k Set Similarity Joins**. In *Proceedings of the 2009 IEEE International Conference on Data Engineering*, ICDE '09, pages 916–927, Washington, DC, USA, 2009. IEEE Computer Society. 1, 4

[20] Wei Lu, Yanyan Shen, Su Chen, and Beng Chin Ooi. **Efficient Processing of K Nearest Neighbor Joins Using MapReduce**. *Proc. VLDB Endow.*, **5**(10):1016–1027, June 2012. 2, 4, 8

[21] Chi Zhang, Feifei Li, and Jeffrey Jestes. **Efficient Parallel kNN Joins for Large Data in MapReduce**. In *Proceedings of the 15th International Conference on Extending Database Technology*, EDBT '12, pages 38–49, New York, NY, USA, 2012. ACM. 2, 4, 8

[22] Rares Vernica, Michael J. Carey, and Chen Li. **Efficient Parallel Set-similarity Joins Using MapReduce**. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, SIGMOD '10, pages 495–506, New York, NY, USA, 2010. ACM. 2, 4, 8

[23] Luis Gravano, Panagiotis G. Ipeirotis, H. V. Jagadish, Nick Koudas, S. Muthukrishnan, and Divesh Srivastava. **Approximate String Joins in a Database (Almost) for Free**. In *Proceedings of the 27th International Conference on Very Large Data Bases*, VLDB '01, pages 491–500, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc. 3, 6, 7

[24] Antonin Guttman. **R-trees: A Dynamic Index Structure for Spatial Searching**. In *Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data*, SIGMOD '84, pages 47–57, New York, NY, USA, 1984. ACM. 5, 14, 23

[25] Ahmed K. Elmagarmid, Panagiotis G. Ipeirotis, and Vassilios S. Verykios. **Duplicate Record Detection: A Survey**. *IEEE Trans. on Knowl. and Data Eng.*, **19**(1):1–16, January 2007. 6

[26] Justin Zobel, Alistair Moffat, and Kotagiri Ramamohanarao. **Inverted Files Versus Signature Files for Text Indexing**. *ACM Trans. Database Syst.*, **23**(4):453–490, December 1998. 6, 12, 17

[27] Einat Amitay, Nadav Har'El, Ron Sivan, and Aya Soffer. **Web-a-where: Geotagging Web Content**. In *Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '04, pages 273–280, New York, NY, USA, 2004. ACM. 7, 12

[28] Junyan Ding, Luis Gravano, and Narayanan Shivakumar. **Computing Geographical Scopes of Web Resources**. In *Proceedings of the 26th International Conference on Very Large Data Bases*, VLDB '00, pages 545–556, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc. 7

[29] Michael D. Lieberman, Hanan Samet, and Jagan Sankaranarayanan. **Geotagging with local lexicons to build indexes for textually-specified**

spatial data. In *Proceedings of the 26th International Conference on Data Engineering, ICDE 2010, March 1-6, 2010, Long Beach, California, USA*, pages 201–212, 2010. 7

[30] Subodh Vaid, Christopher B. Jones, Hideo Joho, and Mark Sanderson. **Spatio-textual Indexing for Geographical Search on the Web**. In *Proceedings of the 9th International Conference on Advances in Spatial and Temporal Databases*, SSTD'05, pages 218–235, Berlin, Heidelberg, 2005. Springer-Verlag. 7

[31] Yen-Yu Chen, Torsten Suel, and Alexander Markowetz. **Efficient Query Processing in Geographic Web Search Engines**. In *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data*, SIGMOD '06, pages 277–288, New York, NY, USA, 2006. ACM. 7

[32] Ramaswamy Hariharan, Bijit Hore, Chen Li, and Sharad Mehrotra. **Processing Spatial-Keyword (SK) Queries in Geographic Information Retrieval (GIR) Systems**. In *Proceedings of the 19th International Conference on Scientific and Statistical Database Management*, SSDBM '07, pages 16–, Washington, DC, USA, 2007. IEEE Computer Society. 7

[33] Yinghua Zhou, Xing Xie, Chuang Wang, Yuchang Gong, and Wei-Ying Ma. **Hybrid Index Structures for Location-based Web Search**. In *Proceedings of the 14th ACM International Conference on Information and Knowledge Management*, CIKM '05, pages 155–162, New York, NY, USA, 2005. ACM. 7

[34] Xin Cao, Gao Cong, Christian S. Jensen, and Beng Chin Ooi. **Collective Spatial Keyword Querying**. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, SIGMOD '11, pages 373–384, New York, NY, USA, 2011. ACM. 7

[35] Anish Das Sarma, Foto N. Afrati, Semih Salihoglu, and Jeffrey D. Ullman. **Upper and lower bounds on the cost of a map-reduce computation**. In *Proceedings of the 39th international conference on Very Large Data Bases*, PVLDB'13, pages 277–288. VLDB Endowment, 2013. 25

[36] Jie Bao, Yu Zheng, and Mohamed F. Mokbel. **Location-based and Preference-aware Recommendation Using Sparse Geo-social Networking Data**. In *Proceedings of the 20th International Conference on Advances in Geographic Information Systems*, SIGSPATIAL '12, pages 199–208, New York, NY, USA, 2012. ACM. 33