Discovering GIS Sources on the Web using Summaries

Ramaswamy Hariharan, Bijit Hore, and Sharad Mehrotra
Donald Bren School of Information and Computer Sciences
University of California, Irvine
Irvine, CA 92697
{rharihar,bhore,sharad}@ics.uci.edu

ABSTRACT

In this paper, we consider the problem of discovering GIS data sources on the web. Source discovery queries for GIS data are specified using keywords and a region of interest. A source is considered relevant if it contains data that matches the keywords in the specified region. Existing techniques simply rely on textual metadata accompanying such datasets to compute relevance to user-queries. Such approaches result in poor search results, often missing the most relevant sources on the web. We address this problem by developing more meaningful summaries of GIS datasets that preserve the spatial distribution of keywords. We conduct experiments showing the effectiveness of proposed summarization techniques by significantly improving the quality of query results over baseline approaches, while guaranteeing scalability and high performance.

Categories and Subject Descriptors

H.3 [**Information Storage and Retrieval**]: General

General Terms

Algorithms, Performance

Keywords

GIS, histograms, search engine, index structures, query processing

1. INTRODUCTION

The past decade has witnessed significant growth in the number of web sites that contain geographical information, specifically the number of GIS datasets available over the internet has grown exponentially. Such a growth in freely available GIS datasets is a direct result of systematic initiatives within government agencies (e.g., geodata.gov, fgdc.gov, etc.) and the private sector in response to the growing use for GIS data [7, 8]. Organizations use the data made available for a variety of coordination and analysis tasks in a range of applications such as, emergency response, urban planning, and transportation planning.

Proliferation of millions of high quality GIS data sources has led to an important challenge of *source discovery* – that of, "discovering most relevant GIS data sources given a particular task". Today, such data sources are discovered over the internet in one of the following two ways:
(i) *Using Search Engines (e.g., Google, Yahoo, MSN, etc.):* Most GIS data sources are accompanied with metadata that describe the data source. Such metadata may include information on format, data types, schema, as well as some information about the data content and the spatial footprint of the data sources. Search engines index such metadata and analysts can discover relevant sources using keyword queries.
(ii) *Specialized Geographical Data Portals (e.g., Geodata.gov):* Similar to the first strategy, such portals also create an index for the data sources based on metadata provided. However, in addition to keyword queries, they also support search based on geography specified as boundary in the metadata.

Both of the above approaches to discovering relevant geographical data sources exhibit significant limitations. To appreciate the shortcomings let us consider a particular source discovery problem: Imagine that an analyst doing shelter planning for a city wants to:

"Find databases of **public buildings** containing information about **emergency shelters** in **Orange County**".

It is difficult to convert such a source discovery task into a simple keyword query permitted by current search engines. Since keywords do not capture spatial semantics, if data is marked as that belonging to Irvine, California (instead of Orange County) – a purely keyword based match will consider the source irrelevant even though it is not. Current portals overcome this problem by interpreting geography explicitly using interactive maps. While this alleviates the problem to a certain extent, there remains significant opportunity to improve the quality of search. The prime limitation of portals is that they contain only textual metadata summaries created by humans, that describe the data at a very coarse level. Metadata lacks textual clues and do not capture the geospatial characteristics [19]. This affects the retrieval quality of sources significantly. Hence, instead of relying on the metadata, retrieval systems should focus on GIS contents that are rich in both textual and spatial descriptions of the data.

Source discovery based on contents has been previously studied for textual data [12, 16]. For each source, a summary in the form of keyword frequencies is created by counting the number of documents in which the keyword appears. Data sources are deemed most relevant for a given query based on their keyword frequencies. Such an approach needs significant generalization in case of GIS data. The goal is not frequency but rather frequency in the area of interest. To see this, consider an user-query asking for "High schools in Irvine". A dataset that contains general information about Irvine and also about schools will be deemed relevant to the query, even if it has no information on schools within Irvine. A simple keyword based approach is not able to capture this spatial distribution of objects in significant detail.

In this paper, we consider such a solution that uses summaries to retrieve sources based on frequency of resources (keywords) in a region. Traditionally, summarization is done using histograms, and many techniques have been developed for approximating joint distribution of data. The histograms are used to estimate the frequency count of keywords in the region specified in a query. Accurate estimation of frequency counts is the key component of most relevance-ranking metrics used in web-based searches. However, creating histograms for keywords distributed in space throws up a new challenge – that of summarizing the spatial distribution of multiple keywords simultaneously. In this paper, we develop a set of new algorithms to address this problem and provide ample empirical proof of the practical advantages of our proposed solutions. Our main contributions are the following:

1. We develop a suite of novel histogram-based algorithms for summarizing GIS datasets. These summary structures preserve both spatial and textual information present in the original datasets while achieving significant compression.

2. We propose techniques to implement source discovery queries using the summaries.

3. Finally, we evaluate the effectiveness of our summarization algorithms through experiments which show significant performance in retrieving GIS datasets while maintaining the high quality of results.

The remainder of the paper is organized as follows. In Section 2, we introduce our GIS data, query model, and relevance score model. In Section 3, we provide technical background on histogram construction techniques and formally define the problem. In Section 4, we describe in detail our summarization techniques for GIS datasets. We report experimental evaluation of our techniques in Section 5. We discuss related work in Section 6 and conclude the paper with suggestions for future work in Section 7.

2. GIS DATA AND QUERY MODEL

In this Section, we describe the GIS data model, and formally define a spatio-keyword query and the relevance-scoring model used for ranked retrieval of the search results over GIS data sources.

GIS Data: A GIS dataset consists of information about spatial objects from some geographic region of interest, where a region may correspond to a country, state, county, city, or other user-defined places. The information contained about objects in a GIS dataset can be classified primarily into two classes: spatial footprint and textual description. The spatial footprint of an object is often represented by its "minimum bounding rectangle" (MBR) and the textual description is a set of keywords, chosen from a standard GIS dictionary [30]. We ignore numerical attributes since our query model do not ask for such information. Formally, a GIS dataset and a spatial object can be defined as follows:

Definition 2.1. **(GIS Data Model)** *A GIS dataset G contains a set of spatial objects S. The attribute set of G is $\{R, T\}$, where R contains a set of rectangles $\{r_1, r_2, ..., r_n\}$ and T contains a set of subset of GIS keywords $\{k_1, k_2, ..., k_n\}$. The cardinality of G is n. Each spatial object in S is represented by a triplet $\langle id, r, k\rangle$, where id is the object identifier, $r \in R$ is the MBR of the object's spatial footprint and $k \in T$ is a set of keyword descriptors associated with the object.*

We denote a rectangle r by two components, its lower-left and upper-right vertices, specified in Cartesian coordinates as $[(x_1, y_1), (x_2, y_2)|x_1 \leq x_2, y_1 \leq y_2]$, $x_1, y_1, x_2, y_2 \in \Re$. All other representations (e.g., lat-long format) can be transformed to Cartesian coordinates using suitable geographic projections. If G contains line or polygon objects, their spatial footprints are represented by their MBRs. However, for point objects, the corresponding MBR is a degenerate rectangle whose two corners are represented by the same point, that is, $x_1 = x_2$ and $y_1 = y_2$. While the domain of T is the set of all GIS terms, we will represent its active domain by a finite set $\{t_1, t_2, t_3, \ldots, t_d\}$, where d is the number of distinct terms that appear in G.

Query Model: Given a collection $C = \{G_1, G_2, ..., G_m\}$ of m GIS datasets, a *spatio-keyword* (SK) query consists of a set of keywords and a geographic region. For example, the query "{rivers, streams} in orange county" is a spatio-keyword query. The relevant datasets in C are those which contain spatial objects in the geographic region orange county and have the associated keywords rivers and streams. If the query region is specified as a set of keywords, we assume there exists a function that converts the set of keywords into a rectangle using a geographic thesaurus [10]. Now, We formally define a *SK Query* as follows:

Definition 2.2. **(SK-Query)** *A SK query Q is a pair (q_r, q_t), where q_r denotes a rectangle, and q_t is a set of keywords.*

Relevance Score: The goal of the search engine is to determine the "relevant" datasets and present them to the user in a ranked order. A dataset G_i is relevant to a *SK* query if it contains one or more spatial objects whose MBR intersects with q_r and contains the associated set of keywords in q_t. Given a *SK* query Q, the ranking function F assigns a score $F(G_i, Q)$ to each dataset G_i and returns the top-k datasets for some user-defined positive integer k. There are many relevance-scoring mechanisms proposed in literature, such as vector space model, boolean model, and extended boolean model [27]. A multi-keyword query might be evaluated as an "OR" or an "AND" query. Different measures choose to incorporate the two semantics in their relevance-scoring equation differently. We adapt the *Extended Boolean Model* (which is a popular relevance-scoring technique used in document retrieval systems [25]) for our application. This model captures the two semantics succinctly as shown in the following two formulae. Given, a positive real p, the p-norm relevance score for a dataset G_i is computed as follows:

$$F(G_i, Q)_{OR} = \sqrt[1/p]{\frac{\sum_{k=1}^{|q_t|} w^p_{(k,q_r,i)}}{|q_t|}} \tag{1}$$

$$F(G_i, Q)_{AND} = 1 - \sqrt[1/p]{\frac{\sum_{k=1}^{|q_t|}(1 - w_{(k,q_r,i)})^p}{|q_t|}} \tag{2}$$

where $|q_t|$ is the number of terms in Q and $w_{(k,q_r,i)}$ is the weight of term t_k with respect to query region q_r and dataset G_i. It is computed as follows:

$$w_{(k,q_r,i)} = \frac{tf_{(k,q_r,i)}}{tf^{max}_{(k,q_r)}} \tag{3}$$

where $tf_{(k,q_r,i)}$ is the frequency count of term t_k appearing in dataset G_i and associated with objects within spatial region q_r. $tf^{max}_{(k,q_r)}$ is the max_i $\{tf_{(k,q_r,i)}\}$. In short, the two formulae score sources based on their relative importance with respect to each query keyword in comparison to the most authoritative source for that keyword. Here "most authoritative" refers to the source with the highest occurrence of the keyword in the specified region. The above mentioned measure, and numerous others are all some form of frequency-based measures. Hence computing $tf_{(k,q_r,i)}$ is fundamental for any relevance based model for the GIS source discovery problem. However, accurately computing $tf_{(k,q_r,i)}$ for each data source is computationally expensive. Therefore, we resort to summarization of data that allow quick estimation of keyword frequencies in geographic regions. We refer to this as selectivity estimation problem.

Next, we describe the histogram-based approach for summarizing GIS data distributions and define the selectivity estimation problem.

3. SELECTIVITY ESTIMATION

Data with multiple attributes is often modeled as a distribution in a multidimensional space, where each dimension corresponds to an attribute. Histograms are a popular technique for approximating joint distributions [15] and are used in large data-processing systems. Algorithms for creating a histogram partition the data space into a number of regions called buckets such that all the data points are covered by at least one bucket. The summary consists of the extents of each bucket along with the total number of data objects enclosed within the bucket. The density of points is assumed to be uniform within each bucket. Therefore, given a query region, its selectivity can be easily estimated by computing the fractional overlap of this region with each bucket and aggregating the count over all overlapping buckets. Typically, histograms result in substantially compressed representation of the distribution since the number of buckets used is much smaller than the size of the dataset. Histograms maybe single-dimensional, where the distribution of data along different dimensions is approximated separately and "attribute independence" assumption is used to estimate the joint density in any region of this space. Alternatively, histograms may take a multidimensional view of the dataset, and partition the space into a given number of multidimensional buckets. In either case, the objective is to ensure high estimation accuracy, while achieving substantial compression in representation.

3.1 Histograms for GIS Datasets

Before we formally state the histogram creation problem for GIS datasets, we need to describe the pre-processing steps and features of the histogram construction process.

Pre-processing - Discretization of Spatial Domain: Since the spatial domain is real-valued and continuous, it is a common practice to discretize it into regular grid cells and associate a frequency (i.e., number of objects) with each cell [1]. The issue of associating correct frequencies arises for spatial objects such as lines and polygons that may span more than one cell. While there might be other ways to resolve the count issue, for simplicity, we will unambiguously assign (only for counting purposes) each object to the cell corresponding to its centroid[1]. Hereafter, the histograms are constructed using these cells as the basic units (i.e., each cell is completely covered (contained) by a bucket or not covered at all). For simplicity, we will assume a n x n regular grid. Let $\{c_{i,j} | 1 \le i, j \le n\}$ be the set of cells formed after discretization, where $c_{i,j}$ refers to an individual cell and the indices i, j refer to the row and column of $c_{i,j}$. We denote by $f_{t_k,i,j}$, the number of objects in cell $c_{i,j}$ that have the keyword t_k associated with them. Therefore, each $c_{i,j}$ is associated with a set of *keyword-frequency* pairs, $F_{i,j} = \{(t_1, f_{t_1,i,j}), (t_2, f_{t_2,i,j}), (t_3, f_{t_3,i,j}), ..\}$. After the pre-processing stage, the spatial-keyword distribution of a dataset G is represented by $T(G) = \{(c_{i,j}, F_{i,j}) | \, 1 \le i, j \le n\}$. Though $T(G)$ itself can be considered an approximation of the distribution in G, most of the time it may turn out to be as large as G itself, even for small values of n (since the total number of cells is n^2). As a result, compression is still required for compact representation and efficient selectivity estimation.

Bucket: In the discretized domain, a bucket is a rectangle specified by its lower-left and upper-right corner cells. The associated list of $(keyword, frequency)$ pairs is formed by merging the corresponding lists over all the cells comprising the bucket.

Error Metrics: In order to estimate the accuracy of histogram, we use the standard *Sum Squared Error* (SSE) metric, which is defined as follows:

Given any bucket b_l with corner cells $(c_{l_x,l_y}, c_{u_x,u_y})$ and keyword t_k,

$$SSE(t_k, b_l) = \sum_{i=l_x}^{u_x} \sum_{j=l_y}^{u_y} (f_{t_k,i,j} - avg(t_k, i, j))^2 \tag{4}$$

Given any bucket b_l with corner cells $(c_{l_x,l_y}, c_{u_x,u_y})$ and a set of keywords $T_s \subseteq T$,

$$SSE(T_s, b_l) = \sum_{k=1}^{|T_s|} \sum_{i=l_x}^{u_x} \sum_{j=l_y}^{u_y} (f_{t_k,i,j} - avg(t_k, i, j))^2 \tag{5}$$

where

[1]if the centroid lies on a boundary point, it is assigned to one of the neighboring cells randomly.

$$avg(t_k, i, j) = \frac{\sum_{i=l_x}^{u_x} \sum_{j=l_y}^{u_y} f_{t_k,i,j}}{(u_x - l_x + 1) * (u_y - l_y + 1)}$$

Space Constraint: For our algorithms, we simply specify the space budget (maximum storage) in KBs that the histogram may use and leave the number of buckets unspecified. In general, histogram creation algorithms specify the maximum number of buckets to be used. This works fine when it takes the same amount of memory to specify each bucket but, in the case of multi-keyword distributions, buckets might be heterogeneous, that is, contain multiple keywords. As a result, different buckets may require substantially different amounts of space for representation.

Now, the summarization problem can be stated as follows:

Definition 3.1. ***(Optimal Histograms)*** *Given a dataset G, its discretized representation $T(G)$, and space budget M, compute a histogram $H(G)$, which minimizes the value of $\sum_{k=1}^{d} \sum_{l=1}^{|B|} error(t_k, b_l)$, where d denotes the number of keywords and $error(t_k, b_l)$ denotes the error measure of spatial distribution of keyword t_k in bucket b_l (as specified in equation 4) and $|B|$ is the total number of buckets in $H(G)$ with the constraint that total space required for representing the buckets is $\leq M$.*

We are ready to describe our histogram construction techniques (that attempt to solve the above mentioned optimization problem) for GIS datasets in the next Section.

4. SUMMARIZING SPATIAL-KEYWORD DISTRIBUTIONS

The problem for GIS datasets is significantly more challenging than plain, vanilla spatial datasets due to the presence of multiple keywords associated with each object. The optimal histogram construction problem (as posed in the problem 3.1) is NP-hard even for datasets with a single keyword [23]. As a result, a number of heuristics have been proposed in the literature that work well in practice. In this Section, we present a novel set of algorithms which use one such heuristic method (*MinSkew* [1]) as a building block. First we describe the *MinSkew* algorithm and the standard storage model for histograms.

MinSkew: The *MinSkew* histogram construction technique is shown in Algorithm 1. The input consists of the dataset G (actually $T(G)$) and the desired number of final buckets. It takes a greedy *Binary Space Partitioning* approach to determine the best partitioning of the data space. The algorithm starts with a single bucket which is the MBR of the whole dataset. In each subsequent iteration, it systematically explores all possible vertical and horizontal splits (along the rows and columns of cells) within each bucket, and determines the most beneficial split value (line 5), that is, the one that results in maximal error-reduction (SSE_{red}). After iterating over all the buckets, the algorithm picks the bucket with maximal SSE_{red} (line 8) and splits it into two, thus incrementing the number of buckets by 1. As a result, the algorithm terminates in $|B| - 1$ iterations. The total number of candidate splits to consider in each iteration is $O(n)$ ($n-1$ horizontal and $n-1$ vertical). Also, in each iteration, the algorithm has to look at $O(|B|)$ buckets at most, therefore, the worst case complexity of the algorithm

Algorithm 1 MinSkew(Data G, NumBuckets $|B|$)

```
1: Init: $b \leftarrow MBR(G)$, $Buckets = \{b\}$, $CurNumBuckets = 1$
2: while $CurNumBuckets < |B|$ do
3:   for all $b_i \in Buckets$ do
4:     Compute $SSE(b_i)$
5:     $\{b_{i_1}, b_{i_2}\} \leftarrow$ Split $(b_i)$
6:     Compute $SSE_{red} = SSE(b_i) - SSE(b_{i_1}) - SSE(b_{i_2})$
7:   end for
8:   Pick bucket $b_i^*$ with maximal $SSE_{red}$
9:   $Buckets = Buckets \cup \{b_{i_1}^*, b_{i_2}^*\} \setminus \{b_i^*\}$
10:  $CurNumBuckets++$
11: end while
12: return $Buckets$
```

is $O(n|B|^2)$. In practice the *MinSkew* algorithm runs very fast and the time taken is almost linear in n and $|B|$.

Summary Representation: Since the dataset consists of both spatial and textual information, the choices for representing the histograms can be broadly classified into two groups: The first one stores a single representation of each bucket with a pointer to a list of *keyword-frequency* (*k-f*) pairs that appear within the bucket. In this representation, a keyword may appear in multiple bucket-lists. The second approach does the opposite, it stores a single representation for each keyword with a corresponding list of *bucket-frequency* (*b-f*) pairs, where b is a pointer to a bucket and f is the frequency of occurrence of the keyword in the bucket[2]. Given a space budget M, we represent the storage requirements for the above two schemes as follows:

$$M = c_1.|B|.T_{avg} \quad (6)$$

where T_{avg} is the average number of *k-f* pairs corresponding to buckets in B and c_1 is a constant.

$$M = c_2.|T|.B_{avg} \quad (7)$$

where B_{avg} is the average number of *b-f* pairs in the list corresponding to keywords in T and c_2 is a constant.

The performance of different estimation techniques varies depending on which of the above two representation schemes is used. We will discuss and illustrate these characteristics below and in the next Section. Now, we describe the three summarization techniques for GIS datasets.

4.1 Keyword Distribution as Separate Layers

In this technique, we view $T(G)$ as a multi-layered data distribution, where each layer corresponds to a distinct keyword. The *MinSkew* algorithm can be applied to each layer's distribution separately. The union of the resulting buckets computed for all the layers represents the final histogram. Here, to represent the histogram as *k-f* pairs, we assign $T_{avg} = 1$ in Equation 6 (since only single keyword appears in each bucket) which results in $M = c_1.|B|$. Without loss of generality, let us assume $c_1 = 1$, so that $M = |B|$. However, each layer needs to be allocated its share of buckets from the total budget. The goal is to make this allocation in a way that minimizes the overall error (across all layers). The problem can be formalized as follows:

Let the original data distribution $T(G)$ be decomposed into a set of layers, $\{L_k\}$, where L_k represents the spatial

[2]Pairs with only non-zero frequencies are stored in both cases.

Algorithm 2 LayeredMinSkew(Data G, NumBuckets $|B|$)

```
1: Init: $Layers = \{L_1, L_2, .., L_d\}$, $Buckets = \{\}$
2: // pre-computation of buckets to layers
3: for all $L_i \in Layers$ do
4:     for $j = 1$ to $|B| - d + 1$ do
5:         $Bkts[i, j] \leftarrow$ MinSkew$(b_i, j)$
6:         $SSE[i, j] = SSE(Bkts[i, j])$
7:     end for
8: end for
9: // DP to compute optimal buckets to each layer
10: for $k = 1$ to $d$ do
11:     for $b = k$ to $|B|$ do
12:         $SSE(k, b) = \min_{1 \le j \le b-k+1} \{SSE(k-1, b-j) + SSE[k, j]\}$
13:         $AllocBuckets(k, b) = argmin_j$
14:     end for
15: end for
16: // reconstruction of final buckets
17: $n = |B|$
18: for $k = d$ to $1$ do
19:     $Buckets = Buckets \cup Bkts[k, AllocBuckets(k, n)]$
20:     $n = n - AllocBuckets(k, n)$
21: end for
22: return $Buckets$
```

distribution of keyword t_k. Let $|B|$ denote the maximum total number of buckets allowed. Then, the problem of layered histogram construction is the following:

DEFINITION 4.1. ***(Layered Histogram)*** *Given a dataset with d distinct keywords and a budget of $|B|$ buckets, find a partition $[n_1, n_2, ..., n_d]$ of $|B|$ (i.e., $\sum_{i=1}^{d} n_i = |B|$), such that, when layer L_i is assigned n_i buckets, the error measure $\sum_{k=1}^{d} \sum_{l=1}^{n_k} SSE(t_k, b_l)$ is minimized, assuming the MinSkew algorithm is used to summarize each layer.*

Our bucket-allocation algorithm is based on the observation that "*more uniform a distribution, smaller the number of buckets required to achieve a given level of error*". For example, a layer that has a perfectly uniform distribution of points (i.e., same number of data points in each cell of the grid), can achieve minimum value of SSE using just a single bucket (one that encloses all the n^2 cells).

Naive Solution: A naive solution to partitioning $|B|$ buckets to d layers explores all possible allocation of buckets to each layer, such that $\sum_{i=1}^{d} n_i = |B|$ and pick the best partition that minimize the SSE. There are $\frac{(|B|-1)!}{(|B|-d)!(d-1)!} \approx |B|^d$ such possibilities of allocating $|B|$ buckets to d layers. Hence, the naive solution is clearly impractical.

Optimal Solution using Dynamic Programming: We propose a dynamic programming solution that allocates $|B|$ buckets to d layers ($|B| \ge d$) in an optimal manner as shown in Algorithm 2. The algorithm starts with a pre-computation step (lines 3-8) where each layer is allocated a number of buckets ranging from 1 to $|B|$. The variable $SSE[i, j]$ captures the SSE of allocating j buckets to layer i. The buckets formed are stored in $Bkts[i, j]$. The main idea of dynamic programming is captured in lines 10-15. We denote $SSE(k, b)$ (line 12) as total error incurred in allocating b buckets to layers 1 through k. Now, fixing an order on the set of keywords, the optimum solution to the problem can be written recursively in terms of optimal solutions to smaller sub-problems, where $SSE[k, j]$ denotes the error of the histogram created on the k^{th} layer's data distribution using a total of j buckets. The variable $AllocBuckets(k, b)$ captures the value of j in the previous step for which $SSE(k, b)$ was minimal. Finally, the reconstruction of optimal buckets allocated to each layer is done in lines 17-21 by looking up the number of buckets allocated to each layer and retrieving the actual buckets from the $Bkts$ array.

Complexity Analysis: In lines 3-8, the algorithm needs to pre-compute $SSE[k, j]$ for all $1 \le k \le d$ and all j where $1 \le j \le |B| - d + 1$. The range of j is decided by the fact that, there are at least 1 and at most $|B| - d + 1$ buckets allocated to each layer. Hence, the pre-computation cost is $O(n|B|^2 d)$. We need to compute $SSE(k, b)$ for all $1 \le k \le d$ and for all $1 \le b \le |B|$. This requires $O(d|B|)$ iterations. For each iteration, we need to look up the $SSE[i, j]$ array $O(|B|)$ times, which gives a total complexity of $O(d|B|^2)$. Finally, in lines 17-21, the reconstruction of buckets takes d steps. Hence, the overall complexity of our dynamic programming approach is $O(d|B|^2 + n|B|^2 d + d) \approx O(n|B|^2 d)$.

Discussion: This technique is akin to the dynamic program approach in [17]. They build separate histogram for multiple attributes. The goal there is to optimally allocate given number of buckets to each attribute so that overall error is minimized. Our technique is suited for *b-f* pairs based representation, since it groups together all buckets that belong to a particular keyword. In contrast, we cannot group more than one *k-f* pair in each bucket as the MBRs of buckets belonging to different keywords are not the same. As a result, this summarization scheme is not efficient for *k-f* based representation. We observe this effect in our experiments (see Figures 1(a) and 1(b)). Our next technique favors the representation of final summary as *k-f* pairs.

4.2 Multi-Keyword Space Partitioning

In this technique, all the keywords that appear in certain spatial region share the same bucket. The new algorithm called *MultiKeyMinSkew* is shown in Algorithm 3. It modifies the *MinSkew* algorithm in two important ways: First, we introduce new aggregate error-metric $SSE(b)$ which combines the SSE of all keywords present in the spatial footprint of bucket b (see Error Metrics in Section 3.1). Second, instead of measuring the SSE reduction per extra bucket, we measure the reduction in the combined SSE per unit additional space (line 6). This modification is necessitated by the fact that two heterogeneous buckets may require significantly different amounts of space depending on the number of keywords falling within their extents.

From Equation 6 or 7, we cannot compute the total space M, by specifying $|B|$, as we do not know how many keywords are going to be associated with a bucket. Hence, we specify the space budget M itself in the input to the algorithm. The algorithm starts by assigning all the data in G to a single bucket as usual. It uses the function $Space(b)$ to compute the space requirement for a bucket b. In lines 4-11, the algorithm determines for each bucket, its maximal $SSE_{red/space}$ (i.e., the reduction in SSE per unit increase in storage) by exploring all possible binary space partitions of the bucket. The best overall split is added and the space required for representing the new bucket is deducted from the budget M. The algorithm terminates when no new bucket can be accommodated within the remaining space budget.

Algorithm 3 MultiKeyMinSkew(Data G, Budget M)

```
1: Init: $b \leftarrow MBR(G)$, $Buckets = \{b\}$, $CurSpace = Space(b)$
2: while $CurSpace < M$ do
3:     for all $b_i \in Buckets$ do
4:         Compute $SSE(b_i)$
5:         $\{b_{i_1}, b_{i_2}\} \leftarrow$ Split $(b_i)$
6:         $SSE_{red/space} = \frac{SSE(b_i) - SSE(b_{i_1}) - SSE(b_{i_2})}{Space(b_{i_1}) + Space(b_{i_2}) - Space(b_i)}$
7:     end for
8:     Pick bucket $b_i$ with maximal $SSE_{red/space}$
9:     $Buckets = Buckets \cup \{b_{i_1}, b_{i_2}\}$
10:    $CurSpace = CurSpace - Space(b_i) + Space(b_{i_1}) + Space(b_{i_2})$
11: end while
12: return $Buckets$
```

Complexity Analysis: In each step of the algorithm, at most d keywords appear in a bucket. To compute the best split requires $O(n)$ steps for each keyword. Hence, the complexity of this step is $O(dn)$. Let I be the number of steps the algorithm executes. The choice of the next bucket to split requires a maximum of $O(I)$ steps. Finally, the overall complexity is $O(I^2dn)$. It is to be noted that we don't have prior knowledge of how many iterations the algorithm runs. However, the lower and upper bounds on I can be easily derived. In Equation 6, $|B|$ denotes the number of iterations I. Hence, the lower and upper bounds on $|B|$ occurs when $T_{avg} = d$ and $T_{avg} = 1$, respectively.

Discussion: This technique naturally generates summaries in the form of *k-f* pairs. This representation can be converted into a set of *b-f* pairs by grouping all the buckets together in which a keyword appears. However, a disadvantage is, if the dataset has lots of keywords distributed across the space, each keyword could potentially point to many *b-f* pairs, thereby losing its storage compactness. We notice this effect in the estimation quality of our experimental dataset. Our next technique aims to alleviate the ill-effects of the first two by producing summaries suitable for both the schemes.

4.3 Keyword Clustering

In the previous two algorithms, we explored two extreme cases - in the first one, we isolated each keyword layer from another, while, in the second approach, we considered all layers together, viewing them as a single spatial distribution. From the discussion that followed, there is reason to believe that a hybrid approach that combines similar layers together, while keeping layers with significantly different spatial distributions separate, can produce better histograms in the same amount of space, irrespective of the final representation. Therefore, our third algorithm proposes a clustering technique to determine "similarly distributed" keywords that can be summarized together. Once the clusters are determined, each group of keywords is allocated a share of overall space budget. Then, within each group, we create buckets using Algorithm 3 discussed in Section 4.2.

4.3.1 Clustering Keyword Distributions

We use one of the widely used clustering techniques, the *k-means* algorithm for grouping the spatial distribution of keywords into a pre-specified number of clusters [29]. *k-means* algorithm groups items together that are similar in nature by using an user-specified similarity (distance) measure. We propose the following measure based on the spatial distribution of keywords to form the initial keyword clusters using the *k-means* algorithm.

DEFINITION 4.2. **(Distributional Distance)***: Let $f_{t_k,i,j}$ be the frequency of keyword t_k in cell c_{ij} and the spatial frequency distribution of t_k be denoted $F_{t_k} = \{f_{t_k,i,j}, 1 \leq i,j \leq n\}$, then the distance between two keyword distributions F_{t_1} and F_{t_2} is given by the following formula:*

$$Dist(F_{t_1}, F_{t_2}) = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{n}(f_{t_1,i,j} - f_{t_2,i,j})^2} \qquad (8)$$

Our clustering algorithm **KeyClusters** takes as input the frequency distribution set of all keywords F_T, and the number, c, and outputs c-way partition of the keyword set. Classical *k-means* algorithm suffers from the problem of determining the right number of clusters. Recently, many techniques are proposed that address this problem. We utilize one such technique, the *X-means* [24] algorithm, that outputs the right number of clusters based on the optimization of the Bayesian Information Criterion (BIC). *X-means* takes as input a range of values for k. It then performs a locally optimal *k-means* clusterings within this range, evaluates them using the BIC criterion, and returns the value of k that evaluates the best. We discuss next, our clustering based spatial partitioning algorithm, *KeyClusterMinSkew*.

KeyClusterMinSkew: The input to the algorithm is a set of k clusters, $\{c_1, c_2, ..., c_k\}$ and space budget M. Each cluster, c_i is initialized to bucket b_i. Thereafter, the algorithm works the same as **MultiKeyMinSkew**, where we modify the initial variable *Buckets* to contain buckets formed from each c_i and variable *CurSpace* specifies the space occupied by the current set of buckets.

Discussion: There are important differences that we note between *KeyClusterMinSkew* and the first two algorithms. Unlike *LayeredMinSkew*, this algorithm does not end up with keywords pointing to single *k-f* pair, since a cluster contains multiple keywords that can share the same bucket. Hence it is suitable for representation as a set of *k-f* pairs. Unlike *MultiKeyMinSkew*, the number of buckets pointing to a keyword is minimized by the initial formation of clusters. Hence, the final summary can also be represented as a set of *b-f* pairs. We observe this effect in our experiments where the performance of this technique (actually, a slightly modified version of it) is good for both representation schemes.

4.4 Rank-evaluation using Summaries

The goal of rank-evaluation is to identify relevant sources in ranked order for a given user query. First summaries are created using the histogram techniques for each of the dataset present in our original collection C. Next, the summaries are indexed using a suitable index structure. When a user query arrives, we first traverse the index structure to retrieve the summarized datasets that satisfy the query. Next, the relevance of each of these datasets is estimated using the scoring functions in Equations 1 or 2 and presented to user in a ranked order.

Creating Summaries: Given the summarization techniques, our goal is to create summaries of datasets in C. There are various parameters such as storage constraints, quality constraints, that affect the creation of final summaries. If storage limit is specified, each dataset is allocated a storage share so that the total storage stays within the limit. The issue is how to decide storage share, given that each dataset differs in size, number of keywords, and its data distribution. One simple approach is to allocate storage proportional to a dataset's original size. This ensures that bigger dataset gets enough buckets. However, for similar sized datasets, with varying number of keywords, allocating same storage might result in different summarization quality. This points to another possible approach where quality becomes a criteria. Our techniques measure the quality of summarization using the SSE metric (see Section 3.1). Lower the SSE, better the quality of summarization. We can place a threshold on the quality of summarization required for each dataset by specifying a target SSE reduction. Each dataset is summarized until it reaches the threshold error rate. While this approach ensures quality, it loses control on the storage bound. Each dataset requires different storage to reach the same error threshold. An ideal requirement would be to place a limit on the storage and ensure maximum quality within the storage limit. However, this will be a difficult optimization problem particularly when size of C becomes large. While each of these issues are worth further investigation, for our experimental purposes we use a target SSE reduction for creating final summaries.

Indexing Summaries: Once summaries are created for all datasets in C, the next step is to index the summaries for query processing. Since the summaries contain both keywords and MBRs, we use an index structure called *Inverted File-R*-tree* that combines keyword and spatial indices together [31]. We briefly explain this structure as follows: First an inverted index file [14] on the keywords is built. Then, a R*-tree [26] is constructed to the set of objects's MBR pointed to by each keyword. When a query is issued, the query keywords are first filtered using the inverted index. Later, the R*-tree corresponding to each query keyword are used to filter objects that satisfy spatial part of the query. The final answer set is produced by intersecting the filtered objects from each R*-tree.

There are other index structures discussed in the literature [6, 31]. In our experiments, we go by the choice of *Inverted File-R*-tree* for simplicity of index construction and performance. We represent the summary for each dataset either as *k-f* pairs or *b-f* pairs and then index it as follows: We create R*-tree for each new keyword and insert the bucket's MBR corresponding to the keyword along with the *frequency* information and dataset *id*. Each keyword's R*-tree indexes a set of MBRs irrespective of the dataset *id*.

Query Processing: When a *SK* query arrives with a set of keywords and spatial region, we first filter the keywords using the index structure. Using each keyword's R*-tree, the spatial part of the query is executed that results in a set of buckets. These buckets are grouped by dataset *ids* and for each dataset, the frequency count in the query region is estimated. Using the estimation count, the relevance score and ranking for each dataset is computed and presented to the user.

5. EXPERIMENTS

In this Section, we report experiments validating the effectiveness of our summarization techniques. The experiments are divided into two parts. In the first part, we compare three histogram algorithms discussed in Section 4 using real GIS datasets. In the second part, we apply the summarization techniques to a collection of some 475 real GIS datasets and perform rank evaluation for synthetic query workloads.

5.1 Comparison of Histogram Algorithms

In order to compare the estimation quality of our histogram algorithms, we first downloaded real GIS datasets from a website called *www.mapdex.org.* The experimental dataset consisted of 116,000 data objects with 96 unique keywords. We pre-processed the dataset by extracting only spatial and keyword attributes. Then, we did a basic stopword removal and stemming on the keywords. We created four different summaries using the algorithms in the previous Section[3]. We tested the quality of the summaries using different query workloads which we describe next.

5.1.1 SK Queries

We generated two kinds of workload consisting of 10000 *SK Queries* each. The first workload, w_{data}, follows the data distribution and is generated in the following manner: (1) Randomly choose a keyword according to the keyword-frequency distribution and (2) generate a rectangle from the spatial distribution of the selected keyword. The keyword and rectangle together form a *SK* query. The second workload, w_{uni}, instead of following the data distribution, is generated from an uniform distribution as follows: (1) Randomly select a keyword from the set of all keywords and (2) randomly generate a rectangle whose extent lies with the spatial domain of the dataset.

These are reasonable workloads in the absence of real query traces which is difficult to obtain for our purposes. We also created a variant workload, $w_{top-k\%}$ based on *top-k%* keywords, that is, we consider only most frequent set of keywords whose frequencies add up to the top $(100 - k)$% of the size of the dataset. For this part of the experiment, we only used single-keyword queries. In the second set of experiments, we also test the quality of ranking using two-keyword queries.

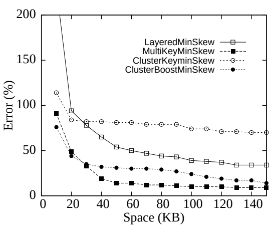
5.1.2 Results

For each of the histogram construction algorithms, we allocated space budget ranging from 10 to 200 KB. We evaluated the three different query workloads on each summary, as well as the actual dataset. Each summary was represented using both storage schemes (as discussed in Section 4). We measured the algorithm performance on a query workload by computing the *relative error* q_{err} for each query q and averaging them over all queries. q_{err} is computed as follows:

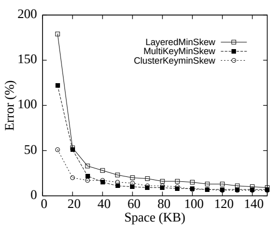
$$q_{err} = \frac{|n_a - n_e|}{max(n_a, 1)}$$

where n_a denotes the frequency of actual answer set and n_e is the frequency of estimated answer.

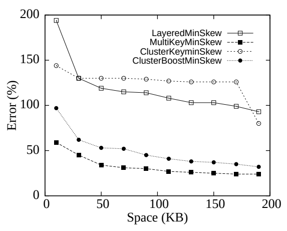
[3]Three of the histograms correspond to the three algorithms in Section 4 and the fourth one corresponds to a modified keyword clustering technique to be described in Section 5.1.2



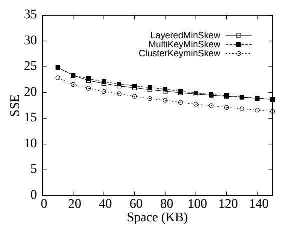
(a) w_{data}, kf



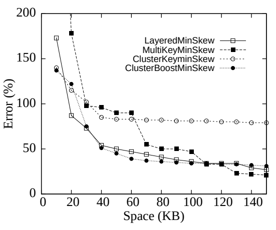
(b) $w_{top-10\%}$, kf



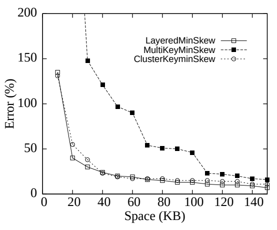
(c) w_{uni}, kf



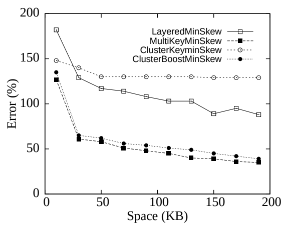
(d) $SSE : kf$



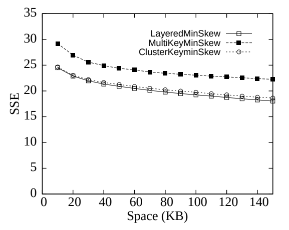
(e) w_{data}, bf



(f) $w_{top-10\%}$, bf



(g) w_{uni}, bf



(h) $SSE : bf$

Figure 1: Results.

Performance on Data Distribution Workload: We show the plots of *space-budget vs. average estimation-error* in Figures 1(a), 1(b), 1(e), and 1(f) for the three algorithms. Figure 1(a) shows the plot for representation scheme using *k-f* pairs. *MultiKeyMinSkew* starts at an error-rate of 90% and drops all the way down to less than 10% for 150 KB of storage. The error-rate for *LayeredHistogram* is high throughout and drops to only 35%. Counter-intuitively, the third algorithm *KeyClusterMinSkew*, which is expected to perform well since it achieves the lowest SSE for a given space budget (Figures 1(d) and 1(h)), performs poorly and shows hardly any improvement in estimation error with increasing storage. We can explain this phenomenon as follows: Recall that *ClusterKeyMinSkew* takes as input, k clusters. The hope is that the distance measure used would partition the keywords such that spatial distribution of keywords within a cluster is similar. Typically, the clustering step groups sparsely distributed keywords together into a single cluster. Subsequently, this algorithm, in an effort to reduce the overall SSE metric, allocates (almost) the whole space budget to the dense clusters since this leads to the maximum SSE reduction and therefore, ignores sparse clusters altogether. For instance, in this dataset the sparse cluster consisted of 79 out of 96 keywords. As a result, all the queries that consisted of keywords from the sparse group performed poorly. However, the queries from the dense group achieved low estimation error as shown in Figure 1(b). Here, we plot the estimation error for a *top-10%* variant workload.

For the second representation scheme, that uses *b-f* pairs (Figure 1(e)), we observe an exactly opposite phenomenon. Here, the *LayeredHistogram* performs better than the other two algorithms. However, the error rate almost drops the same to less than 25% for 150 KB of storage for both the algorithms. This drop is better for *LayeredHistogram* and worse for *MultiKeyMinSkew* compared to their other representation scheme. As for the performance of the *ClusterKeyMinSkew* algorithm, here too, the results obtained were poor. We attribute the cause to the same reason (i.e., allocation of all space to dense buckets). However, Figure 1(f) shows better performance on *top-10%* workload.

Boosting *ClusterKeyMinSkew*'s Performance: The poor performance of the *ClusterKeyMinSkew* algorithm for either representation schemes can be attributed to its bias towards dense clusters. This fact is validated by the performance shown on the plots of *top-10%* variant query workloads, on which the algorithm performs very well. Hence, we boosted the performance of the vanilla *ClusterKeyMinSkew* technique by forcing a certain fraction of the total budget to be used for the sparse cluster(s) exclusively. We made the fraction of budget that is pre-allocated to a sparse cluster to be proportional to the ratio of its initial SSE to the maximum SSE of any of the cluster. This improved the performance of the algorithm significantly for all our test workloads. We call this modified algorithm, the *ClusterBoostMinSkew*. It performs well for both the representation schemes as seen in Figures 1(a) and 1(e).

Performance on Uniformly Distributed Workload: We tested our algorithms on a uniform query workload. The results are shown in Figures 1(c) and 1(g). *MultiKeyMinSkew* performed consistently better for both the representation schemes, while *ClusterBoostMinSkew* significantly improved in performance compared to its previous version *ClusterKeyMinSkew*. *LayeredMinSkew* suffered in its performance with both the representation schemes. Recall, that the uniform workload largely consists of sparsely distributed keywords. The *MultiKeyMinSkew* again performs the best, since it transparent allocation policy is much more likely to distribute buckets more equitably amongst keywords, than the other algorithms.

Discussion: Our experiments with other similar sized real datasets more or less showed the same characteristics. Hence, we draw the following conclusions from the first set of experiments: The *b-f* pair based representation is suitable for summaries using *LayeredHistogram*, while the *k-f* pair based representation is suitable for summaries using *MultiKeyMinSkew*. Both the representations are more or less suitable for summaries using *ClusterBoostMinSkew*. For *Data* and *Uniform* distribution workloads, *MultiKeyMinSkew* performs well. *ClusterBoostMinSkew* showed significant improvement over its counterpart *ClusterKeyMinSkew.* If the workload consists of only *top-k%* keywords, *ClusterKeyMinSkew* performs slightly better than the other two.

5.2 Rank-Evaluation of Search Results

Here, we describe the results from our second set of experiments - for evaluating the ranking quality using our summary datasets. We downloaded 475 real GIS datasets from *www.mapdex.org.* For each dataset, we executed the preprocessing steps discussed in Section 5.1 and compressed using all the three histogram algorithms except the vanilla clustering-based algorithm, *ClusterKeyMinSkew.* We created 5 different sets of summaries by using a cutoff value for the *SSE Reduction* achieved. We used the following cutoff values for as stopping criteria: 55%, 65%, 75%, 85%, and 95%. The buckets that resulted from the summarized datasets were inserted into an *Inverted File-R*-Tree* index structure for quick retrieval. Our base-line comparison was against the original dataset, i.e, answers to queries were ranked by evaluating the query on the complete actual datasets.

We generated two uniform query workloads consisting of 10000 *SK* queries each. The first one consisted of "1-keyword" queries and the second one consisted of "2-keyword" queries. We used *OR* semantics in Equation 1 for computing the relevance of "2-keyword" queries using $p = 2$. Extension to queries with *AND* semantics is similar and therefore, we do not include the results here. The workload was created from a set of unique keywords collected from all the 475 datasets. We used only keywords that had a dataset (document) frequency of at least 10, to avoid generating zero hit queries.

We processed each query workload using the uncompressed datasets first. This resulted in a ranked list of sources for each query denoted by r_{orig}. We consider r_{orig} to be the true ranking and evaluate the performance of the three algorithms against this using a *rank_match* metric. We define *rank_match* as the fraction of data sources in another ranked list, r_{alg} (produced using a particular algorithm) that matches in rank with r_{orig}. For the purposes of this experiment we consider only top-5 sources for computing *rank_match.*

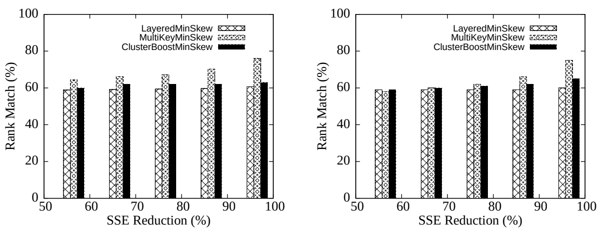
5.2.1 Results

In Figures 2(a) and 2(b), we show the plots of *SSE Reduction vs. rank_match* for all the three summarization algorithms. The best performance was again shown by the *MultiKeyMinSkew* algorithm, where the *rank_match* went up to 78% for "1-keyword" queries and up to 75% for "2-keyword" queries using summaries at the 95% *SSE Reduction* level. The comparative performances among the three algorithms almost agreed with the performances shown in the first part. These results leads us to conclude that *MultiKeyMinSkew* performs consistently better across all datasets for a uniform query workload. In comparison, the *rank_match* achieved by using simply metadata was only about 30% which was far worse than the level of matching achieved using any of the three techniques proposed in this paper.

In Table 1, we show the average execution time for each of our algorithm. Columns 1 and 2 corresponds to *1-keyword* and *2-keywords* queries, respectively. Barring minute differences among them, all the three achieved at least *7x* speed up compared to their uncompressed counterpart which is quite significant. In Column 3, we show the storage com-

Algorithm	Time (ms)	Time (ms)	Storage (MB)
LayeredMinSkew	2	2.9	10-25
MultiKeyMinSkew	2.3	2.6	5-11
ClusterBoostMinSkew	2.1	2.2	4-12
Uncompressed	17	32	251

Table 1: Average Execution Time and Storage.



(a) 1-Keyword (b) 2-Keyword

Figure 2: Rank Evaluation.

pression for all the three algorithms that followed *k-f* pair representation. All the three achieved quite a significant amount of more than 90% compression ratio. However, the compressions achieved by *MultiKeyMinSkew* and *ClusterBoostMinSkew* are slightly better.

Our experiments show tremendous improvement on the performance of *SK Queries* as a result of the compression, yet maintaining a high quality of search results. Given the encouraging results, we plan to test different keyword clustering and boosting mechanisms.

6. RELATED WORK

Data summarization using histograms is a well-researched problem in the database community. A variety of histogram based approaches have been proposed for both single dimensional and multi-dimensional data. A comprehensive survey of histogram creation techniques can be found in [15]. Adaptation of histograms to spatial datasets has been explored in [1, 2, 28]. Traditionally, all these techniques consider only the spatial attribute for approximation. However, GIS datasets contain both spatial and textual attributes.

Searching GIS information on the web has recently received a lot of attention. Researchers have addressed the issue of extracting geographic contents implicitly and explicitly present in web pages and turning them into a searchable GIS information catalog [3, 11, 18, 22]. The main focus here is to extract keywords denoting names, resource/utility descriptions and other geo-features present in web pages. In the context of query processing, the main focus is to build efficient indices over spatial and textual data [6, 13, 31]. In [6], the authors propose index structures separately for textual and spatial data and suggest many optimization techniques that speed up the retrieval of objects. The work in [13, 31] discusses hybrid index structures that consider text and space together.

Discovering textual information sources on the deep web is closely related in spirit to our work [5, 12, 16]. The main motivation of that work is to index billions of web pages hidden in the web behind query interfaces. The only way to index such hidden pages is to query the source through its interface and retrieve the contents. However, retrieving huge repository of textual information through queries is time consuming. Hence such work concentrate on minimal querying of sources to create summaries of textual information [4, 21]. Later, these summaries are used to route user queries to discover textual information sources.

There are many ranking models developed for Geographic Information Retrieval systems based on the metadata. These models compute spatial similarity measures based on the overlap between query region and spatial description (MBR) found in the metadata. They range from simple Boolean approach to recent probabilistic approaches [20].

7. CONCLUSIONS & FUTURE WORK

In this paper, we looked at the problem of GIS source discovery for spatial-keyword queries. We provided a solution to this problem by creating summaries of GIS datasets and using them for source discovery. We adapt existing well-known histogram based techniques and present three summarization algorithms for GIS data that take both textual and spatial information-content into consideration. To the best of our knowledge there exists no histogram technique that focus on summarizing a collection of multiple spatial distribution simultaneously. We conducted experiments evaluating the performance of the proposed algorithms. We characterized the effectiveness of each algorithm under different query workloads and representation schemes. We used a simple boosting technique to improve *ClusterKeyMinSkew*'s performance. With the boost, the algorithm showed promising results by consistently being close with the best performing algorithms in many situations. Our rank-evaluation experiments show that by using the summaries we can answer user queries with faster response time and high quality results.

There are many interesting possibilities for future research. Since, *ClusterKeyMinSkew* combines the positive effects of keyword clustering and spatial partitioning, we believe that by exploring further tuning techniques, it's performance can be improved significantly. We plan to conduct more experiments on real as well as synthetic datasets with varying distribution and characteristics. We mainly focused on GIS datasets that are downloadable, however, there are many useful sources that can only be queried and not downloaded in their entirety (e.g., GNIS [9]). It will be interesting to explore probing techniques similar to the ones proposed for textual sources that can create summaries of such databases. We tested our algorithms for rank evaluation with few hundred datasets, but we would like to explore challenges in scaling it to hundreds of thousands of datasets. In particular, response time for *top-k* queries can be improved by storing extra information in the index structures and use them to prune insignificant results ahead of time. Both these interesting problems are part of our ongoing and future work.

8. REFERENCES

[1] S. Acharya, V. Poosala, and S. Ramaswamy. Selectivity Estimation in Spatial Databases. In *Proc. of SIGMOD*, 1999.

[2] A. Aboulnaga, J. Naughton. Accurate Estimation of the Cost of Spatial Selections. In *Proc. of ICDE*, 2000, pp. 123-134.

[3] O. Buyukkoten, J. Cho, H. Garcia-Molina, L. Gravano, and N. Shivakumar. Exploiting Geographical Information of Web Pages. In *WebDB*, pp. 91-96, 1999.

[4] J. Callan. Query-Based Sampling of Text Databases. In *Information Systems*, 2001, vol 19(2), pp. 97-130.

[5] K. C.-C. Chang, B. He, and Z. Zhang. Toward Large Scale Integration: Building a MetaQuerier over Databases on the Web. In *Proc. of CIDR*, pp. 44-55, 2005.

[6] Y. Chen, T. Suel, and A. Markowetz. Efficeint Query Processing in Geographic Web Search Engines. In *Proc. of ACM SIGMOD*, pp. 277-288, June 2006.

[7] The Federal Geographic Data Committee. www.fgdc.gov.

[8] Geospatial One-Stop Initiative. www.geodata.gov.

[9] GNIS System. http://geonames.usgs.gov/pls/gnispublic/.

[10] Getty Thesaurus of Geographic Names Online.

[11] L. Gravano. Geosearch: A Geographically-Aware Search Engine, 2003. http://geosearch.cs.columbia.edu.

[12] L. Gravano, H. Garcia-Molina, and A. Tomasic. GlOSS: Text-Source Discovery over the Internet. In *ACM TODS*, vol. 24(2), pp. 229-264, 1999.

[13] R. Hariharan, B. Hore, C. Li, and S. Mehrotra. Processing Spatial-Keyword (SK) Queries in Geographic Information Retrieval (GIR) Systems. In *Proc. of SSDBM*, 2007.

[14] B. Harmon, E.Fox, R. Baeza-Yates, and W. Lee. Inverted Files. In *Information Retrieval: Data Structures and Algorithms*, Chapter 3, pp. 28-43, Prentice Hall, NJ, 1992.

[15] Y. Ioannidis. The History of Histograms (abridged). In *Proc. of VLDB*, 2003, 19-30.

[16] P. Ipeirotis, L. Gravano. Distributed Search over the Hidden Web: Hierarchical Database Sampling and Selection. In *Proc. of VLDB*, 2002, pp. 394-405.

[17] H. V. Jagadish, H. Jin, B. C. Ooi, and K.-L. Tan. Global Optimization of Histograms. In *Proc. of SIGMOD*, 2001.

[18] C. Jones, A. Abdelmoty, D. Finch, G. Fu, and S. Vaid. The Spirit Spatial Search Engine.: Architecture, Ontologies and Spatial Indexing. In *Proc. of GIScience*, pp. 125-139, 2004.

[19] R. Larson and P. Frontiera. Geographic Information Retrieval (GIR): Searching Where and What. In *Proc. of ACM-SIGIR*, 2004.

[20] R. Larson and P. Frontiera. Spatial Ranking Methods for Geographic Information Retrieval (GIR) in Digital Libraries. In *Proc. of ECDL*, 2004, pp. 45-56.

[21] Z. Liu, C. Luo, J. Cho, W.W. Chu. A Probabilistic Approach to Metasearching with Adaptive Probing. In *Proc. of ICDE*, 2004, pp. 547-559.

[22] A. Markowetz, Y. Chen, T. Suel, X. Long, and B. Seeger. Design and Implementation of a Geographic Search Engine. In *Proc. of WebDB*, 2005.

[23] S. Muthukrishnan, V. Poosala, T. Suel. On Rectangular Partitionings in Two Dimensions: Algorithms, Complexity, and Applications. In *Proc. of ICDT*, 1999.

[24] D. Pelleg, A. Moore. X-means: Extending K-means with Efficient Estimation of the Number of Clusters. In *Proc. of ICML*, 2000, pp. 727-734.

[25] G. Salton, E. Fox, H. Wu. Extended Boolean Information Retrieval. In *Communications of the ACM*, 1983, vol. 26(11), pp. 1022-1036.

[26] H. Samet. The Design and Analysis of Spatial Data Structures. Addison-Wesley Publishing, Reading, MA, 1990.

[27] A. Singhal. Modern Information Retrieval: A Brief Overview. In *IEEE Data Engineering Bulletin*, Special Issue on Text and Databases, Vol. 24, No. 4, 2001.

[28] C. Sun, D. Agrawal, and A. El Abbadi. Exploring Spatial Datasets with Histograms. In *Proc. of ICDE*, 2002, pp. 93-102.

[29] P.-N. Tan, M. Steinbach, V. Kumar. Introduction to Data Mining. Addison-Wesley, 2005.

[30] T. Wade, S. Sommer. A to Z GIS: An Illustrated Dictionary of GIS. ESRI Press, Redlands, CA, 2006.

[31] Y. Zhou, X. Xie, C. Wang, Y. Gong, and W. Ma. Hybrid Index Structures for Location-Based Web Search. In *Proc. of CIKM*, pp. 155-162, 2005.