Querying Spatial Patterns

Vishwakarma Singh
Dept. of Computer Science,
University of California, Santa
Barbara,
CA 93106, USA.
vsingh@cs.ucsb.edu

Arnab Bhattacharya
Dept. of Computer Science
and Engineering,
Indian Institute of Technology,
Kanpur,
Kanpur 208016, India.
arnabb@iitk.ac.in

Ambuj K. Singh
Dept. of Computer Science,
University of California, Santa
Barbara,
CA 93106, USA.
ambuj@cs.ucsb.edu

ABSTRACT

Spatial data are common in many scientific and commercial domains such as geographical information systems and gene/protein expression profiles. Querying for distribution patterns on such data can discover underlying spatial relationships and suggest avenues for further scientific exploration. Supporting such pattern retrieval requires not only the formulation of an appropriate scoring function for defining relevant connected subregions, but also the design of new access methods that can scale to large databases. In this paper, we propose a solution to this problem of querying significant subregions on spatial data provided as raster images. We design a scoring scheme to measure the similarity of subregions. All the raster images are tiled and each alignment of the query and a database image produces a tile score matrix. We show that the problem of finding the best connected subregion from this matrix is NP-hard and develop a dynamic programming heuristic. With this heuristic, we develop two index-based scalable search strategies, TARS and SPARS, to query patterns in large data repositories. Experimental results on real image datasets show that TARS offers an 87% improvement for small queries, and SPARS a 52% improvement in runtime for large queries, as compared to linear search. Qualitative tests on real datasets achieve precision of more than 80%.

1. MOTIVATION

Spatial data arise in various domains such as geographical information systems, biology, environmental management, and IC fabrication. Often, the distribution of a spatial attribute of interest (e.g., population density, contamination rate, vegetation growth, protein expression, etc.) is captured using a raster image [27, 34, 33]. Such an example is shown in Figure 1 which displays the population density map of Afghanistan[1]. The color of each pixel is associated with a particular value of the population density. In biological and medical images, pixel intensity represents the distribution of tissues, gene, or proteins. Figure 2 shows the fluorescent microscopy image of a cross section of a feline retina [9]. The intensity of a pixel reveals the distribution of peanut-agglutinin, a lectin found in the retina.

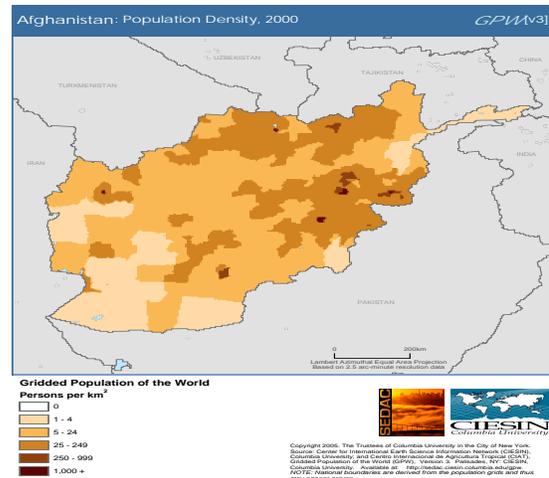
[1] http://sedac.ciesin.columbia.edu/gpw/

Figure 1: Population density map of Afghanistan (best viewed in color).
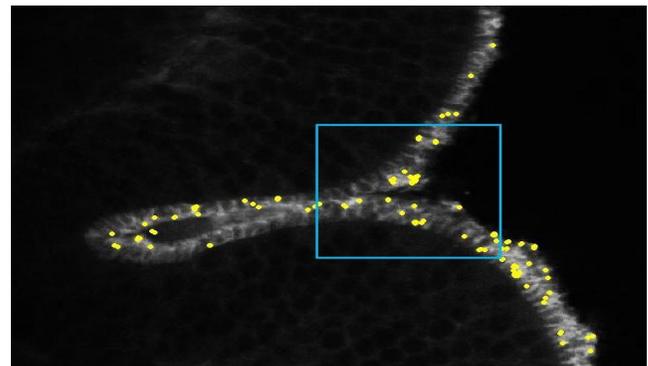

Figure 2: Example of a biologically interesting spatial pattern (best viewed in color). The marked pattern highlights a fold of the retinal tissue labeled with peanut-agglutinin conjugated to a fluorescent probe. Yellow dots are the point of interests detected using affine covariant region technique [25] of computing local descriptor.

Ever since John Snow's analysis of cholera outbreaks that resulted in finding a contaminated water pump in London in 1854 [38], the analysis of spatial data distributions has been a popular avenue of scientific inquiry. Revelation of similarity in demographic patterns helps us correlate and understand various geographic factors affecting population growth. Similarity in vegetation pattern dis-

covered by querying aerial images can help relate climate cycle and land formation at various places on Earth. In retinal images, the similarity in spatial patterns may offer new insights into biological processes.

In this paper, we propose to search for a specified pattern in a database of spatial distributions represented as raster images. A query pattern can be described either by specifying a local distribution or by marking a rectangular region of interest on a given image as shown in Figure 2. The database may consist of population density maps, biological images, remote sensing images, or raster images of any other domain. The task is to find sub-regions of images in the database that are similar to the specified query pattern and are meaningful.

Our problem statement is close to sub-image search for natural images [32]. The methods developed in this domain use local descriptors [26] that are computed around few key points of interest as shown by yellow circles in Figure 2. These descriptors are obtained from a small number of neighborhood pixels around detected points of interest and are designed to provide robustness for photometric, scale, viewpoint and affine changes for natural image matching. State-of-the-art SIFT descriptors [21] are histograms of the gradients of the sampled points around the key point. Sampled points are divided into 4×4 grids. Gradients of the sampled points in each grid are summarized using 8 bin orientation histograms. Histograms of all the grids are concatenated to yield a SIFT descriptor of size 128 for the key point. Local descriptors till now have had only limited success (around 66% precision [32]) because of the difficulty of coping with all the image variations. Raster images do not have challenges of photometric, viewpoint and affine changes. For spatial distributions in raster images, the resolution is known, and this permits easy normalization. Points of interest in natural images are computed using pixel intensity gradient and, therefore, may be absent in a large portion of an image as seen in Figure 2 making it impossible to carry out sub-image search for those regions. These points are also less than sufficient to summarize all the useful information in an image. Instead of just focusing on key points as for natural images, a solution to the proposed problem needs to capture the information over the entire useful part of an image (foreground), e.g., summarize each pixel representing population density in Figure 1 using a histogram.

Our method tiles the query image and database images into atomic units. Then, a domain-based scoring function is used to score an alignment of two atomic units. Finally, the score is aggregated over a connected region to find the best match. The idea of a match is illustrated in Figure 3. A query is aligned with each image in the database under all possible translations. Each alignment generates a matrix of scores, both positive and negative, between corresponding tiles. Positive scores denote foreground matches while a negative score means that a background tile of the query is matched to a database tile. A connected subregion over the matrix identifies a meaningful matching subregion. Scores over all possible connected subregions can be used to define answers to range and nearest-neighbor queries. The generality of the solution and the identification of best connected subregions are the unique aspects of our design.

Once we adopt the score and subregion based idea for retrieval of high-quality answers, the next challenge is one of scalability. How to identify the best subregions over millions of alignments? Clearly, a region-by-region search design will not work. So, how to develop access methods and index structures that can find the best subregions without examining all of them? Our solution to the scalability problem is two-fold: (i) development of an index structure that works with our definition of score, and (ii) design
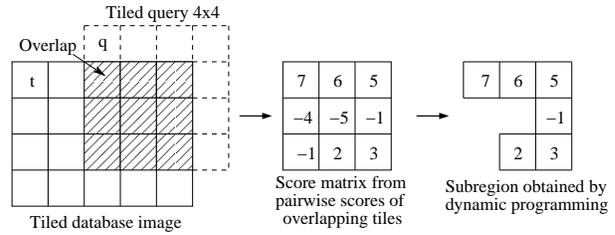

Figure 3: A 4×4 query is overlapped with a database map. For each tile in the 3×3 overlapped region, a score for the match is computed. Dynamic programming is run on the score matrix to obtain the maximal scoring connected subregion.

of two new algorithms that use the index structure to find the best subregions in an efficient manner.

The idea of finding the best connected subregion in a matrix that maximizes the sum of piecemeal scores is itself of theoretical and practical interest. We show that this problem is NP-hard. This necessitates appropriate heuristics that examine not all but a subset of connected subregions. We develop a dynamic programming based solution and characterize the class of subregions that is examined by this heuristic. Our access methods are unaffected by how the best connected subregions in an alignment are identified; they work correctly with any such heuristic.

In a nutshell, our contributions in this paper are as follows:

- We develop a score-based framework for identifying the best connected sub-regions for a given query region with tile-based decomposition (Section 3).

- We develop index structure based access methods to query the best matching subregions efficiently. The first method, TARS, is instance optimal but traverses the index multiple times and, therefore, performs better for small queries. The second method, SPARS, makes a single pass through the index and is suited for large queries (Section 4).

- We study the computational complexity of finding the highest scoring subregion. We show that this problem is NP-hard by reduction from the Thumbnail Rectilinear Steiner Tree problem [10]. We develop an efficient dynamic programming heuristic and characterize the class of subregions explored by this heuristic (Section 3).

- We empirically show scalability (Section 5) and quality (Section 5.4) of our methods on two real datasets.

2. RELATED WORK

Query by example for images, called Content-Based Image Retrieval (CBIR), has been extensively studied. Region-Based Image Retrieval (RBIR) systems extend CBIR by making the search sensitive to different regions of an image. A survey on the recent methods of CBIR and RBIR can be found in [5]. Most of the RBIR systems use automatic or manual region segmentation in order to characterize regions and then compute a one-to-one or many-to-one mapping to match query regions to those in the database [1, 4, 28, 39, 40]. Weakness of the segmentation based methods lies is their incapability to handle region queries that partially extend across various segments or regions of an image. Malki et al. [22] avoided segmentation by using a multi-resolution quadtree [8] to organize images. Their method had no constraint of connected pattern and had equal weight for foreground and background. Sub-image retrieval using local descriptors [26] has been addressed recently for

natural images [18, 32] but cannot be extended for querying patterns in raster images as discussed in Section 1.

Tiling is the most common way of storing raster data [37] in spatial DBMS. Image tiling at varying scales was used by Svetlana et al. [19] for recognizing natural scene categories using full image matching. Tiles were also used by [3] to partition images into clusters in the color space.

Methods for querying similar images based on full image matching has also been developed for aerial [23, 35] and biomedical [31, 6] images separately. Baumann et al. [2] proposed a web-enabled service over a multidimensional DBMS, used as storage, for interactive navigation and SQL based querying on raster images. Their system does not support pattern querying. Vinhas et al. [37] also proposed DBMS system for handling raster image in spatial databases. OLAP techniques were exploited by [13] to speed up aggregate query processing in raster image databases. New geo-raster operations with array algebra is proposed in [14]. Zhang et al. [42] developed index structure for spatio-temporal aggregation over streaming raster images for a given query region. They first split the images into tiles and then computed aggregate for each tiles. Gertz et al. [12] proposed a data and query model by extending Image Algebra to formulate and answer queries over geospatial image data. Pajarola et al. [29] provides a compression technique for large raster images and designs methods to support spatial range queries directly over compressed images. Hadjieleftheriou et al. [16] address the problem of querying a user defined movement patterns in space and time from a large collection of spatiotemporal trajectories. Yankov et al. [41] develop best-match searching algorithm for two-dimensional shapes.

Our work is the first to support pattern querying on geographic maps like demography, pollution, etc. The technique is generic enough to be extended to raster images of other domains like biology, medicine, etc. It differs from image retrieval techniques by supporting pattern querying without image segmentation into regions or objects, developing score based similarity measure, finding the best connected match, and discriminating between foreground and background to discover meaningful patterns.

3. SUB-REGION SIMILARITY

In this section, we discuss feature extraction from images, define similarity measure between a pair of image tiles, and then extend the idea of similarity between tiles to regions. Then, we show that the computation of the optimal score between two sub-regions is NP-hard. Finally, we develop a dynamic programming heuristic to compute a good alignment.

Each raster image in the database is split into tiles [37]. All the pixel values in a tile is summarized as a histogram. Dimension of the histogram equals the number of discrete levels of pixel values which is later reduced by a dimensionality reduction technique (PCA [30]) for efficiency. Finally, our database DB consists of the feature vectors of these tiles. We also tile the query image Q and obtain feature vectors similar to a database image tile. We search for similar regions for a given query image in a feature vector space. We use L_1 norm as the measure of distance between a pair of histograms. Raster images can be of varying resolution or scale. In this paper, we assume that a given database consists of raster images of the same scale. If the images are of varying resolution, they can be preprocessed and normalized to the same resolution as scale is known.

3.1 Scoring Function

We measure similarity between a pair of tiles using a scoring function. Our scoring function is a monotonically decreasing func-
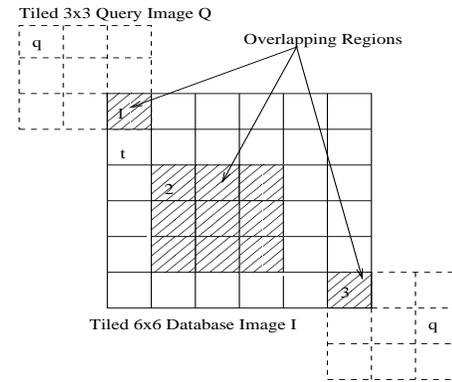

Figure 4: Overlapping regions found by translation of a query image Q on a database image I at 3 alignments.

tion of the distance between the feature vectors of a query tile q and an image tile t. Scoring function is defined as

$$score(q,t) = f(q) - g(d(q,t)) - c \qquad (1)$$

where f is a function based on domain knowledge, g is a monotonically increasing function, d is the distance between tiles and c is a constant. The score can be positive or negative. The intent of the scoring function is to discriminate between foreground (region of interest) and background. A query tile with little or no information forms the background and, therefore, should get a negative or low score no matter how good the match is. A tile with more pattern information is a part of *region of interest* (ROI) and should get a high score when matched with a similar database image tile. The function $f(q)$ measures whether the tile q is in a ROI. Singh et al. [36] design an instance of such scoring function called *Discriminator* scoring function. It computes the score between a query tile q and an image tile t using

$$score(q,t) \;\; = \;\; d(q,b) - \lambda\, d(q,t) - c \qquad (2)$$

where λ and c are independent constants. $d(q,b)$ is the distance of the query tile q from the background tile b which is determined using domain knowledge. Comparing Eq. (2) with Eq. (1), we see that $f(q) = d(q,b)$ and $g(d(q,t)) = \lambda\, d(q,t)$.

3.2 Score of an Overlapping Region

Once we have a model to measure the similarity between a pair of tiles, we next consider how to measure similarity between two regions. The alignment or the overlap of a query image Q with a database image produces a score matrix of pairwise aligned tiles, as depicted in Figure 3. The score of the alignment is defined as the score of a *connected subregion* that has the *maximal* possible *cumulative* score. We are interested in the alignment of a single pattern and, hence, the justification for finding a single connected region. The maximal scoring subregion may include negative scores and may not be rectangular in shape, as shown in Figure 3.

The best match in a database of images is found by considering all possible alignments, i.e., translations of the query image over each database image. This is illustrated in Figure 4 where three alignments are shown.

3.3 NP-Completeness Proof

We next prove that the problem of finding the maximal scoring subregion in a score matrix is NP-hard. We prove this by showing that the corresponding decision problem is NP-complete. We first define the "graph" analog of the matrix problem as follows: *Given*

a graph representation $G = (V, E)$ of a matrix, with weight $w(v)$ on each vertex $v \in V$ corresponding to the entry in the matrix, is there a connected subgraph of weight $\geq W$? We denote this problem by MAXIMAL WEIGHTED CONNECTED SUBGRAPH or MWCS.

THEOREM 1. MAXIMAL WEIGHTED CONNECTED SUBGRAPH (MWCS) *is NP-complete, for a matrix graph of degree at most 4.*

PROOF. MWCS is in NP since the weight of a connected subgraph can be computed in polynomial time.

For reduction, we use the RECTILINEAR STEINER TREE (RST) problem that is known to be NP-complete [11]. The RST problem asks: Given a set of n terminal points that are embedded in an integer grid in a plane, is there a spanning tree of total length at most l such that the vertices of the spanning tree are the input points of the set and the grid points, where the length of an edge is the L_1 distance between the corresponding vertices?

There is a special case of the RST problem known as the THUMB-NAIL RECTILINEAR STEINER TREE (TRST) problem [10]. The TRST problem restricts the terminal points to an $m \times m$ grid. The TRST problem remains NP-complete even when m is bounded by a polynomial of n [11].

Given an instance of the TRST, we construct an instance of the MWCS as follows: We first find the bounding box of the points of the TRST, i.e., the $m \times m$ grid. Then, we replace each terminal point by a vertex of weight $w \gg l$. At each grid point that is not already occupied by the n terminal points, we place a vertex with weight 0. Between a pair of consecutive vertices on the same grid line (e.g., on the half-grid positions), we place a vertex with weight -1. Each vertex is connected to only to its horizontal and vertical neighbors, thus producing a matrix graph. Figure 5 shows an example of the construction. The original points are shown by double circles. The construction takes time polynomial in m, and hence polynomial in n, and the graph G thus constructed is planar with degree at most 4.

We claim that the original TRST on n points has a rectilinear Steiner tree of length $\leq l$ if and only if the MWCS graph has a connected subgraph of weight $\geq W = n.w - l$.

Only if: Assume that there is a Steiner tree of length at most l. By definition, it spans all the terminal points and is connected. Note that for a length l path between two points, there are exactly l vertices of weight -1. The vertices corresponding to the n terminal points have a weight of w each. Therefore, the weight of this tree is at least $n.w - l$. Figure 5 shows such a Steiner tree in solid lines.

If: Any connected subgraph of weight at least $n.w - l$ in G must include all the n vertices of weight w and at most l vertices of weight -1. There is no way to connect two vertices of weight ≥ 0 without passing through a vertex of weight -1. Therefore, the length of this path is at most l, since otherwise, the connected subgraph would have included more than l vertices of weight -1. Also, if the subgraph has the maximal weight, it is a tree, since, if it is not, at least one pair of vertices has more than one path between them. Removing that path increases the weight of the tree by the absolute weight of the negatively-weighted vertices in the path. Therefore, this subgraph defines a Steiner tree for the original n points. An example of such a subgraph is shown in Figure 5 in solid lines. □

3.4 Dynamic Programming Heuristic

Now, we design a dynamic programming (DP) heuristic as an alternative to examining all possible subregions for finding the maximal score. Assume that the score in cell $C(i, j)$ of the score matrix
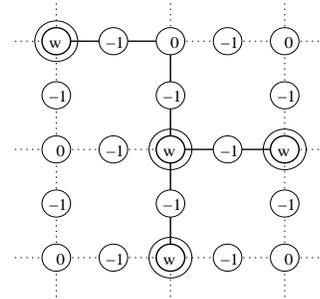

Figure 5: Construction from Thumbnail Rectilinear Steiner Tree instance to Maximal Weighted Connected Subgraph (MWCS) instance. The double lined vertices are the original terminal points. The solid lines represent the optimal solution of both the problems.

is denoted by $s(i, j)$. The DP starts from one of the corner cells of the score matrix. For discussion purposes, assume that it starts at cell $C(0, 0)$ in Figure 6. Next, it proceeds by first moving towards the right ($\rightarrow$) and calculates a subregion corresponding to each cell in 0th row. Then it goes to 1st row 0th cell $C(1, 0)$ by moving in the top ($\uparrow$) direction in the score matrix (akin to a row-scan order). DP completes this iteration when it reaches the top-most and right-most cell in the matrix. In Figure 6, this cell is $C(2, 2)$.

Suppose, $R(i, j)$ is a maximal scoring sub-region that has its top-right corner at the cell $C(i, j)$. Also, suppose $s(i, j)$ denotes the score of $C(i, j)$ and $S(i, j)$ denotes the maximal score for the subregion $R(i, j)$. DP examines 4 possibilities to find the maximal score of $R(i, j)$: (i) the score of the cell itself, (ii) the score of the cell plus the maximal score for the bottom subregion, (iii) the score of the cell plus the maximal score for the left subregion, and (iv) the score of the cell plus the maximal scores for the bottom and the left subregions. Since the bottom and the left subregions can intersect, the score of the intersecting region should be subtracted from the cumulative scores of the two subregions so that it is not counted twice.

The DP algorithm computes the following recurrence relation to find all the subregions in the score matrix and their scores:

$$S(i,j) = \max \begin{cases} s(i,j) \\ s(i,j) + S(i, j-1) \\ s(i,j) + S(i-1, j) \\ s(i,j) + S(i, j-1) + S(i-1, j) \\ \quad - S\big(R(i, j-1) \cap R(i-1, j)\big) \end{cases} \quad (3)$$

The corresponding subregions maintained for the 4 cases are, respectively:

$$R(i,j) = \begin{cases} C(i,j) \\ C(i,j) \cup R(i, j-1) \\ C(i,j) \cup R(i-1, j) \\ C(i,j) \cup R(i, j-1) \cup R(i-1, j) \end{cases} \quad (4)$$

To improve the overall score, DP executes the above logic starting from all the 4 corner cells with the following combinations of moves: (i) Starting at bottom-left cell and moving in $\uparrow$ and $\rightarrow$ direction, (ii) Starting at bottom-right cell and moving in $\uparrow$ and $\leftarrow$ direction, (iii) Starting at top-left cell and moving in $\downarrow$ and $\rightarrow$ direction, (iv) Starting at top-right cell and moving in $\downarrow$ and $\leftarrow$ direction. It returns the subregion having the maximum score of all these 4 possibilities. Such a subregion explored by DP on a score matrix is illustrated by Figure 3.
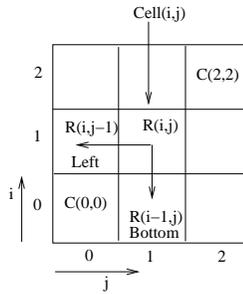
Figure 6: DP forms sub-region $R(i,j)$ by looking at scores of $C(i,j)$, $R(i-1,j)$ and $R(i,j-1)$.

Figure 7: Example of a shape not captured by DP. The scores are shown in brackets. The optimal solution consists of the cells $(3,3)$, $(2,2)$, $(2,3)$, $(2,4)$ and $(1,3)$ having scores 40, 10, 1, 35 and 10 respectively.

Running Time: For a score matrix of size $m \times n$, for each cell, the DP computes the maximal score for the subregion ending at that cell. Calculating the scores for each cell requires finding an intersection of the largest scoring subregions on its bottom and left. This requires a running time of $O(mn)$ in the worst case. Thus, the total running time of the DP algorithm is $O(m^2n^2)$. For a particular score matrix, the DP needs to be run from all the 4 corners, which is constant. Thus, the worst case running time for the DP is *quadratic* in the size of the score matrix.

Class of Subregions Examined: The DP algorithm does not (and cannot!) investigate all the possible connected subregions; it chooses the maximal scoring connected subregion from only a certain class of shapes. Next, we analyze the class of such shapes. Consider only right ($\rightarrow$) and top ($\uparrow$) moves starting at left-bottom corner. The maximal scoring subregion $R(i,j)$ for cell $C(i,j)$ will include another cell $C(i',j')$ only if $C(i',j')$ is included in either $R(i,j-1)$ or $R(i-1,j)$. Similarly, the subregions $R(i,j-1)$ and $R(i-1,j)$ contain only those cells that are towards the left and bottom of them. Therefore, by induction, if cell $C(i',j')$ is included in $R(i,j)$, then $C(i',j')$ must be at the left and bottom of $C(i,j)$. No cell which is towards the right or top of $C(i,j)$ will be included in $R(i,j)$. As an example, consider the cell $(2,4)$ in Figure 7. It can consider only the cells (i,j) where $i \leq 2$ and $j \leq 4$, i.e., all cells to its left and bottom. It cannot consider any other cell (e.g., cell $(3,3)$ in the figure). The DP ends at the top-right corner. However, the maximal scoring subregion may end at any cell, and not necessarily at the top-right corner cell. Thus, for example, if the maximal scoring sub-region ends at $(3,3)$, then the nature of DP forbids it to consider cells $(1,4)$, $(2,4)$ and $(3,4)$ (Figure 7). Hence, even though the optimal solution for the example in Figure 7 consists of all the five shaded cells, this DP will find only the region consisting of cells $(1,3)$, $(2,2)$, $(2,3)$ and $(3,3)$ as the answer. The shaded region given in Figure 7 cannot be obtained by DP starting from any corner.

Since the DP is run from the four corners in four sets of moves, the subregions captured are of four types: containing cells (i) to-
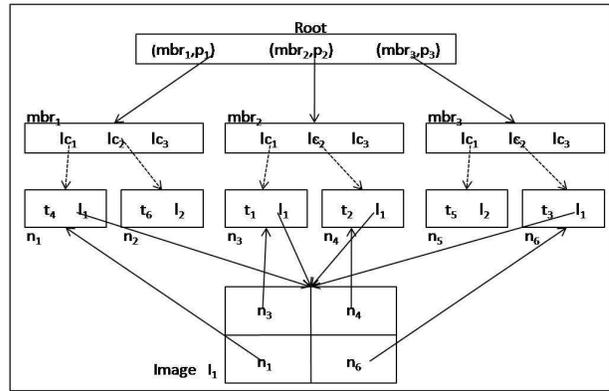

Figure 9: Index structure. Image I_1 maintains pointers to leaf nodes of its tiles. Leaf nodes maintain pointer to I_1.

wards bottom and left, (ii) towards bottom and right, (iii) towards top and left, and (iv) towards top and right.

Formally, the shapes for the class of such subregions can be characterized in the following way. For a particular shape P, a cell $C(i,j)$ *sinks* another cell $C(i',j')$, denoted by $C(i,j) \triangleleft C(i',j')$, if $C(i,j)$ can be reached from $C(i',j')$ in P by taking one of the four combinations of moves described earlier. For example, in Figure 7, cell $(3,3)$ sinks cell $(2,3)$ for right and top moves since it can be reached from $(2,3)$ by this move combination. For the same move combination, it does not sink cell $(2,4)$ as it cannot reached from $(2,4)$ using only right and top moves. A cell $C(i,j)$ *sinks* a shape P, denoted by $C(i,j) \triangleleft P$, if and only if for all cells $C(i',j')$ belonging to P, $C(i,j)$ sinks $C(i',j')$, i.e.,

$$C(i,j) \triangleleft P \iff \forall C(i',j') \in P, \quad C(i,j) \triangleleft C(i',j') \quad (5)$$

A particular shape P can be captured by DP if and only if there exists a cell $C(i,j) \in P$ that sinks P. Combining all the 4 sets of moves as mentioned earlier characterizes the entire set of shapes captured by DP. Examples of shapes captured are: ⊓, □, etc. Shapes that cannot be captured include $+$, $\times$. We also present few examples of the count of the shapes captured by DP for varying number of grids. DP captures all the possible 14 shapes for a 2×4 grid. DP captures 31 of the 38 possible shapes for a 3×3 grid.

Advantages of Score Based Similarity: We performed quality experiments to compare our score based similarity measure with distance based measures. We used the *Discriminator* scoring function [36] to measure the similarity between a pair of tiles. We compared the first result retrieved by our similarity measure to the sum of L_1 distance measured between corresponding histogram of tiles of the overlapping region. One such result over biological images is shown in Figure 8. We can see that a simple distance measure fails to discriminate between foreground and background and hence generates more false matches. Our similarity measure maximizes the score of the best matching subregion and performs better.

4. QUERY ALGORITHMS

In this section, we discuss linear search and two new query algorithms to find the top-k similar regions from a database. These strategies are general enough to work with other scoring schemes and heuristics. The first algorithm is a naïve linear search through the database. The other two algorithms use a multi-dimensional index structure to prune the search space to achieve efficiency and scalability. In the ensuing discussion, the size of a query image Q

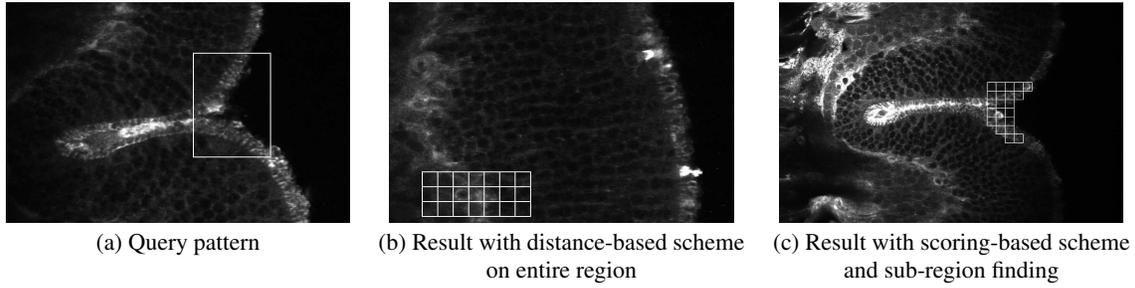| (a) Query pattern | (b) Result with distance-based scheme on entire region | (c) Result with scoring-based scheme and sub-region finding |

Figure 8: (a) Example of a biologically interesting pattern. The marked pattern highlights a fold of the retinal tissue labeled with peanut-agglutinin conjugated to a fluorescent probe. (b) Retrieved result for distance-based matching on entire region. (c) Retrieved result for score-based matching on subregions.

Algorithm 1 TARS

Input: T: tree root, Q: list of query tiles
Input: k: number of top results
Output: RQ: priority queue of top-k matches
 1: $RQ \leftarrow [(-,-\infty)]$
 2: $T \leftarrow +\infty$
 3: $BS \leftarrow \phi$: bit-vector for explored overlap region
 4: **while** T > GetHead(RQ).s **do**
 5:    **for all** $q_i \in Q$ **do**
 6:       nnTile[i] $\leftarrow$ GetNextNN(q_i)
 7:    **end for**
 8:    **for all** $q_i \in Q$ **do**
 9:       $org \leftarrow$ FindOverlapRegion(nnTile[i],q_i)
10:       **if** org not flagged in BS **then**
11:          $sm \leftarrow$ GetScoreMatrix(Q,org)
12:          $e(rg,s) \leftarrow$ DP(sm)
13:          flag org in BS
14:          Insert(RQ, $e(rg,s)$)
15:       **end if**
16:       $sm \leftarrow$ score matrix of overlapping Q with nnTile
17:       $T \leftarrow$ score of DP(sm)
18:    **end for**
19: **end while**

---

and a database image I is defined in terms of the number of constituent tiles. We take the size of Q to be n.

The *Linear Search* algorithm searches through all the possible overlaps to find the top-k matching regions. It translates the query image over all the database images and computes a score for each of them. It maintains a priority queue of the results to find the k highest scoring regions. Since the number of possible overlaps increases linearly with increase in image and database size, this method does not scale and is impractical for large queries and database sizes.

To make the search scalable and efficient, we next propose two algorithms TARS and SPARS. Both algorithms use an index on the feature vectors of the image tiles to query nearest neighbors for a given tile. We can use any R-tree [15] (data-partitioning) like index structure for this. We choose bulk loadable STR-Tree [20] as the index because of its simplicity and availability. Each leaf node of the index is an entry of the form *lmbr(t, I)* where t is the feature vector of an image tile and I is a pointer to its parent image. Each non-leaf node has a list containing an entry per child of type (MBR, child-pointer) where MBR is the minimum bounding box and child-pointer is the pointer to the child node respectively. Each database image I is a two-dimensional array of pointers to the *lmbrs* containing its tiles as shown in Figure 9. This structure allows for

q_1	q_2
$(t_3,I_2,10)$	$(t_1,I_1,9)$
$(t_2,I_1,8)$	$(t_2,I_1,7)$
$(t_1,I_1,2)$	$(t_3,I_2,6)$

Table 1: Sorted access of tiles for a given query (q_1,q_2) **in TARS.**

full access from a tile to its parent image and vice versa. We can easily find the row and column position of a tile from the image array to find an overlap.

4.1 TARS (Threshold Algorithm for Region-based Search)

The algorithm TARS formulates the region retrieval query as a top-k aggregate query. It views the given query image as a multi-component object where each of its tile $q_i \in Q, \forall i = 1, \dots, n$ constitutes its independent components. Similarly, it views DB as a list of multi-component objects. Each DB object is a set of connected tiles from a DB image. It uses sub-region similarity function as aggregate function. Query image (query object) is overlapped with a connected set of tiles from an image (DB object), a score matrix is obtained by computing pairwise tile similarity, and then maximum similarity score is computed using DP on the score matrix. The goal in this setting now is to retrieve the top-k maximum scoring DB objects.

Algorithm TARS adopts a strategy similar to the Threshold Algorithm (TA) [7] to solve this aggregate query. For each q_i, TARS views that all the database tiles are ranked in a decreasing order of their scores with q_i. Overlapping regions are determined by the tiles from the ranked lists. Table 1 shows a sorted view of the database for a query image consisting of two tiles. Maximum similarity score computed using DP is monotonic. For two score matrices a and b obtained by overlapping a query object with two different DB objects, if a has tile-wise larger scores than b, then the score of DP on a will be larger than b. This monotonic property is used to terminate TARS.

TARS performs incremental nearest-neighbor searches for each q_i on the index structure to get sorted access to the database tiles. It starts by accessing the first nearest neighbor t_{i_1} for each q_i (steps 5-7 of Algorithm 1). Then, for each q_i, it finds the overlapping region of Q with a database image I ($t_{i_1} \in I$) such that q_i aligns with t_{i_1} in the overlap. Figure 10 shows how Q having 4 tiles overlaps with an image I also having 4 tiles such that q_1 aligns with t_1.

The algorithm then uses DP to find the maximum scoring subregion rg and its score s for each such overlap org (steps 11-12). It inserts all the results $e(rg,s)$ in a priority queue RQ of size at most k. The entries in RQ are sorted based on their scores. Once RQ
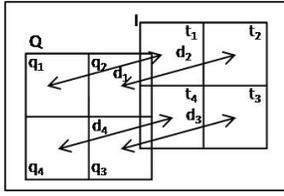
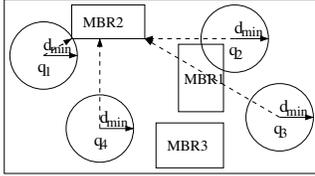**Figure 10: Overlap of query image Q with database image I such that q_1 aligns with t_1.**

Figure 11: MBR and its nearest query tile. q_1 **is nearest to** MBR_2 **with distance** d_{min}**.**

has k regions, a result is inserted only if its score is more than the kth current region with the least score. In order to prevent multiple processing of the same database region, TARS flags the explored overlapping regions in a bit-vector BS.

At the end of the first iteration, TARS builds a score matrix by aligning each q_i with t_{i_1}. The DP score on this score matrix is the *threshold* score T. The threshold score is an upper bound on the scores of all the regions that have not yet been explored; this is because all the tiles to be accessed in the next iteration by each q_i have scores lower or at best equal to the current t_i's and the DP score is monotonic. This threshold score is updated after every iteration of the algorithm. The algorithm proceeds to the next iteration only if T is greater than the least score in *RQ*. As TARS proceeds, T decreases and the algorithm terminates with optimal results.

The performance of algorithm TARS worsens with increase in query size. It is instance optimal but it traverses the index structure separately for each $q_i \in Q$ to access the database tiles in sorted order. The cost of this multiple nearest-neighbor traversal grows quickly with increasing query size. To avoid this scalability problem, we next propose a technique SPARS that finds the top-k regions by performing a single traversal through the index structure and has better performance than TARS for large queries.

4.2 SPARS (Single Pass Region-based Search)

Algorithm SPARS is a novel top-k aggregate query algorithm which makes a single traversal through the index tree. It finds the top scoring regions by performing a *best-first search* [17]. It maintains a priority queue *BQ* to find the next best node to process. When the algorithm encounters a leaf node, it explores an actual region in the database corresponding to its image tile. Similar to TARS, it maintains a priority queue *RQ* of the top-k regions.

The search for top-k regions starts at the root node of the index, which is the first entry in *BQ* with $+\infty$ score. The algorithm next processes each of its children. If the child is a non-leaf index node *mbr*, then it computes a score for it as follows (outlined in steps 8-12 of Algorithm 2). It determines the minimum distance between any query tile and the node $d_{min} = \min_i d(q_i, mbr)$, as shown in Figure 11. Then, it computes a score matrix by aligning each $q_i \in Q$ with *virtual* image tiles having a distance d_{min} from q_i as shown in Figure 12. The score s of the node is the score of the maximum scoring subregion found using DP on this score matrix.

Algorithm 2 SPARS

Input: T: tree root, Q: list of query tiles
Input: k: number of top results
Output: RQ: priority queue of top-k matches
1: $n \leftarrow \text{size}(Q)$
2: RQ $\leftarrow [(-, -\infty)]$
3: BQ: queue of intermediate entities$\leftarrow [(T, +\infty)]$
4: e(mbr,s) $\leftarrow$ GetHead(BQ)
5: **while** e.$s \geq$ GetHead(RQ).s **do**
6:     **if** e is of type (mbr,s) **then**
7:         **for all** child node cn in e.mbr **do**
8:             **if** cn is mbr **then**
9:                 $d_{min} \leftarrow$ GetMinDistance(Q,mbr)
10:                $sm \leftarrow$ GetScoreMatrix(Q,$[d_{min}]$)
11:                e(rg,s) $\leftarrow$ DP(sm)
12:                Insert(BQ,e(mbr,s))
13:             **else**
14:                /*if cn is a $lmbr$*/
15:                $q_j \leftarrow$ query tile nearest to $lmbr$
16:                e(rg,s) $\leftarrow$ GetMaxSubRg($lmbr$,q_j,Q)
17:                RQ.Insert(e(rg,s))
18:                **for all** q_i in Q and $i \neq j$ **do**
19:                   $d_{min} \leftarrow$ GetMinDistance(q_i,$lmbr$)
20:                   $sm \leftarrow$ GetScoreMatrix(Q,$[d_{min}]$)
21:                   e($lmbr$,q_i,s) $\leftarrow$ DP(sm)
22:                   Insert(BQ,e($lmbr$,q_i,s))
23:                 **end for**
24:             **end if**
25:         **end for**
26:     **else**
27:         /*if e is of type $(q_j,lmbr,s)$*/
28:         e(rg,s) $\leftarrow$ GetMaxSubRg($lmbr$,q,Q)
29:         Insert(RQ,e(rg,s))
30:     **end if**
31:     e(mbr,s) $\leftarrow$ GetHead(BQ)
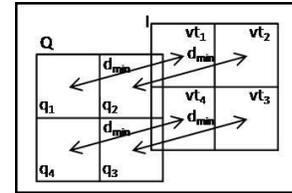32: **end while**
33: **return** RQ

Figure 12: Overlap of query image Q with virtual tiles $(vt_1, vt_2, \dots)$ **at distance** d_{min}**.**

SPARS inserts the *mbr* along with the score *s* as an element *e(mbr,s)* in *BQ*. It inserts *e* into *BQ* only if *RQ* has less than k elements or *s* is greater than the minimum score in *RQ*.

If the child is a leaf node *lmbr*, then the algorithm finds the nearest query tile q_i to tile t of *lmbr* and computes the minimum distance $d_{min} = \min_i d(q_i, lmbr)$. It explores the actual image region for the (q_i,t) alignment using Algorithm 3 as illustrated in steps 15-17 of Algorithm 2. Algorithm *GetMaxSubRg* finds the overlap of query *Q* with image *I* pointed by *lmbr* by aligning q_i with t. Since the same overlapping region can be encountered later for a query and image tile pair, *GetMaxSubRg* maintains a bit-vector *BS* to flag the explored regions; this prevents multiple processing

Algorithm 3 GetMaxSubRg (SPARS)

Input: $lmbr$: leaf node, q_i: query tile, Q: query tiles list
Output: e: entity of maxsubregion and score
1: BS: bit-vector for explored overlap region
2: $org \leftarrow$ FindOverlapRegion($lmbr,q_i$)
3: **if** org not flagged in BS **then**
4:    $dm \leftarrow$ GetDistanceMatrix(org)
5:    $sm \leftarrow$ GetScoreMatrix(Q,dm)
6:    e(rg,s) $\leftarrow$ DP(sm)
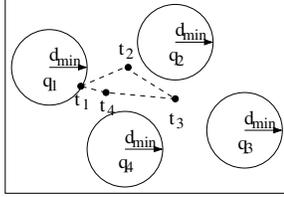7:   flag org in BS
8: **end if**
9: **return** e(rg,s)

Figure 13: Tiles of the overlapping region for q_1 aligning with t_1 lie at distances greater than d_{min}.

of the same database region. Algorithm *GetMaxSubRg* returns the maximum scoring subregion *rg* of the overlap and the corresponding score *s* using DP. The result *e(rg,s)* is inserted into *RQ* only if *RQ* has less than *k* elements or *s* is greater than the minimum score in *RQ*.

After processing the alignment of q_i with t, we still need to process the other alignments corresponding to other query tiles and t. The SPARS algorithm delays exploring these alignments in order to save computation cost. It calculates a score s of the *lmbr* for each $q_j, j \neq i$ using the same method discussed for a *mbr* (outlined by the steps 18-23 of Algorithm 2). It finds the minimum distance d_j between q_j and tile t in *lmbr*. It overlaps the query image with a virtual image such that the distance between each aligned pair of tiles is d_j. DP is run on this score matrix to compute score s of the maximum scoring subregion for the overlap. SPARS inserts elements e(*lmbr*,q_j,s) in *BQ* for each q_j only if *RQ* has less than *k* elements or s is greater than the minimum score in *RQ*. The algorithm explores these regions during the access of the elements from *BQ* as outlined by steps 28-29 of Algorithm 2.

SPARS proceeds by accessing the current highest scoring element from *BQ* and terminates when the lowest score in *RQ* is greater than the highest score in *BQ*.

Pruning Strategy: The scoring function $s(q,t)$ is a monotonically decreasing function of $d(q,t)$, as discussed in Section 3.1. The aggregate score of an overlap is also monotonic with respect to individual scores of the score matrix of an overlap. With this monotonicity property, the following lemma holds. SPARS uses this lemma to prune the search space.

LEMMA 1. *The score $S(Q,I)$ of the overlap of a query image Q with an image I with $d(q_i,t_i) = r$, $\forall i$, is an upper bound on the score $S(Q,I')$ of all possible overlaps of Q with image I' provided $d(q_i,t_i') \geq r$, $\forall i$.*

From this lemma, we see that the score s of a node (*mbr* or *lmbr*) at a minimum distance d_{min} from the query tiles is an upper bound on the score of all nodes whose minimum distance is greater than d_{min}. We visualize such an example in Figure 11 where MBR_2
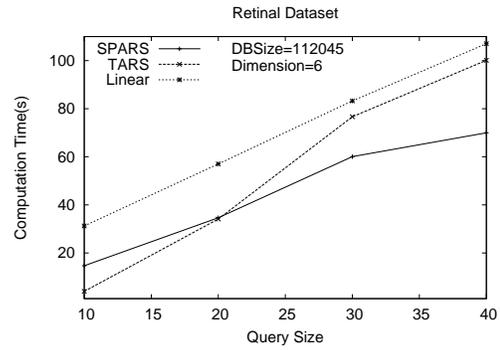

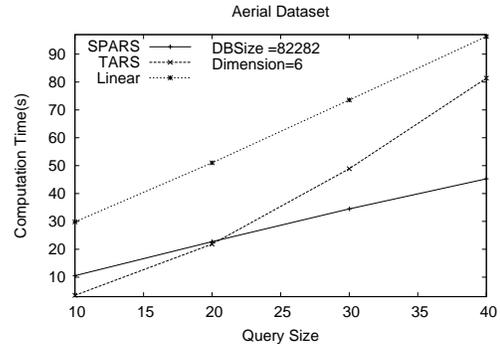**Figure 14: Effect of query size on the performance of the algorithms for retinal images.**

Figure 15: Effect of query size on the performance of the algorithms for aerial images.

is at a distance of d_{min} but MBR_3 is at a greater (minimum) distance from all the query tiles. Therefore, score of MBR_3 will be less than MBR_2. The score s is also an upper bound on the score of an actual overlapping region if the distance between the corresponding tiles of *Q* and *I* have distance greater than d_{min}. We have such a scenario in Figure 13 in which tile t_1 finds q_1 as its nearest neighbor. All the tiles of the corresponding overlap as shown in Figure 10 lie at a distance greater than d_{min}. Therefore, the score of this overlap is less than the score s of a node. These facts justify that the score of an element in *BQ* is an upper bound on all the nodes and regions that have not been explored and are ranked lower in *BQ*. At any point during the search, SPARS has already explored a hypersphere of radius d_{min} centered at each query tile if the next candidate from *BQ* has a minimum distance d_{min} from all the query tiles.

SPARS processes the nodes in decreasing order of their scores. It explores all nodes having score greater than the least score in *RQ* since they are potential candidates to yield regions with higher score. It terminates the search once the minimum score in *RQ* becomes more than the highest score in *BQ*. Thus, this pruning strategy ensures an optimal result for SPARS.

5. EXPERIMENTAL STUDIES

In this section, we first empirically analyze the performance and efficiency of our access methods. Then, we present detailed quality analysis of our new similarity measure with visual results. We used Java 5.2 as our implementation language. We performed experiments on a 3.2 GHz, 4 GB memory PC running Debian Linux 4.0.

5.1 Dataset Preparation

We used two large real image datasets belonging to raster im-

Reduced dimension	Energy retained	
	Retinal dataset	Aerial dataset
3	85.73%	81.21%
6	96.14%	93.38%
13	98.94%	97.55%

Table 2: Percentage energy remaining after PCA.

Retinal dataset		Aerial dataset	
Image count	Region count	Image count	Region count
112,045	10,004,850	82,282	10,625,200
56,241	5,000,050	37,037	5,000,000
33,762	3,000,000	21,744	3,000,000
11,112	1,000,100	5,560	1,000,000

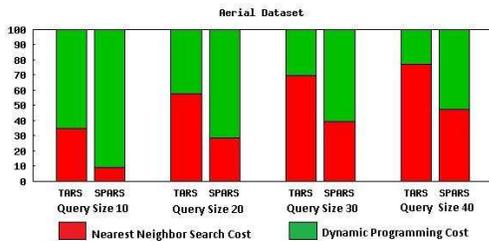**Table 3: Database sizes of retinal and aerial images.**

Figure 16: Percentage split of NN and DP time for varying query sizes for TARS and SPARS for aerial images.

age family to empirically analyze the efficiency and scalability of our algorithms. The first dataset consists of 112,045 gray-scale images of various tissues and layers of retina [9] from different experimental conditions. Multiple molecular probes such as lectins and antibodies were used to examine the localizations of specific protein expression in retinal cells and the expression patterns of these proteins in different layers of retina. The fluorescence tagged probes were imaged by immunohistochemistry using confocal microscopes. We used the magnification of these images to scale them to a standard magnification using the *CubicFilter* from Graphics-Magick[2]. Our second dataset consists of 82,282 gray-scale aerial images from the Alexandria Digital Library[3]. These are satellite images and air photos of different regions of California. The size of the images in both datasets varied from 320×160 pixels to 640×480 pixels.

We split the images into non-overlapping tiles of size 32×32 pixels and compute feature vector for each tile. The feature vector of each tile is a histogram of its pixel values similar to CSD feature vector [24]. To enhance efficiency, we performed PCA [30] on these feature vectors to reduce the dimensionality. The number of principal components retained and the corresponding energy preserved is shown in Table 2. The index structure was built on this transformed data. We used the *Discriminator* scoring function [36] to measure the similarity between a pair of tiles. We discuss the choice of parameters for the scoring function later in Section 5.4.

The parameters that are crucial to the performance of the access methods are: (i) Query size, n (ii) Database size, N, and (iii) Dimensionality of the feature vector, dim. Query size n is defined as the number of constituent tiles in the query image. Database

[2]http://www.graphicsmagick.org/
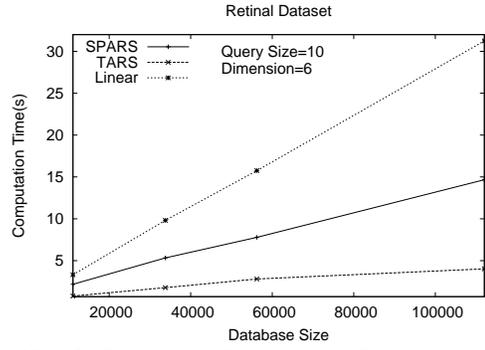[3]http://www.alexandria.ucsb.edu/

Figure 17: Performance of algorithms for varying database sizes of retinal images for query size 10.
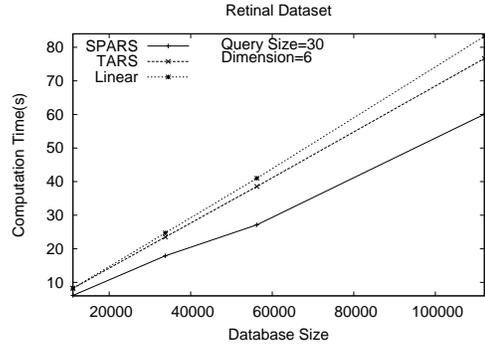

Figure 18: Performance of algorithms for varying database sizes of retinal images for query size 30.
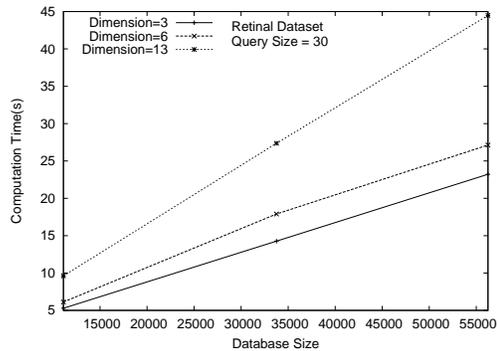

Figure 19: Effect of database size and dimension on the performance of SPARS on retinal images.

size N is defined as the number of images. The number of images and possible overlapping regions obtained by translation for varying N is described in Table 3 for both retinal and aerial datasets. For each experiment, we used 100 randomly picked queries from the dataset. All the reported time measurements are averaged over these 100 queries for top-10 results.

5.2 Performance Comparison of the Algorithms

We experimented with varying query sizes to compare the performance of the algorithms. We use the largest datasets of size $N = 112,045$ for retinal images and $N = 82,282$ for aerial images with $dim = 6$ for this experiment. Our results show that both TARS and SPARS outperformed linear search on both the datasets, as shown by Figures 14 and 15. Comparing TARS with SPARS on the aerial dataset, we found TARS to be 3 times faster for query
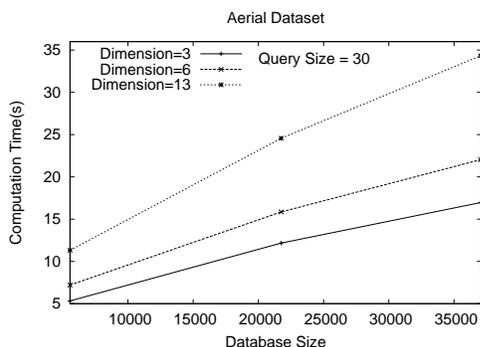
Figure 20: Effect of database size and dimension on the performance of SPARS on aerial images.
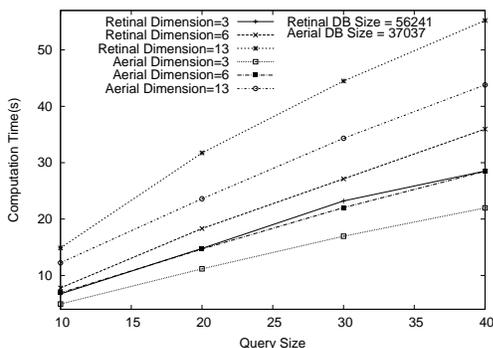

Figure 21: Effect of query size and dimension on the performance of SPARS on retinal and aerial images.
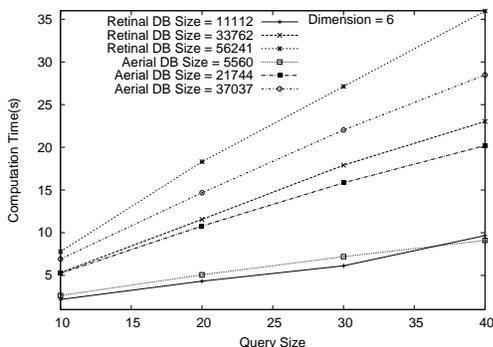

Figure 22: Performance of SPARS for varying query sizes and database sizes of retinal and aerial images.

size 10 but slower by 2 times for query size of 40 than SPARS (Figure 15). The same behavior was noticed for the retinal dataset where TARS was faster by 3.6 times for $n = 10$ but slower by 1.4 times for $n = 40$ than SPARS (Figure 14).

SPARS performs better than TARS for query sizes of more than 20. We attribute this change in performance of TARS to its multiple traversal through the index structure, as discussed in Section 4.1. We measured the average of total nearest-neighbor search cost *NN* and dynamic programming *DP* cost for varying query sizes for both the algorithms TARS and SPARS. We present the percentage of time spent by each algorithm on NN and DP in Figure 16. We observe that the *NN* cost increases faster in TARS compared to SPARS as query size increases. TARS is instance optimal [7] and, therefore, it performs better than linear search and SPARS for

Dataset	Images	Queries	λ	c	Precision
PA Retinal	80	18	1	23000	80.3%
NF Retinal	37	8	1	23000	82.5%
Aerial	550	7	1	115000	88%

Table 4: Datasets used for quality analysis, corresponding parameter values for scoring function, and precision measures.

smaller query sizes when the cost of this multiple traversal is not high. As this cost increases with increase in query size, its performance is poorer as compared to SPARS.

We next experimented with varying database sizes for the retinal dataset to confirm the above behavior of the algorithms. For a query size of $n = 10$ and $dim = 6$, we found TARS to be more than 2.7 times faster than SPARS across the database sizes as shown Figure 17. SPARS is more than 1.5 times faster than linear search. The performance difference increases with increase in database size. Our other experiment with $n = 30$ and $dim = 6$ found SPARS to be 1.3 time better than TARS and more than 1.3 times better than linear search (Figure 18).

We see from the results discussed above that TARS is a better algorithm than the other two for $n \leq 20$ whereas SPARS is better for $n > 20$. TARS saves more than 87% of the query time for $n = 10$ on both the datasets for the largest size. SPARS has a saving of 34% on retinal and 52% on aerial for $n = 40$ on the largest datasets. The average query time of TARS is approximately $4\,s$ on a database of size $N = 112,045$ and a query size of $n = 10$. The average query time for SPARS on the same database is $70\,s$ for query size of 40.

5.3 Performance Analysis of SPARS

We next performed detailed analysis of the behavior of the algorithm SPARS for varying n (query size), N (database size) and dim (dimension) on both the datasets. The performance results of SPARS on both datasets for varying N and dim are shown in Figures 19 and 20. The dataset size is 56,241 for retinal images and 37,037 for aerial images. SPARS scales linearly for a given query size across varying database sizes and dimensions. The performance of SPARS on both datasets for varying n and dim is shown in Figure 21. SPARS exhibited linear scalability for a given database size across varying query sizes and dimensions. Experiments with varying N and n for $dim = 6$ on both datasets also revealed a linear scalability performance as shown in Figure 22.

The exhaustive set of empirical results discussed above confirms a sub-linear scalability for SPARS across varying query size and database size compared to a linear scan of the database. Its scalability is also sub-linear for low dimensions compared to linear scan. This establishes the scalability and efficiency of our algorithm.

5.4 Quality Analysis

In this section we analyze the quality of our similarity measure and describe the datasets used for experiments.

Dataset Preparation: We used 3 different datasets to verify the quality of our new similarity measure. From each dataset, we chose interesting regions for querying. For each query, regions in images were manually tagged as a true or a false match. Since the process is manually intensive, we used small datasets as shown in Table 4. PA and NF datasets are confocal microscopic images of cross-sections of feline retina labeled with the lectin peanut-agglutinin and anti-neurofilament antibody respectively. Aerial data-

set consists of satellite images of Beverly Hills in California.

Parameter Learning and Precision: Here, we discuss the choice of parameters for the *Discriminator* scoring function [36]. We use pure black as background for retinal images which is true for the most of the real microscopic images. We use pure black for aerial images also for the purpose of simplicity, though, it can have other backgrounds. Background for other domains need to be determined from knowledge and training. We take the sum of all the pixels in a tile as its distance from background. Distance between tiles is measured using L_1 norm.

We learn the parameter λ and c using manual training with an approach similar to Walrus [28]. For each query, we measured *top-k* precision where $k = 5$ and precision is the ratio of true matches to total matches. We trained the PA dataset on 10 queries. Highest precision (82.0%) was achieved for $\lambda = 1$ and $c = 23000$. These parameters gave an accuracy of 78.6% for 8 other queries over PA dataset. For the same parameter values, 8 queries on NF dataset gave precision of 82.5%. Training and testing on 7 aerial image queries had a precision of 88% for $\lambda = 1$ and $c = 115000$. We summarize the results in Table 4. The same parameter values were used for scalability and efficiency measurements in Section 5.2. Our size of the training set was limited by the manually intensive nature of the task.

Finally, we present results for a set of queries from these three datasets in Figure 23. All the match for retinal images have been validated by domain scientists and found to be significantly interesting. As shown in the first row of Figure 23 , query and the results are examples of biologically interesting fold of retinal tissues labeled with peanut-agglutinin.

6. CONCLUSIONS

In this paper, we addressed the problem of querying significant subregions in raster images. We designed a generic scoring scheme to measure similarity between a query image and an image region. We tiled the images to represent a region as a collection of tiles, and each overlap between a query and a database image as a matrix of scores. We proved that the problem of finding a connected subregion of maximal score in a score matrix is NP-hard and then developed a dynamic programming heuristic to score an overlapping region. With this similarity measure, we proposed two index-based scalable search strategies TARS and SPARS for querying in a large repository. These strategies are general enough to work with any scoring scheme and heuristic. We empirically analyzed the performance of these algorithms on datasets of 112,045 retinal images and 82,282 aerial images. We save more than 87% search time on small queries using TARS and up to 52% search time on large queries with SPARS on these datasets as compared to linear search. It should be noted that our heuristic for finding the best connected subregions and our access methods for top-k queries (TARS and SPARS) are independent of each other. We demonstrate the quality of our similarity measure (more than 80% precision) with analysis over two real datasets. The ability to extract significant subregions (connected regions with highest score) can have a significant impact on analyzing raster images. Future work includes the formulation of other heuristics for finding similar subregions that have bounded approximation errors on quality and the formulation of other domain-specific scoring functions.

7. ACKNOWLEDGMENTS

8. REFERENCES

[1] I. Bartolini, P. Ciaccia, and M. Patella. A Sound Algorithm for Region-Based Image Retrieval Using an Index. In *DEXA Workshop*, pages 930–934, 2000.

[2] P. Baumann. Web-enabled raster gis services for large image and map databases. In *DEXA*, page 870, 2001.

[3] S. Berretti, A. D. Bimbo, and E. Vicario. Spatial Arrangement of Color in Retrieval by Visual Similarity. *Pattern Recognition*, 35(8):1661–1674, 2002.

[4] C. Carson, S. Belongie, H. Greenspan, and J. Malik. Blobworld: Image Segmentation Using Expectation-Maximization and Its Application to Image Querying. *PAMI*, 24(8):1026–1038, 2002.

[5] R. Datta, J. Li, and J. Z. Wang. Content-Based Image Retrieval: Approaches and Trends of the New Age. In *MIR '05: Int. Workshop on Multimedia Information Retrieval*, pages 253–262, 2005.

[6] T. L. Department, M. Güld, C. Thies, and T. M. Lehmann. Content-based image retrieval in medical applications. In *Procs. Int. Society for Optical Engineering (SPIE*, pages 312–320, 2000.

[7] R. Fagin, A. Lotem, and M. Naor. Optimal Aggregation Algorithms for Middleware. In *PODS*, pages 102–113, 2001.

[8] R. Finkel and J. L. Bentley. Quad Trees: A Data Structure for Retrieval on Composite Keys. *Acta Informatica*, 4(1):1–9, 1974.

[9] S. K. Fisher, G. P. Lewis, K. A. Linberg, and M. R. Verardo. Cellular Remodeling in Mammalian Retina: Results from Studies of Experimental Retinal Detachment. *Progress in Retinal and Eye Research*, 24(3):395–431, 2005.

[10] J. L. Ganley and J. P. Cohoon. The Rectilinear Steiner Tree on a Checkerboard. *ACM Trans. Design Automation of Electronic Systems*, 1(4):512–522, 1996.

[11] M. R. Garey and D. S. Johnson. The Rectilinear Steiner Tree Problem is NP-Complete. *SIAM J. on Applied Mathematics*, 32(4):826–834, 1977.

[12] M. Gertz, Q. Hart, C. Rueda, S. Singhal, and J. Zhang. A data and query model for streaming geospatial image data. In *EDBT Workshops*, pages 687–699, 2006.

[13] A. G. Gutierrez. Applying OLAP pre-aggregation techniques to speed up query response times in raster image databases. In *ICSOFT (ISDM/EHST/DC)*, pages 259–266, 2007.

[14] A. G. Gutierrez and P. Baumann. Modeling fundamental geo-raster operations with array algebra. In *ICDM Workshops*, page 607, 2007.

[15] A. Guttman. R-Trees: A Dynamic Index Structure for Spatial Searching. In *SIGMOD*, pages 47–57, 1984.

[16] M. Hadjieleftheriou, G. Kollios, P. Bakalov, and V. J. Tsotras. Complex spatio-temporal pattern queries. In *VLDB*, 2005.

[17] G. R. Hjaltason and H. Samet. Distance Browsing in Spatial Databases. *ACM Trans. Database Syst.*, 24:265–318, 1999.

[18] Y. Ke, R. Sukthankar, and L. Huston. An efficient parts-based near-duplicate and sub-image retrieval system. In *MM*, pages 869–876, 2004.

[19] S. Lazebnik, C. Schmid, and J. Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *CVPR*, pages 2169–2178, 2006.
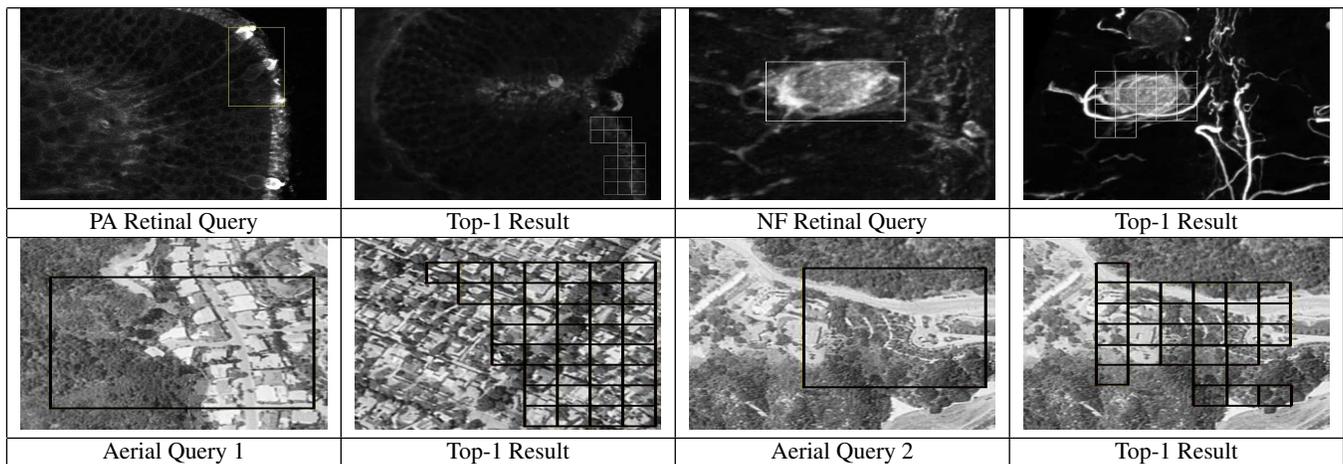
| PA Retinal Query | Top-1 Result | NF Retinal Query | Top-1 Result |
| Aerial Query 1 | Top-1 Result | Aerial Query 2 | Top-1 Result |

Figure 23: Top-1 result for various queries from three real datasets.

[20] S. T. Leutenegger, J. M. Edgington, and M. A. Lopez. STR: A simple and efficient algorithm for R-tree packing. Technical report, Institute for Computer Applications in Science and Engineering (ICASE), 1997.

[21] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *IJCV*, 60:91–110, 2004.

[22] J. Malki, N. Boujemaa, C. Nastar, and A. Winter. Region Queries without Segmentation for Image Retrieval by Content. In *Visual Information and Information Systems (VISUAL)*, pages 115–122, 1999.

[23] B. S. Manjunath and W. Y. Ma. Browsing large satellite and aerial photographs. In *ICIP*, pages 765–768, 1996.

[24] B. S. Manjunath, P. Salembier, and T. Sikora. *Introduction to MPEG-7: Multimedia Content Description Interface*. Wiley, 2002.

[25] K. Mikolajczyk and C. Schmid. Scale and affine invariant interest point detectors. *IJCV*, 60(1):63–86, 2004.

[26] K. Mikolajczyk and C. Schmid. A performance evaluation of local descriptors. *PAMI*, 27(10), 2005.

[27] S. Morain and S. L. Baros. *Raster Imagery in Geographic Information Systems*. ONWARD press, 1996.

[28] A. Natsev, R. Rastogi, and K. Shim. WALRUS: A Similarity Retrieval Algorithm for Image Databases. *TKDE*, 16:301–316, 2004.

[29] R. Pajarola and P. Widmayer. Spatial indexing into compressed raster images: How to answer range queries without decompression. In *IW-MMDBMS*, page 94, 1996.

[30] K. Pearson. On lines and planes of closest fit to systems of points in space. *Philosophical Magazine*, 2(6):559–572, 1901.

[31] E. G. M. Petrakis and C. Faloutsos. Similarity searching in medical image databases. *TKDE*, 9(3):435–447, 1997.

[32] J. Philbin, O. Chum, M. Isard, J. Sivic, and A. Zisserman. Object retrieval with large vocabularies and fast spatial matching. In *CVPR*, pages 1–8, 2007.

[33] P. Rigaux, M. Scholl, and A. Voisard. *Spatial Databases: With Application to GIS*. Morgan Kaufmann, 2001.

[34] S. Shekhar and S. Chawla. *Spatial Databases: A Tour*. Prentice Hall, 2003.

[35] M. Silva, G. Camara, R. Souza, D. Valeriano, and M. Escada. Mining patterns of change in remote sensing image databases. In *ICDM*, 2005.

[36] V. Singh, A. Bhattacharya, and A. K. Singh. Finding significant subregions in large image databases. In *arXiv:0906.3585v1*, 2009.

[37] L. Vinhas, R. C. M. de Souza, and G. Câmara. Image data handling in spatial databases. In *GeoInfo*, 2003.

[38] P. Vinten-Johansen, H. Brody, N. Paneth, S. Rachman, M. Rip, and D. Zuck. *Cholera, Chloroform, and the Science of Medicine: A Life of John Snow*. Oxford University Press, 2003.

[39] J. Z. Wang, J. Li, and G. Wiederhold. SIMPLIcity: Semantics-Sensitive Integrated Matching for Picture LIbraries. *PAMI*, pages 947–963, 2001.

[40] R. Weber and M. Milvoncic. Efficient Region-Based Image Retrieval. In *CIKM*, pages 69–76, 2003.

[41] D. Yankov, E. Keogh, L. Wei, X. Xi, and W. Hodges. Fast best-match shape searching in rotation-invariant metric spaces. *IEEE Transactions on Multimedia*, 10(2):230–239, 2008.

[42] J. Zhang, M. Gertz, and D. Aksoy. Spatio-temporal aggregates over raster image data. In *GIS*, pages 39–46, 2004.