Parallel and Distributed Processing of Spatial Preference Queries using Keywords

Christos Doulkeridis[1], Akrivi Vlachou[1], Dimitris Mpestas[1,2], Nikos Mamoulis[2]
[1]Dept. of Digital Systems, University of Piraeus, Greece
[2]Dept. of Computer Science & Engineering, University of Ioannina, Greece
cdoulk@unipi.gr, avlachou@aueb.gr, dmpestas@teemail.gr, nikos@cs.uoi.gr

ABSTRACT

Advanced queries that combine spatial constraints with textual relevance to retrieve objects of interest have attracted increased attention recently due to the ever-increasing rate of user-generated spatio-textual data. Motivated by this trend, in this paper, we study the novel problem of parallel and distributed processing of *spatial preference queries using keywords*, where the input data is stored in a distributed way. Given a set of keywords, a set of spatial data objects and a set of spatial feature objects that are additionally annotated with textual descriptions, the *spatial preference query using keywords* retrieves the top-k spatial data objects ranked according to the textual relevance of feature objects in their vicinity. This query type is processing-intensive, especially for large datasets, since any data objects may belong to the result set while the spatial range defines the score, and the k data objects with the highest score need to be retrieved. Our solution has two notable features: (a) we propose a deliberate re-partitioning mechanism of input data to servers, which allows parallelized processing, thus establishing the foundations for a scalable query processing algorithm, and (b) we boost the query processing performance in each partition by introducing an early termination mechanism that delivers the correct result by only examining few data objects. Capitalizing on this, we implement parallel algorithms that solve the problem in the MapReduce framework. Our experimental study using both real and synthetic data in a cluster of sixteen physical machines demonstrates the efficiency of our solution.

1. INTRODUCTION

With the advent of modern applications that record the position of mobile users by means of GPS, and the extensive use of mobile smartphones, we have entered the era of Big Spatial Data. The fact that an increasing amount of user-generated content (e.g., messages in Twitter, photos in Flickr, etc.) is geotagged also contributes to the daily creation of huge volumes of location-based data. Apart from spatial locations, the data typically contain textual descriptions or annotations. Analyzing and exploiting such textually annotated location-based data is estimated to bring high economic benefits in the near future.

In order to extract useful insights from this wealth of Big Spatial Data, advanced querying mechanisms are required that retrieve results of interest from massively distributed spatio-textual data. In this paper, we study an advanced query type that retrieves data objects based on the textual relevance of other (feature) objects in their spatial neighborhood. In particular, given a keyword-based query, a set of spatial *data objects* and a set of spatial *feature objects* that are additionally annotated with textual descriptions, the *spatial preference query using keywords* retrieves the top-k spatial data objects ranked according to the textual relevance of feature objects in their vicinity. This query is generic, as it can be used to retrieve locations of interest based on the relevance of Tweets in their vicinity, based on popular places (bars, restaurants, etc.), and/or based on the comments of other people in the surrounding area.

However, processing this query raises significant challenges. First, due to the query definition, every data object is a potential result and cannot be pruned by spatial or textual constraints. Second, the distribution of data raises the need to find a way to parallelize computation by assigning units of work that can be processed independently from others. In this paper, we address these technical challenges and provide the first solution to parallel/distributed processing of the spatial preference query using keywords. Our approach has two notable features: (a) we propose a method to parallelize processing by deliberately re-partitioning input data, in such a way that the partitions can be processed in parallel, independently from each other, and (b) within each partition, we apply an early termination mechanism that eagerly restricts the number of objects that need to be processed in order to provide the correct result set.

In more detail, we make the following contributions in this paper:

- We formulate and address a novel problem, namely parallel/distributed evaluation of spatial preference queries using keywords over massive and distributed spatio-textual data.
- We propose a grid-based partitioning method that uses careful duplication of feature objects in selected neighboring cells and allows independent processing of subsets of input data in parallel, thus establishing the foundations for a scalable, parallel query evaluation algorithm.

- We further boost the performance of our algorithm by introducing an early termination mechanism for each independent work unit, thereby reducing the processing cost.

- We demonstrate the efficiency of our algorithms by means of experimental evaluation using both real and synthetic datasets in a medium-sized cluster.

The remainder of this paper is structured as follows: Section 2 presents the preliminary concepts and necessary background. Section 3 formally defines the problem under study and explains the rationale of our approach along with a brief overview. The proposed query processing algorithm that relies on the grid-based partitioning is presented in Section 4. Section 5 presents the algorithms that use early termination. Section 6 analyzes the complexity of the proposed algorithms. Then, in Section 7, we present the results of our experimental evaluation. Related research efforts are outlined in Section 8. Finally, in Section 9, we provide concluding remarks.

2. PRELIMINARIES

In this section we give a brief overview of MapReduce and HDFS, and define the type of queries we will focus on.

2.1 MapReduce and HDFS

Hadoop is an open-source implementation of MapReduce [4], providing an environment for large-scale, fault-tolerant data processing. Hadoop consists of two main parts: the HDFS distributed file system and MapReduce for distributed processing.

Files in HDFS are split into a number of large blocks which are stored on DataNodes, and one file is typically distributed over a number of DataNodes in order to facilitate high bandwidth during parallel processing. In addition, blocks can be replicated to multiple DataNodes (by default three replicas), in order to ensure fault-tolerance. A separate NameNode is responsible for keeping track of the location of files, blocks, and replicas thereof. HDFS is designed for use-cases where large datasets are loaded ("write-once") and processed by many different queries that perform various data analysis tasks ("read-many").

A task to be performed using the MapReduce framework has to be specified as two steps. The *Map* step as specified by a map function takes some input (typically from HDFS files), possibly performs some computation on this input, and redistributes it to worker nodes (a process known as "shuffle"). An important aspect of MapReduce is that both the input and output of the Map step is represented as key-value pairs, and that pairs with same key will be processed as one group by a Reducer. As such, the *Reduce* step receives all values associated with a given key, from multiple map functions, as a result of the re-distribution process, and typically performs some aggregation on the values, as specified by a reduce function.

It is important to note that one can customize the redistribution of data to Reducers by implementing a *Partitioner* that operates on the output key of the map function, thus practically enforcing an application-specific grouping of data in the Reduce phase. Also, the ordering of values in the reduce function can be specified, by implementing a customized Comparator. In our work, we employ such customizations to obtain a scalable and efficient solution to our problem.

Symbol	Description
O	Set of data objects
p	Object in O, $p \in O$
F	Set of feature objects
f	Feature object in F, $f \in F$
$f.\mathcal{W}$	Keywords associated with feature object f
$q(k, r, \mathcal{W})$	Query for top-k data objects
$w(f, q)$	Textual relevance of feature f to query q
$\overline{w}(f, q)$	Upper bound of $w(f, q)$
$dist(p, f)$	Spatial distance between p and f
$\tau(p)$	Score of data object p
$\overline{\tau}$	Score of the k-th best data object
R	Number of Reduce tasks
$\mathcal{C} = \{C_1, \ldots, C_R\}$	Grid cells

Table 1: Overview of symbols.

2.2 Spatial Preference Queries

The spatial preference query belongs to a class of queries that rank objects based on the quality of other (feature) objects in their spatial neighborhood [12, 16, 17]. Inherently a spatial preference query assumes that two types of objects exist: the data objects, which will be ranked and returned by the query, and the feature objects, which are responsible for ranking the data objects. As such, the feature objects determine the score of each data object according to a user-specified metric. Spatial preference queries find more applications in the case of textually annotated feature objects [14], where the score of data objects is determined by a textual similarity function applied on query keywords and textual annotations of feature objects. This query is known as top-k spatio-textual preference query [14]. In this paper, we study a distributed variant of this query.

3. PROBLEM STATEMENT AND OVERVIEW OF SOLUTION

3.1 Problem Formulation

Consider an *object* dataset O of spatial objects $p \in O$, where p is described by its coordinates $p.x$ and $p.y$. Also, consider a *feature* dataset F of spatio-textual objects $f \in F$, which are represented by spatial coordinates $f.x$ and $f.y$, and a set of keywords $f.\mathcal{W}$.

The spatial preference query using keywords returns the k data objects $\{p_1, \ldots, p_k\}$ from O with the highest score. The score of a data object $p \in O$ is defined by the scores of feature objects $f \in F$ in its spatial neighborhood. As already mentioned, each feature object f is associated with a set of keywords $f.\mathcal{W}$. A query q consists of a neighborhood distance threshold r, a set of query keywords $q.\mathcal{W}$ for the feature set F, and the value k that determines how many data objects need to be retrieved. For a quick overview of the basic symbols used in this paper, we refer to Table 1.

Given a query $q(k, r, \mathcal{W})$ and a feature object $f \in F$, we define the *non-spatial score* $w(f, q)$ that indicates the goodness (quality) of f as the similarity of sets $q.\mathcal{W}$ and $t.\mathcal{W}$. In this work, we employ Jaccard similarity for this purpose. Obviously, the domain of values of $w(f, q)$ is the range $[0, 1]$.

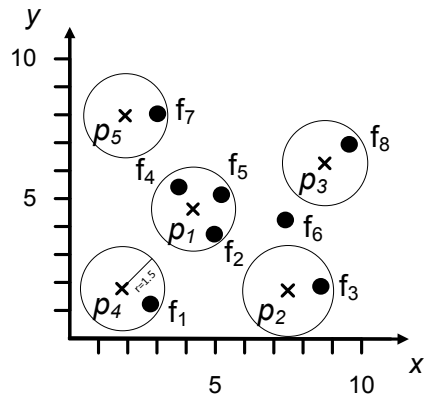


Figure 1: Example of spatial preference query using keywords (SPQ).

Object	X	Y	Keywords	Jaccard
p_1	4.6	4.8	-	-
p_2	7.5	1.7	-	-
p_3	8.9	5.2	-	-
p_4	1.8	1.8	-	-
p_5	1.9	9.0	-	-
f_1	2.8	1.2	italian,gourmet	0.5
f_2	5.0	3.8	chinese,cheap	0
f_3	8.7	1.9	sushi,wine	0
f_4	3.8	5.5	italian	1
f_5	5.2	5.1	mexican,exotic	0
f_6	7.4	5.4	greek,traditional	notInRange
f_7	3.0	8.1	italian,spaghetti	0.5
f_8	9.5	7.0	indian	0

Table 2: Example of datasets and scores for query $q.\mathcal{W} = \{italian\}$.

DEFINITION 1. *(*Non-spatial score $w(f,q)$*): Given a query q and a feature object $f \in F$, the non-spatial score $w(f,q)$ determines the textual relevance between the set of query keywords $q.\mathcal{W}$ and the keywords $f.\mathcal{W}$ of f using Jaccard similarity:*

$$w(f,q) = \frac{|q.\mathcal{W} \cap f.\mathcal{W}|}{|q.\mathcal{W} \cup f.\mathcal{W}|}$$

The score $\tau(p)$ of a data object p is determined by the feature objects that are within distance r from p. More specifically, $\tau(p)$ is defined by the maximum non-spatial score $w(f,q)$ of any feature object f in the r-neighborhood of p. This range-based neighborhood condition is typically used in the related work [12, 14, 16, 17]. Formally,

DEFINITION 2. *The score $\tau(p)$ of p based on feature dataset F, given the range-based neighborhood condition r is defined as:*

$$\tau(p) = max\{w(f,q) \mid f \in F : d(p,f) \leq r\}$$

EXAMPLE 1. *Figure 1 depicts an example. The spatial area contains both data objects (denoted as p_i) and feature objects (denoted as f_i). The data objects represent hotels and the feature objects represent restaurants. Assume a user issues the following query: Find the best (top-k) hotels that have an Italian restaurant nearby. Let us assume that k=1 and "nearby" is translated to $r = 1.5$ units of distance. Then, the query is expressed as: Find the top-1 data object for which a highly ranked feature object exists based on the keyword "italian" and at a distance of at most 1.5 units.*

Table 2 lists the locations and descriptions of both data and feature objects. Only feature objects f_1, f_4 and f_7 have a common term with the user specified query (the keyword "italian"). Thus, only f_1, f_4 and f_7 will have a Jaccard score other than 0. In the last column of Table 2 the Jaccard score for all feature objects is shown. The score of each data object is influenced only by the feature objects within a distance of at most 1.5 units. In Figure 1 the circles with radius 1.5 range units and center each data object include the feature objects that are nearby each data object and influence its score. The actual score of a data object is the highest score of all nearby feature objects. Data object p_4 has a score of 0.5 due to feature object f_1, data object p_1 has a score of 1 because of feature object f_4 and data object p_5 has a score of 0.5 due to feature object f_7. Hence, the top-1 result is object p_1.

In the parallel and distributed setting that is targeted in this paper, datasets O and F are horizontally partitioned and distributed to different machines (servers), which means that each server stores only a fraction (partition) of the entire datasets. In other words, a number of partitions $O_i \in O$ and $F_i \in F$ of datasets O and F respectively exists, such that $\bigcup O_i = O$, $O_i \bigcap O_j = \emptyset$ for $i \neq j$, and $\bigcup F_i = F$, $F_i \bigcap F_j = \emptyset$ for $i \neq j$. Due to horizontal partitioning, any (data or feature) object belongs to a single partition (O_i or F_i respectively). We make no assumption on the number of such partitions nor on having equal number of data and feature object partitions. Also, no assumptions are made on the specific partitioning method used; in fact, our proposed solution is independent of the actual partitioning method employed, which makes it applicable in the most generic case.

PROBLEM 1. *(*Parallel/Distributed Spatial Preference Query using Keywords (SPQ)*) Given an object dataset O and a feature dataset F, which are horizontally partitioned and distributed to a set of servers, the parallel/distributed spatial preference query using keywords returns the k data objects $\{p_1, \ldots, p_k\}$ from O with the highest $\tau(p_i)$ scores.*

3.2 Design Rationale

The spatial preference query using keywords (SPQ) targeted in this paper is a complex query operator, since any data object p may belong to the result set and the spatial range cannot be used for pruning the data space. As a result, the computation becomes more challenging and efficient query processing mechanisms are required that can exploit parallelism and the availability of hardware resources. Parallelizing this query is also challenging because any given data object p and all feature objects within the query range r from p must be assigned to the same server to ensure the correct computation of the score $\tau(p)$ of p. As such, a repartitioning mechanism is required in order to assign (data and feature) objects to servers in a deliberate way that allows local processing at each server. To achieve the desired independent score computation at each server, duplication of feature objects to multiple servers is typically necessary.

Based on this, we set the following objectives for achieving parallel, scalable and efficient query processing:

- **Objective #1:** parallelize processing by breaking the work into independent parts, while minimizing feature object duplication. In addition, the union of the results in each part should suffice to produce the final result set.
- **Objective #2:** avoid processing the input data in its entirety, by providing early termination mechanisms for query processing.

To meet the above objectives, we design our solution by using the following two techniques:

- a grid-partitioning of the spatial data space that uses careful duplication of feature objects in selected neighboring cells, in order to create independent work units (Section 4), and
- sorted access to the feature objects in a deliberate order along with a thresholding mechanism that allows early termination of query processing that guarantees the correctness of the result (Section 5).

4. GRID-BASED PARTITIONING AND INITIAL ALGORITHM

In this section, we present an algorithm for solving the parallel/distributed spatial preference query using keywords, which relies on a grid-based partitioning of the 2-dimensional space in order to identify subsets of the original data that can be processed in parallel. To ensure correctness of the result computed in parallel, we re-partition the input data to grid cells and deliberately duplicate some feature objects in neighboring grid cells. As a result, this technique lays the foundation for parallelizing the query processing and leads to the first scalable solution.

4.1 Grid-based Partitioning

Consider a regular, uniform grid in the 2-dimensional data space that consists of R cells: $\mathcal{C} = \{C_1, \ldots, C_R\}$. Our approach assigns all data and feature objects to cells of this grid, and expects each cell to be processed independently of the other cells. In MapReduce terms, we assign each cell to a single processing task (Reducer), thus all data that is mapped to this cell need to be sent to the assigned Reducer.

The re-partitioning mechanism operates in the following way. Based on its spatial location, an object (data or feature object) is assigned to the cell that encloses it in a straightforward way. However, some feature objects must be additionally assigned to other cells (i.e., duplicated), in order to ensure that the data in each cell can indeed be processed independently of the other cells and produce the correct result. More specifically, given a feature object $f \in C_j$ and any grid cell C_i ($C_i \neq C_j$), we denote by $MINDIST(f, C_i)$ the minimum distance between feature object f and C_i. This distance is defined as the distance of f to the nearest edge of C_i, since f is outside C_i. When this minimum distance is smaller than the query radius r, i.e., $MINDIST(f, C_i) \leq r$, then it is possible that f is within distance r from a data object $p \in C_i$. Therefore, f needs to be assigned (duplicated) also to cell C_i. The following lemma guarantees the correctness of the afore-described technique.

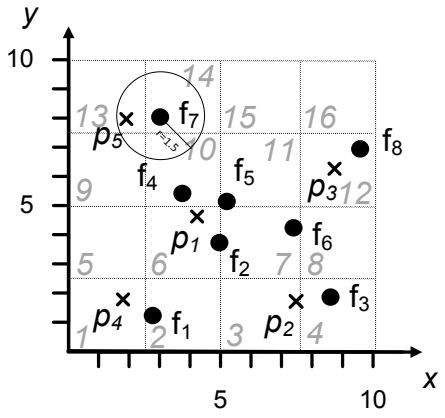


Figure 2: Example of grid partitioning.

LEMMA 1. *(Correctness) Given a parallel/distributed spatial preference query using keywords with radius r, any feature object $f \in C_j$ must be assigned to all other grid cells $C_i (C_i \neq C_j)$, if $MINDIST(f, C_i) \leq r$.*

Figure 2 illustrates the same dataset as in Figure 1 and a 4x4 grid (the numbering of the cells is shown in the figure). Consider a query with radius $r = 1.5$, and let us examine feature object f_7 as an example. Assuming that f_7 has at least one common term in its keyword set ($f_7.\mathcal{W}$) with the user specified query ($q.\mathcal{W}$), then f_7 may affect neighboring cells located near cell with identifier C_{14}. It is fairly easy to see that f_7 needs to be duplicated to cells C_9, C_{10}, and C_{13}, for which $MINDIST(f_7, C_i) \leq r$, thus the score of data objects located in these cells may be determined by f_7.

Before presenting the algorithm that exploits this grid partitioning, we make a note on how to select an appropriate grid size, as this affects the amount of duplication required. It should also be noted that in our approach the grid is defined at query time, after the value of r is known. Let α denote the length of the edge of a grid cell. For now, we should ensure that $\alpha \geq r$, otherwise excessive replication to neighboring cells would be performed. Later, in Section 6, we provide a thorough analysis on the effect of the grid cell size to the amount of duplicated data.

4.2 Parallel Algorithm

We design a parallel algorithm, termed *pSPQ*, that solves the problem in MapReduce. The Map phase is responsible for re-partitioning the input data based on the grid introduced earlier. Then, in the Reduce phase, the problem of reporting the top-k data objects is solved in each cell independently of the rest. This is the part of the query that dominates the processing time; the final result is produced by merging the k results of each of the R cells and returning the top-k with the highest score. However, this last step can be performed in a centralized way without significant overhead, given that the number of these results is small because k is typically small.

In more detail, in the Map phase, each Map task (Mapper) receives as input some data objects and some feature objects, without any assumptions on their location. Each Mapper is responsible for assigning data and feature objects to grid cells, including duplicating feature objects. Each grid cell

Algorithm 1 *pSPQ*: Map Function

```
1: Input: q(k, r, W), grid cells C = {C_1, ..., C_R}
2: function MAP(x: input object)
3: C_i ← {C_i : C_i ∈ C and x enclosed in C_i}
4: if x is a data object then
5:    x.tag ← 0
6:    output ⟨(i, x.tag), x⟩
7: else
8:    x.tag ← 1
9:    if (x.W ∩ q.W ≠ ∅) then
10:      output ⟨(i, x.tag), x⟩
11:      for (C_j ∈ C, such that MINDIST(x, C_j) ≤ r) do
12:         output ⟨(j, x.tag), x⟩
13:      end for
14:   end if
15: end if
16: end function
```

corresponds to a single Reduce task, which will take as input all objects assigned to the respective grid cell. Then, the Reducer can accurately compute the score of any data object located in the particular grid cell and report the top-k.

4.2.1 Map Phase

Algorithm 1 shows the pseudo-code of the Map phase, where each call of the Map function processes a single object denoted by x, which can be a data object or a feature object. First, in line 3, the cell C_i that encloses object x is determined. Then, if x is a data object, it is tagged ($x.tag$) with the value 0, otherwise with the value 1. In case of a data object, x is output using a *composite key* that consists of the cell id i and the tag as key, and as value the entire data object x. In case of a feature object, we apply a simple pruning rule (line 9) to eliminate feature objects that do not affect the result of the query. This rule practically eliminates from further processing any feature object that has no common keyword with the query keywords, i.e., $q.\mathcal{W} \cap f.\mathcal{W} = \emptyset$. The reason is that such feature objects cannot contribute to the score of any data object, based on the definition of our query. This pruning rule can significantly limit the number of feature objects that need to be sent to the Reduce phase. For the remaining feature objects that have at least one common keyword with the query, they are first output with the same composite key as above, and value the entire feature object x. In addition, we identify neighboring cells C_j that comply with Lemma 1, and replicate the feature object in those cells too. In this way, we have partitioned the initial data to grid cells and have performed the necessary duplication of feature objects.

The output key-values of the Map phase are grouped by cell id and assigned to Reduce tasks using a customized Partitioner. Also, in each Reduce task, we order the objects within each group by their tag, so that data objects precede feature objects. This is achieved through the use of the composite keys for sorting. As a result, it is guaranteed that each Reducer accesses the feature objects after it has accessed all data objects.

4.2.2 Reduce Phase

As already mentioned, a Reduce task processes all the data assigned to a single cell and reports the top-k data objects within the respective cell. The pseudo-code of the Reduce function is depicted in Algorithm 2. First, all data objects are accessed one-by-one and loaded in memory (O_i). Moreover, a sorted list L_k of the k data objects p_i with higher scores $\tau(p_i)$ is maintained. Let $\overline{\tau}$ denote the k-th best score of any data object so far. Then, for each feature object x accessed, its non-spatial score $w(x, q)$ (i.e., textual similarity to the query terms) is compared to $\overline{\tau}$. Only if the non-spatial score $w(x, q)$ is larger than $\overline{\tau}$ (line 9), may the top-k list of data objects be updated. Therefore, in this case we test all combinations of x with the data objects p kept in memory O_i. If such a combination (x, p) is within distance r (line 11), then we check if the temporary score of p denoted by $score(p)$ can be improved based on x (i.e., $w(x, q)$), and if that is the case we check whether p has obtained a score that places it in the current top-k list of data objects (L_k). Line 12 shows how the score can be improved, however we omit from the pseudo-code the check of score improvement of p for sake of simplicity. Then, in line 13, the list L_k is updated. As explained, this update is needed only if the score of p is improved. In this case, if p already exists in L_k we only update its score, otherwise p is inserted into L_k. After all feature objects have been processed, L_k contains the top-k data objects of this cell.

Algorithm 2 *pSPQ*: Reduce Function

```
1: Input: q(k, r, W)
2: function REDUCE(key, V: objects assigned to cell
   with id key)
3: L_k ← ∅
4: for (x ∈ V) do
5:    if x is a data object then
6:       Load x in memory O_i
7:       score(x) ← 0 // initial score
8:    else
9:       if w(x, q) > τ̄ then
10:         for (p ∈ O_i) do
11:            if d(p, x) ≤ r then
12:               score(p) ← max{score(p), w(x, q)}
13:               update list L_k of top-k data objects and τ̄
14:            end if
15:         end for
16:      end if
17:   end if
18: end for
19: for p ∈ L_k do
20:    output ⟨p, score(p)⟩ // at this point:score(p) = τ(p)
21: end for
22: end function
```

4.2.3 Limitations

The above algorithm provides a correct solution to the problem in a parallel manner, thus achieving Objective #1. However, in each Reducer, it needs to process the entire set of feature objects in order to produce the correct result. In the following section, we present techniques that overcome this limitation, thereby achieving significant performance gains.

5. ALGORITHMS WITH EARLY TERMINATION

Even though the technique outlined in the previous section

Algorithm 3 *eSPQlen*: Map Function (Section 5.1)

```
1: Input: q(k, r, W), grid cells C = {C_1, ..., C_R}
2: function MAP(x: input object)
3: C_i ← {C_i : C_i ∈ C and x enclosed in C_i}
4: if x is a data object then
5:    output ⟨(i, 0), x⟩
6: else
7:    if (x.W ∩ q.W ≠ ∅) then
8:       output ⟨(i, |x.W|), x⟩
9:       for (C_j ∈ C, such that MINDIST(x, C_j) ≤ r) do
10:         output ⟨(j, |x.W|), x⟩
11:      end for
12:   end if
13: end if
14: end function
```

Algorithm 4 *eSPQlen*: Reduce Function with Early Termination (Section 5.1)

```
1: Input: q(k, r, W)
2: function REDUCE(key, V: objects assigned to cell
   with id key)
3: L_k ← ∅
4: for (x ∈ V) do
5:    if x is a data object then
6:       Load x in memory O_i
7:       score(x) ← 0 // initial score
8:    else
9:       if τ̄ ≥ w̄(x, q) then
10:         break
11:      end if
12:      if w(x, q) > τ̄ then
13:         for (p ∈ O_i) do
14:            if d(p, x) ≤ r then
15:               score(p) ← max{score(p), w(x, q)}
16:               update list L_k of top-k data objects and τ̄
17:            end if
18:         end for
19:      end if
20:   end if
21: end for
22: for p ∈ L_k do
23:    output ⟨p, score(p)⟩ // at this point:score(p) = τ(p)
24: end for
25: end function
```

enables parallel processing of independent partitions to solve the problem, it cannot guarantee good performance since it requires processing both data partitions in a cell in their entirety (including duplicated feature objects). To alleviate this shortcoming, we introduce two alternative techniques that achieve *early termination*, i.e., report the correct result after accessing all data objects but only few feature objects. This is achieved by imposing a deliberate order for accessing feature objects in each cell, which in turn allows determining an upper bound for the score of any unseen feature object. When this upper bound cannot improve the score of the current top-k object, we can safely terminate processing of a given Reducer.

5.1 Accessing Feature Objects by Increasing Keyword Length

The first algorithm that employs early termination, termed *eSPQlen*, is based on the intuition that feature objects f with long textual descriptions that consist of many keywords ($|f.\mathcal{W}|$) are expected to produce low scores $w(f, q)$. This is due to the Jaccard similarity used in the definition of $w(f, q)$ (Defn. 1), which has $|q.\mathcal{W} \bigcup f.\mathcal{W}|$ in the denominator. Based on this, we impose an ordering of feature objects in each Reducer by increasing keyword length, aiming at examining first feature objects that will produce high score values $w(f, q)$ with higher probability.

In more details, given the keywords $q.\mathcal{W}$ of a query q, and a feature object f with keywords $f.\mathcal{W}$, we define a bound for the best possible Jaccard score that this feature object can achieve as:

$$\overline{w}(f, q) = \begin{cases} 1 & , |f.\mathcal{W}| < |q.\mathcal{W}| \\ \frac{|q.\mathcal{W}|}{|f.\mathcal{W}|} & , |f.\mathcal{W}| \geq |q.\mathcal{W}| \end{cases} \quad (1)$$

Given that feature objects are accessed by increased keyword length, this bound is derived as follows. As long as feature objects f are accessed that satisfy $|f.\mathcal{W}| < |q.\mathcal{W}|$, it is not possible to terminate processing, thus the bound takes the value of 1. The reason is that when $|f.\mathcal{W}| < |q.\mathcal{W}|$ holds, it is possible that a subsequent feature object f' with more keywords than f may have higher Jaccard score than f. However, as soon as it holds that $|f.\mathcal{W}| \geq |q.\mathcal{W}|$ the bound (best possible score) equals:

$$\frac{\min\{|q.\mathcal{W}|, |f.\mathcal{W}|\}}{\min\{|q.\mathcal{W}|, |f.\mathcal{W}|\} + |f.\mathcal{W}| - |q.\mathcal{W}|} = \frac{|q.\mathcal{W}|}{|f.\mathcal{W}|}$$

because in the best case the intersection of sets $q.\mathcal{W}$ and $f.\mathcal{W}$ will be equal to: $\min\{|q.\mathcal{W}|, |f.\mathcal{W}|\}$, while their union will be equal to: $\min\{|q.\mathcal{W}|, |f.\mathcal{W}|\} + |f.\mathcal{W}| - |q.\mathcal{W}|$.

Recall that $\overline{\tau}$ denotes the k-th best score of any data object so far. Then, the condition for early termination during processing of feature objects by increasing keyword length, can be stated as follows:

LEMMA 2. *(Correctness of Early Termination* eSPQlen*) Given a query q and an ordering of feature objects based on increasing number of keywords, it is safe to stop accessing more feature objects as soon as a feature object f is accessed with:*

$$\overline{\tau} \geq \overline{w}(f, q)$$

Based on this analysis, we introduce a new algorithm that follows the paradigm of Section 4, but imposes the desired access order to feature objects and is able to terminate early in the Reduce phase.

5.1.1 Map Phase

Algorithm 3 describes the Map phase of the new algorithm. The main difference to the algorithm described in Section 4 is in the use of the composite key when objects are output by the Map function (lines 8 and 10). The composite key contains two parts. The first part is the cell id, as previously, but the second part is a number. The second part corresponds to the value zero in the case of data objects, while it corresponds to the length $|f.\mathcal{W}|$ of the keyword description in the case of a feature object f. The rationale behind the use of this composite key is that the cell id is going to be used to group objects to Reducers, while the second part of the key is going to be used to establish the

Algorithm 5 *eSPQsco*: Map Function (Section 5.2)

```
1: Input: q(k, r, W), grid cells C = {C_1, ..., C_R}
2: function MAP(x: input object)
3: C_i ← {C_i : C_i ∈ C and x enclosed in C_i}
4: if x is a data object then
5:    output ⟨(i, 2), x⟩
6: else
7:    if (x.W ∩ q.W ≠ ∅) then
8:       output ⟨(i, w(x, q)), x⟩
9:       for (C_j ∈ C, such that MINDIST(x, C_j) ≤ r) do
10:         output ⟨(j, w(x, q)), x⟩
11:      end for
12:   end if
13: end if
14: end function
```

ordering in the Reduce phase in increased order of the number used. In this sorted order, data objects again precede feature objects, due to the use of the zero value. Between two feature objects, the one with the smallest length of keyword description (i.e., fewer keywords) precedes the other in the sorted order.

5.1.2 Reduce Phase

As already mentioned, feature objects with long keyword lists are expected to result in decreased textual similarity (in terms of Jaccard value). Thus, our hope is that after accessing feature objects with few keywords, we will find a feature object that has so many keywords that all remaining feature objects in the ordering cannot surpass the score of k-th best data object thus far.

Algorithm 4 explains the details of our approach. Again, only the set of data objects assigned to this Reducer is maintained in memory, along with a sorted list L_k of the k data objects with best scores found thus far in the algorithm. The condition for early termination is based on the score $\overline{\tau}$ of the k-th object in list L_k and the best potential score $\overline{w}(f, q)$ of the current feature object f (line 9).

5.2 Accessing Feature Objects by Decreasing Score

In this section, we introduce an even better early termination algorithm, termed *eSPQsco*. The rationale of this algorithm is to compute the Jaccard score in the Map phase and use this score as second part of the composite key. In essence, this can enforce a sorted order in the Reduce phase where feature objects are accessed from the highest scoring feature object to the lowest scoring one.

To explain the intuition of the algorithm, consider the feature object with highest score. Any data object located within distance r from this feature object is guaranteed to belong to the result set, as no other data object can acquire a higher score. This observation leads to a more efficient algorithm that can (in principle) produce results when accessing even a single feature object. As a result, the algorithm is expected to terminate earlier, by accessing only a handful of feature objects. This approach incurs additional processing cost at the Map phase (i.e., computation of the Jaccard score), but the imposed overhead to the overall execution time is minimal.

LEMMA 3. *(Correctness of Early Termination* eSPQsco*)*

Algorithm 6 *eSPQsco*: Reduce Function with Early Termination (Section 5.2)

```
1: Input: q(k, r, W)
2: function REDUCE(key, V: objects assigned to cell
   with id key)
3: for (x ∈ V) do
4:    if x is a data object then
5:       Load x in memory O_i
6:    else
7:       if ∃p ∈ O_i : d(p, x) ≤ r then
8:          output ⟨p, w(x, q)⟩ // here: w(x, q) = τ(p)
9:          cnt++
10:         if (cnt = k) then
11:            break
12:         end if
13:      end if
14:   end if
15: end for
16: end function
```

Given a query q and an ordering of feature objects $\{f_i\}$ based on decreasing score $w(f_i, q)$, it is safe to stop accessing more feature objects as soon as k data objects are retrieved within distance r from any already accessed feature object.

5.2.1 Map Phase

Algorithm 5 describes the Map function, where the only modifications are related to the second part of the composite key (lines 5, 8, and 10). In the case of data objects, this must be set to a value strictly higher than any potential Jaccard value, i.e., it can be set equal to 2, since the Jaccard score is bounded in the range $[0, 1]$. Thus, data objects will be accessed before any feature object. In the case of a feature object f, it is set to the Jaccard score $w(f, q)$ of f with respect to the query q. Obviously, the customized Comparator must also be changed in order to enforce the ordering, from highest scores to lowest scores.

5.2.2 Reduce Phase

Algorithm 6 details the operation of the Reduce phase. After all data objects are loaded in memory, the feature objects are accessed in decreasing order of their Jaccard score to the query. For each such feature object f, any data object located within distance r is immediately reported as a result within the specific cell. As soon as k data objects have been reported as results, the algorithm can safely terminate.

6. THEORETICAL RESULTS

In this section, we analyze the space and time complexity in the Reduce phase, which relate to the number of cells and the number of duplicate objects.

6.1 Complexity Analysis

Let R denote the number of Reducers, which is also equivalent to the number of grid cells. Further, let O_i and F_i denote the subset of the data and feature objects assigned to the i-th Reducer respectively. Notice that F_i contains both the feature objects enclosed in the cell corresponding to the i-th Reducer, as well as the duplicated feature objects that are located in other neighboring cells. In other words, it holds that $\bigcup_{i=1}^{R} |F_i| \geq |F|$.

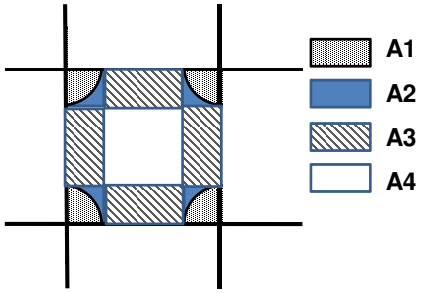


Figure 3: Breaking a cell in areas based on the number of duplicates.

In the case of the initial parallel algorithm that does not use early termination (Section 4), a Reducer needs to store in memory the data objects $|O_i|$ and the list of k data objects with best scores, leading to space complexity: $O(|O_i|)$ since $|O_i| >> k$. On the other hand, the time complexity is: $O(|O_i| \cdot |F_i|)$, since in worst case for all data objects and for each feature object the score is computed. In practice, when using the early termination the processing cost of each Reducer is significantly smaller, since only few feature objects need to be examined before reporting the correct result set.

If we make the simplistic assumption that the work is shared fairly in the R Reducers (e.g., assuming uniform distribution and a regular uniform grid), then we can replace in the above formulas: $|O_i| = \frac{|O|}{R}$. Let us also consider the *duplication factor* d_f of the feature dataset F, which is a real number that is grid-dependent and data-dependent, such that: $\bigcup_{i=1}^{R} |F_i| = d_f \cdot |F|$. Then, we can also replace in the above formulas: $|F_i| = \frac{d_f \cdot |F|}{R}$. Thus, the processing cost of a Reducer is proportional to: $|O_i| \cdot |F_i| = \frac{|O|}{R} \cdot \frac{d_f \cdot |F|}{R}$.

6.2 Estimation of the Duplication Factor

In the following, we assume that the size a of each side of a grid cell is larger than twice the query radius, i.e., $a \geq 2r$, or equivalently $r \leq \frac{a}{2}$. This is reasonable, since we expect r to be smaller than the size of a grid cell.

Depending on the area where a feature object is positioned, different number of duplicates of this object will be created. Figure 3 shows an example of a grid cell. Given a feature object enclosed into a cell, we identify four different cases. If the feature object has a distance smaller than or equal to r from any cell corner then the feature object is enclosed in the area A_1 that is depicted as the dotted area. In this case, the feature object must be duplicated to all three neighboring cells to the corner of the cell. If the feature object has a distance smaller than or equal to r from two cell borders but not from a cell corner then the feature object is enclosed in area A_2. This area is depicted with solid blue color defined by the four rectangles, but does not include the circles. If located in A_2, then only 2 duplicates will be created (not on the diagonal cell). The third area is A_3 depicted as dashed area and corresponds to the feature object that have a distance smaller than or equal to r from only one border of the cell. In this case, only one duplicate is needed. Finally, if the feature object is enclosed in the remaining area of the cell (white area, called area A_4), no duplication is needed.

Obviously, since it holds $r \leq \frac{a}{2}$ any feature object that belongs to a cell is located in only one of these four areas. Let $|A_i|$ denote the surface of area A_i, and A denote the area of the complete cell. Then:

- $|A_1| = 4 \cdot \frac{\pi r^2}{4} = \pi r^2$
- $|A_2| = 4 \cdot r^2 - |A_1| = (4 - \pi) r^2$
- $|A_3| = 4 \cdot (a - 2r) r$
- $|A_4| = a^2 - |A_1| - |A_2| - |A_3| = (a - 2r)^2$
- $|A| = a^2$

Let $P(A_i)$ denote the probability that a feature object belongs to area A_i. Then, if we assume uniform distribution of feature objects in the space, we obtain the following probabilities: $P(A_i) = \frac{|A_i|}{|A|}$. Based on this, given n feature objects enclosed in a cell, we can calculate the total number of feature objects (including duplicates), denoted by $\widehat{n}$, of the n feature points:

$$\widehat{n} = 3 \cdot n \cdot P(A_1) + 2 \cdot n \cdot P(A_2) + n \cdot P(A_3) + n$$

and we can calculate the duplication factor d_f for this cell:

$$d_f = \frac{\widehat{n}}{n} =$$
$$3 \cdot P(A_1) + 2 \cdot P(A_2) + P(A_3) + 1 =$$
$$3\frac{\pi r^2}{a^2} + 2\frac{(4-\pi) r^2}{a^2} + \frac{4 \cdot (a - 2 \cdot r) r}{a^2} + 1 =$$
$$\frac{\pi r^2}{a^2} + \frac{4 \cdot r}{a} + 1$$

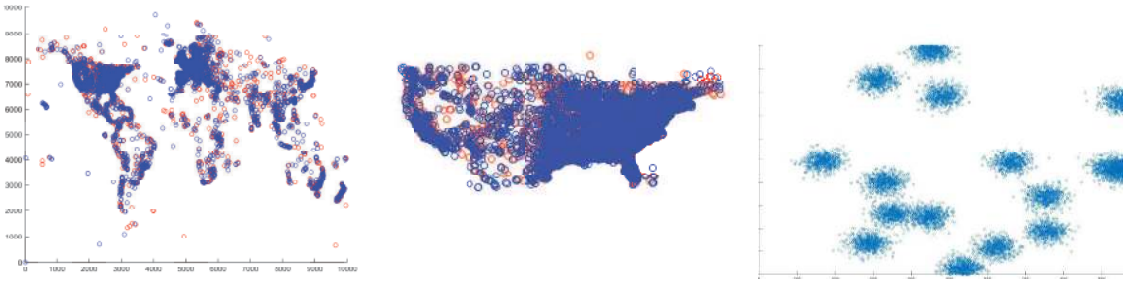
Based on the above formula, we conclude that the worst value of d_f is $3 + \frac{\pi}{4}$ for the case of $a = 2 \cdot r$ and it holds that $1 \leq d_f \leq 3 + \frac{\pi}{4}$ for any query range r such that $a \leq 2 \cdot r$. Also, the duplication factor depends only on the ratio of the cell size to the query range, under the assumption of uniform distribution. Moreover, the formula shows that smaller cell size α (compared to the query range r) increases the number of duplicated feature objects. Put differently, a larger cell size α reduces the duplication of feature objects.

6.3 Analysis of the Cell Size

Even though using a larger cell size α reduces the total number of feature objects, it also has significant disadvantages. First, it results in fewer cells thus reducing parallelism. Second, the probability of obtaining imbalanced partitions in the case of skewed data is increased. Let us assume that all R cells can be processed in a single round, i.e., the hardware resources are sufficient to launch R Reduce tasks in parallel[1]. In this case, the total processing time depends on the performance of one Reducer, which as mentioned before depends on $|O_i| \cdot |F_i| = \frac{|O|}{R} \cdot \frac{d_f \cdot |F|}{R} = |O| \cdot |F| \cdot \frac{d_f}{R^2}$. If we normalize the dataset in $[0, 1] \times [0, 1]$, then $\alpha \leq 1$ and $R = \frac{1}{a}$. Then, $|O_i| \cdot |F_i| = |O| \cdot |F| \cdot d_f \cdot a^4$. In order to study the performance of one Reducer while varying a, it is sufficient to study $d_f \cdot a^4$ since the remaining factors are constant.

Based on the estimation of d_f in the previous section, $d_f \cdot a^4 = (\frac{\pi r^2}{a^2} + \frac{4 \cdot r}{a} + 1) \cdot a^4 = \pi \cdot r^2 \cdot a^2 + 4 \cdot r \cdot a^3 + a^4$. If we consider r as a constant, then for increasing positive values of a, the value of the previous equation increases, which means that the complexity of the algorithm increases. Thus, a smaller cell size α increases the number of cells

[1]We make this assumption to simplify the subsequent analysis, but obviously the number of cores can be smaller than the number of grid cells, and in this case a Reducer will process multiple cells.



(a) Flickr (FL). (b) Twitter (TW). (c) Clustered (CL).

Figure 4: Illustration of spatial distribution of datasets.

and parallelism, and also reduces the processing cost of each Reducer.

7. EXPERIMENTAL EVALUATION

In this section, we report the results of our experimental study. All algorithms are implemented in Java.

7.1 Experimental Setup

Platform. We deployed our algorithms in an in-house CDH cluster consisting of sixteen (16) server nodes. Each of the nodes d1-d8 has 32GB of RAM, 2 disks for HDFS (5TB in total) and 2 CPUs with a total of 8 cores running at 2.6 GHz. The nodes d9-d12 have 128GB of RAM, 4 disks for HDFS (8TB in total) and 2 CPUs with a total of 12 cores (24 hyperthreads) running at 2.6 GHz. Finally, each of the nodes d13-d16 is equipped with 128GB RAM, 4 disks for HDFS (8TB in total), and 2 CPUs with a total of 16 cores running at 2.6GHz. Each of the servers in the cluster function as DataNode and NodeManager, while one of them in addition functions as NameNode and ResourceManager. Each node runs Ubuntu 12.04. We use the CDH 5.4.8.1 version of Cloudera and Oracle Java 1.7. The JVM heap size is set to 1GB for Map and Reduce tasks. We configure HDFS with 128MB block size and replication factor of 3.

Datasets. In order to evaluate the performance of our algorithms we used four different large-scale datasets. Two real datasets are included, a dataset of tweets obtained from Twitter and a dataset of images obtained from Flickr. The Twitter dataset (TW) was created by extracting approximately 80 million tweets which requires 5.7GB on disk. Besides a spatial location, each tweet contains several keywords extracted from its text, with 9.8 keywords on average per tweet, while the size of the dictionary is 88,706 keywords. The Flickr dataset (FL) contains metadata of approximately 40 million images, thus capturing 3.5GB on disk. The average number of keywords per image is 7.9 and the dictionary contains 34,716 unique keywords.

In addition, we created two synthetic datasets in order to test the scalability of our algorithms with even larger datasets. The first synthetic dataset consists of 512 million spatial (data and feature) objects that follow a uniform (UN) distribution in the data space. Each feature object is assigned with a random number of keywords between 10 and 100, and these keywords are selected from a vocabulary of size 1,000. The total size of the file is 160GB. The second synthetic dataset follows a clustered (CL) distribution. We generate 16 clusters whose position in space is selected at random. All other parameters are the same. The total size of the generated dataset is 160GB, as in the case of UN. Figure 4 depicts the spatial distribution of the datasets employed in our experimental study. In all cases, we randomly select half of the objects to act as data objects and the other half as feature objects.

Parameter	Values
Datasets	Real: {TW, FL} Synthetic: {UN, CL}
Query keywords ($\|q.\mathcal{W}\|$)	1, **3**, 5, 10
Query radius (r) (% of side α of grid cell)	5%, **10%**, 25%, 50%
Top-k	5, **10**, 50, 100
Grid size (FL, TW)	35x35, **50x50**, 75x75, 100x100
Grid size (UN, CL)	10x10, **15x15**, 50x50, 100x100

Table 3: Experimental parameters (default values in bold).

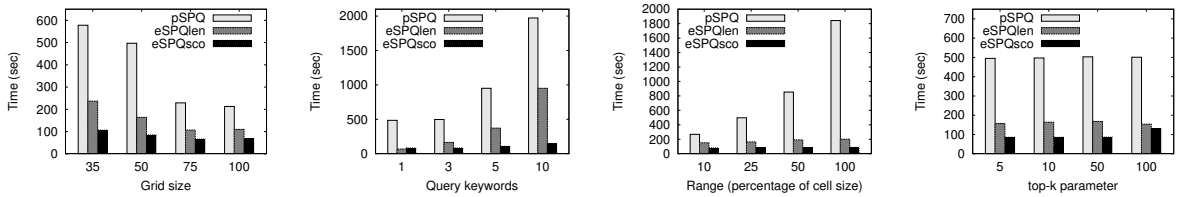
Algorithms. We compare the performance of the following algorithms that are used to compute the spatial preference query using keywords in a distributed and parallel way in Hadoop:

- *pSPQ*: the parallel grid-based algorithm without early termination (Section 4),
- *eSPQlen*: the parallel algorithm that uses early termination by accessing feature objects based on increasing keyword length (Section 5.1), and
- *eSPQsco*: the parallel algorithm that uses early termination by accessing feature objects based on decreasing score (Section 5.2).

For clarification purposes, we note that centralized processing of this query type is infeasible in practice, due to the size of the underlying datasets and the time necessary to build centralized index structures.

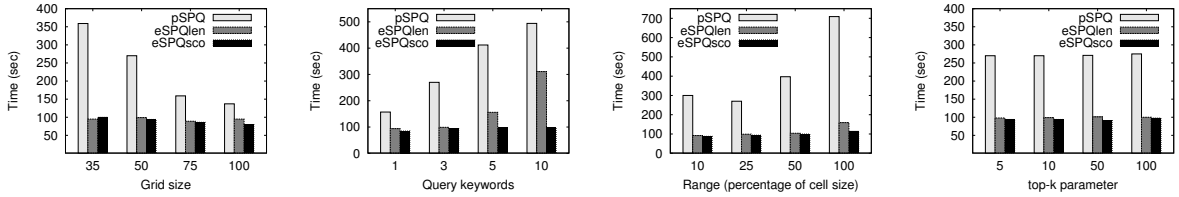
Query generation. Queries are generated by selecting various values for the query radius r and a number of random query keywords $q.\mathcal{W}$ from the vocabulary of the respective dataset[2].

[2]We also explored alternative methods for keyword selection instead of random selection, such as selecting from the most frequent words or the least frequent words, but the execution time of our algorithms was not significantly affected.



(a) Varying grid size. (b) Varying number of query keywords. (c) Varying query radius. (d) Varying k.

Figure 5: Experiments for Flickr (FL) dataset.



(a) Varying grid size. (b) Varying number of query keywords. (c) Varying query radius. (d) Varying k.

Figure 6: Experiments for Twitter (TW) dataset.

Parameters. During the experimental evaluation a number of parameters were varied in order to observe their effect on each algorithm's runtime. These parameters, reported in Table 3, are: (i) the radius of the query, (ii) the number of keywords of the query, (iii) the size of the grid that we use to partition the data space, (iv) the number of the k results that the algorithm returns, and (v) the size of the dataset. In all cases, the number of Reducers is set equal to the number of cells in the spatial grid.

Metrics. The algorithms are evaluated by the time required for the MapReduce job to complete, i.e., the job execution time.

7.2 Experimental Results

7.2.1 Experiments with Real Data: Flickr

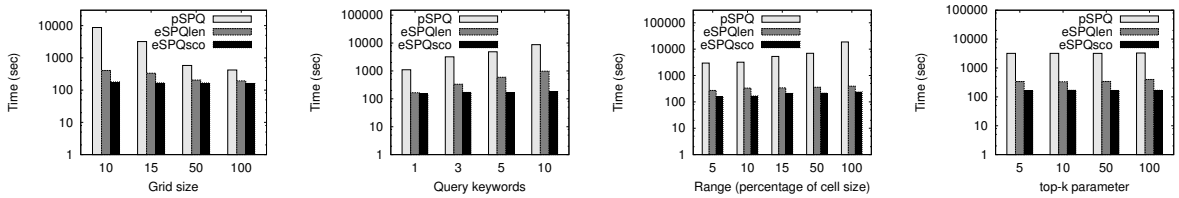
Figure 5 presents the results obtained for the Flickr (FL) dataset. First, in Figure 5(a), we study the effect of grid size to the performance of our algorithms. The first observation is that using more grid cells (i.e., Reduce tasks) improves the performance, since more, yet smaller, parts of the problem need to be computed. The algorithms that employ early termination (*eSPQlen*, *eSPQsco*) are consistently much faster than the grid-based algorithm *pSPQ*. In particular, *eSPQsco* improves *pSPQ* up to a factor of 6x. Between the early termination algorithms, *eSPQsco* is consistently faster due to the sorting based on score, which typically needs to access only a handful of feature objects before reporting the correct result. Figure 5(b) shows the effect of varying the number of query keywords ($|q.\mathcal{W}|$). In general, when more keywords are used in the query more feature objects are passed to the Reduce phase, since the probability of having non-zero Jaccard similarity increases. This is more evident in *pSPQ*, whose cost increases substantially with increased query keyword length. Instead, *eSPQsco* is not significantly affected by the increased number of keywords, because it still manages to find the correct result after examining only few feature objects. This experiment demonstrates the value of the early termination criterion employed in *eSPQsco*. In Figure 5(c), we gradually increase the radius of the query. In principle, this makes query processing more costly as again more feature objects become candidates for determining the score of any data object. However, the early termination algorithms are not significantly affected by increased values of radius, as they can still report the correct result after examining few feature objects only. Finally, in Figure 5(d), we study the effect on increased values of top-k. The chart shows that all algorithms are not particularly sensitive to increased values of k, because the cost of reporting a few more results is marginal compared to the work needed to report the first result.

7.2.2 Experiments with Real Data: Twitter

Figure 6 depicts the results obtained in the case of the Twitter (TW) dataset. In general, the conclusions drawn are quite similar to the case of the FL dataset. The algorithms that employ early termination, and in particular *eSPQsco*, scale gracefully in all setups. Even in the harder setups of many query keywords (Figure 6(b)) and larger query radius (Figure 6(c)), the performance of *eSPQsco* is not significantly affected. This is because as soon as the first few feature objects with highest scores are examined, the algorithm can safely report the top-k data objects in the cell. In other words, the vast majority of feature objects assigned to a cell are not actually processed, and exactly this characteristic makes the algorithm scalable both in the case of more demanding queries as well as in the case of larger datasets.

7.2.3 Experiments with Uniform Data

In order to study the performance of our algorithms for large-scale datasets, we employ in our study synthetic datasets. Figure 7 presents the results obtained for the Uniform (UN) dataset. Notice the log-scale used in the y-axis. A general observation is that *eSPQsco* that uses early termination out-



(a) Varying grid size. (b) Varying number of query keywords. (c) Varying query radius. (d) Varying k.

Figure 7: Experiments for Uniform (UN) dataset.

performs *pSPQ* by more than one order of magnitude. This is a strong indication in favor of the algorithms employing early termination, as their performance gains are more evident for larger datasets, such as the synthetic ones. Also, the general trends are in accordance with the conclusions derived from the real datasets. It is noteworthy that the performance of *eSPQsco* remains relatively stable in the harder setups consisting of many query keywords (Figure 7(b)) and larger query radius (Figure 7(c)).

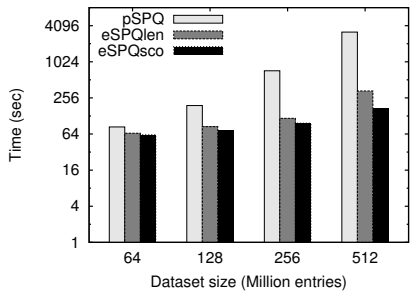


Figure 8: Scalability of all algorithms.

Moreover, Figure 8 shows the results obtained when we vary the dataset size. This experiment aims at demonstrating the nice scaling properties of our algorithms. In particular, *pSPQ* scales linearly with increased dataset size, which is already a good result. However, the algorithms that employ early termination perform much better, since they only examine few feature objects regardless of the increase of dataset size. The experiment also shows that the gain of the algorithms that employ early termination compared to *pSPQ* increases for larger datasets.

7.2.4 Experiments with Clustered Data

Figure 9 presents the results obtained for the Clustered (CL) dataset. It should be noted that such a data distribution is particularly challenging as: (a) it is hard to fairly assign the objects to Reducers, thus typically some Reducers are overburdened, and (b) excessive object duplication can occur when a cluster is located on grid cell boundaries. For the CL dataset, we observed that *pSPQ* results in extremely high execution time, thus it is not depicted in the charts. For instance, for the default setup, it takes approximately 48 hours for *pSPQ* to complete. This is due to the fact that some Reducers are assigned with too many feature objects and *pSPQ* has to perform $O(|O_i| \cdot |F_i|)$ score computations before termination. Still, the algorithms employing early termination perform much better in all cases. Again, when *eSPQsco* is considered, its performance is the best among all algorithms, and it remains quite stable even in the case of more demanding queries. This experiment verifies the nice properties of *eSPQsco*, even for the combination of large-scale dataset with a demanding data distribution.

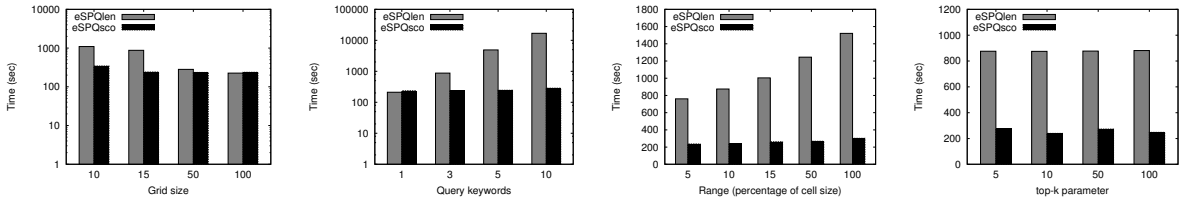
8. RELATED WORK

In this section, we provide a brief overview of related research efforts.

Spatial Preference Queries. The spatial preference query has been originally proposed in [16] and later extended in [17]. Essentially, this query enables the retrieval of interesting spatial locations based on the quality of other facilities located in their vicinity. Rocha et al. [12] propose efficient query processing algorithms based on a mapping in score-distance space that enables the materialization of sufficient pairs of data and feature objects. Ranking of data objects based on their spatial neighborhood without supporting keywords has also been studied in [7, 15]. As already mentioned, none of these approaches support keyword-based retrieval.

Spatio-textual Queries. Object retrieval based on a combination of spatial and textual information is a highly active research area recently. We refer to [3] for an interesting overview of query types along with an experimental comparison. The query studied in this paper has similarities to spatio-textual joins. Spatio-textual similarity join in a centralized setting is studied in [1], while a ranked version of this join which does not require thresholds from the user is studied in [10]. Partitioning strategies that also support multi-threaded processing of spatio-textual joins are examined in [11]. The spatial group keyword query [2] retrieves a group of objects located near the query location, such that the union of their textual descriptions covers the query keywords. In [5], the best keyword cover query is introduced, which retrieves a set of spatial objects that together cover the query keywords and additionally are located nearby. The most relevant query to our work is the ranked spatio-textual preference query proposed in [14]; in this paper, we study a distributed variant of this query. Nevertheless, all the above works target centralized environments, and their adaptation to distributed, large-scale settings is not straightforward.

Spatio-textual Queries at Scale. The current trend for scalable query processing is to employ a parallel processing solution based on MapReduce. For a survey on query processing in MapReduce we refer to [6]. To the best of our knowledge, the area of spatio-textual query processing at scale is not explored yet. Existing systems for parallel and scalable data processing in the context of MapReduce include SpatialHadoop [8]. However, SpatialHadoop targets



(a) Varying grid size. (b) Varying number of query keywords. (c) Varying query radius. (d) Varying k.

Figure 9: Experiments for Clustered (CL) dataset.

spatial data and is not optimized for spatial data annotated with textual descriptions. Spatial joins (binary and multi-way joins) in a MapReduce context are also studied in [18, 9] respectively, but again no provision for textual annotations exists. In [19], an approach for spatio-textual similarity join in MapReduce is presented, where pairs of spatio-textual objects located within a user-specified distance and having textual similarity over a user-specified threshold are retrieved. However, the query in [19] targets a single dataset (essentially a self-join) without ranking and without keywords as user input. Another substantial difference to our work is that their solution requires multiple MapReduce phases (jobs), whereas our algorithms consist of a single MapReduce job. Finally, a recent work on ranked query processing (top-k joins) in MapReduce is presented in [13], which employs early termination in a different way (in the Map phase) from this work (in the Reduce phase).

9. CONCLUSIONS

In this paper we study the problem of parallel/distributed processing of spatial preference queries using keywords. We propose scalable algorithms that rely on grid-based re-partitioning of input data in order to generate partitions that can be processed independently in parallel. To boost the performance of query processing, we employ early termination, thus reporting the correct result set after examining only a handful of the input data points. Our experimental study shows that our best algorithm consistently outperforms the remaining ones, and its performance is not significantly affected even in the case of demanding queries.

Acknowledgments

The work of C. Doulkeridis, A. Vlachou, and D. Mpestas has been co-financed by ESF and Greek national funds through the Operational Program "Education and Lifelong Learning" of the National Strategic Reference Framework (NSRF) - Research Funding Program: Aristeia II, Project: ROADRUNNER. The work of N. Mamoulis was supported by EU-funded Marie Curie Reintegration Grant project titled LBSKQ.

10. REFERENCES

[1] P. Bouros, S. Ge, and N. Mamoulis. Spatio-textual similarity joins. *PVLDB*, 6(1):1–12, 2012.

[2] X. Cao, G. Cong, C. S. Jensen, and B. C. Ooi. Collective spatial keyword querying. In *Proc. of SIGMOD*, pages 373–384, 2011.

[3] L. Chen, G. Cong, C. S. Jensen, and D. Wu. Spatial keyword query processing: An experimental evaluation. *PVLDB*, 6(3):217–228, 2013.

[4] J. Dean and S. Ghemawat. MapReduce: simplified data processing on large clusters. In *Proc. of OSDI*, 2004.

[5] K. Deng, X. Li, J. Lu, and X. Zhou. Best keyword cover search. *IEEE Trans. Knowl. Data Eng.*, 27(1):61–73, 2015.

[6] C. Doulkeridis and K. Nørvåg. A survey of analytical query processing in MapReduce. *VLDB Journal*, 2014.

[7] Y. Du, D. Zhang, and T. Xia. The optimal-location query. In *Proc. of SSTD*, pages 163–180, 2005.

[8] A. Eldawy and M. F. Mokbel. SpatialHadoop: A MapReduce framework for spatial data. In *Proc. of ICDE*, pages 1352–1363, 2015.

[9] H. Gupta, B. Chawda, S. Negi, T. A. Faruquie, L. V. Subramaniam, and M. K. Mohania. Processing multi-way spatial joins on MapReduce. In *Proc. of EDBT*, pages 113–124, 2013.

[10] H. Hu, G. Li, Z. Bao, J. Feng, Y. Wu, Z. Gong, and Y. Xu. Top-k spatio-textual similarity join. *IEEE Trans. Knowl. Data Eng.*, 28(2):551–565, 2016.

[11] J. Rao, J. J. Lin, and H. Samet. Partitioning strategies for spatio-textual similarity join. In *Proc. of BigSpatial Workshop*, pages 40–49, 2014.

[12] J. B. Rocha-Junior, A. Vlachou, C. Doulkeridis, and K. Nørvåg. Efficient processing of top-k spatial preference queries. *PVLDB*, 4(2):93–104, 2010.

[13] M. Saouk, C. Doulkeridis, A. Vlachou, and K. Nørvåg. Efficient processing of top-k joins in MapReduce. In *Proc. of IEEE Big Data*, 2016.

[14] G. Tsatsanifos and A. Vlachou. On processing top-k spatio-textual preference queries. In *Proc. of EDBT*, pages 433–444, 2015.

[15] T. Xia, D. Zhang, E. Kanoulas, and Y. Du. On computing top-t most influential spatial sites. In *Proc. of VLDB*, pages 946–957, 2005.

[16] M. L. Yiu, X. Dai, N. Mamoulis, and M. Vaitis. Top-k spatial preference queries. In *Proc. of ICDE*, pages 1076–1085, 2007.

[17] M. L. Yiu, H. Lu, N. Mamoulis, and M. Vaitis. Ranking spatial data by quality preferences. *IEEE Trans. Knowl. Data Eng.*, 23(3):433–446, 2011.

[18] S. Zhang, J. Han, Z. Liu, K. Wang, and Z. Xu. SJMR: parallelizing spatial join with MapReduce on clusters. In *Proc. of CLUSTER*, pages 1–8, 2009.

[19] Y. Zhang, Y. Ma, and X. Meng. Efficient spatio-textual similarity join using MapReduce. In *Proc. of IEEE Web Intelligence*, pages 52–59, 2014.