Q-Database: Dimensionality Reduction and Efficient Retrieval of Medical Images from Large Databases

K. Arshad, M. Betke, G. Kollios
Computer Science Department
Boston University
Boston, MA 02215
{ kashan, betke, gkollios @cs.bu.edu }

Abstract

Q-Database (Q-DB) refers to a system which involves pre-processing medical images to extract features for efficient storage into a database. In this paper, we develop a system which uses feature reduction/selection (using Branch-and-Bound, Las Vegas Filter, FastMap, etc…). After this, the images are stored in a database that is efficient for fast retrievals. This method is compared with many similar techniques available to determine which is best in terms of efficiency (time to process, and accuracy). This system can easily be automated so that there is no human involvement until the very end to interpret the results. This system may help doctors to evaluate and treat similar symptoms of patients and prescribe better treatments which have more likelihood of success based on other doctors’ evaluation and experiences in similar situations. This may also help avoid intraobserver inconsistencies.
1. Introduction

Each year, radiologists take CT scans of millions of peoples. For each person, the number of CT slices scanned per visit could exceed 250. This may require considerable amount of disk space for storage. However, storing such data based on improper diagnosis may yield to further problems, as it may be incorrectly classified into a different dataset than in which it should belong. This happens because there are borderline or situation with multiple diseases in which the images could be stored multiple times or not stored in any proper category at all.

When storing images into a database, is it likely that raw data will take unreasonable amount of space, thus system resources and time needed in searching and getting results for doctors to use increase. If random parts of raw data are left out, or some similar instances of medical images are not indexed into the database, it is likely that the accuracy and as a result, the diagnosis of the doctor might suffer.

In the Q-DB System, there are steps which if taken, can make classification and querying of medical images fast and accurate. Having such a system is essential in hospitals as it will assist doctors in making diagnosis and prescribing ones that have more success in treatments similar to which have worked in the past with patients with similar troubles, and to avoid those treatments which may not be as successful.

Many methods have been devised to improve different aspect of the problem which include manual intervention(e.g. [1], ASSERT – physician-in-the-loop). Different studies tend to focus either on efficient storage, or efficient retrieval of image features. However, the two have not been studied together to see the effects of different variables.

As you can see that for this system to work best, the amount of data stored should be large. An ideal system would have a centralized repository, which physicians in the region (or anywhere in the world) can access and run queries to obtain results for patients with similar complaints.
In the rest of the paper, we give a brief survey of similar works which have explored different aspects of this problem. This is followed by a detailed outline of the system introduced by this paper. This system is then analyzed by experiments, and conclusions are drawn. Lastly, improvements and future work for different aspects of the system are suggested.

2. Related Work

There are various systems available to address the some of the issues stated in the introduction. These include methods to resolve the problem of storing data for retrieval, maintaining accuracy within a certain tolerance level, and process queries in reasonably short amount of time.

The ASSERT system [1], uses a physician-in-the-loop method which requires a person to manually delineate Pathology Bearing Regions. A Lobular Feature Set is also used to denote Lobe (s) in a lung. These regions are stored according to Opacities. Other features are obtained using texture, shape and grey levels. The texture features include contrast, homogeneity, and cluster tendencies. This yields a high dimensional feature vector which is indexed into a Multi-has index based on histograms of the different grey levels.

In [2] (Efficient Content Based Indexing (ECBI) of Large Image Databases), in order to avoid exhaustive searching, ECBI uses signatures (2SMLSF technique) which is an extension of multilevel signature file which is a forest of b-ary tree with every node, except leaf nodes in the structure having a b-child (p183). It uses multilevel signatures which could make addition of new elements difficult as a perfect hash function is required to be computed beforehand in the preprocessing step. In [3, 6, 7, 19] different ideas are presented on how index images based on what they contain, different information retrieval from image techniques, and how to index them for fast querying in large databases.
Presented in [9] is one of the segmentation algorithms available which uses shape-based curve growing method to recognize objects. The problem is modeled by Bayesian network. The model is the relationship between the curve shape and the image in order to maximize the energy of the points on the curve.

A System is presented in [20] which suggests using ARG (Attribute Relation Graphs), and indexing them using R*-trees. Our paper goes beyond that and explores dimensionality reduction, as well as different forms of indexing for high dimensional databases.

Other indexing techniques which are similar or related to our techniques are in [16, 17, and 22]. [16] uses indexing which has a hash table that requires multiple attributes for classification. This is similar to what Locality Sensitive Hashing (LSH) does. [17] is a similar application of LSH in Bioinformatics (it does not use Feature selection or Dimensionality reduction, but rather improves upon the idea of LSH).

In this paper we present a systematic use of similar approaches for fast and efficient indexing and retrieval of data in order for it to function better.

3. Details of the Q-Database Approach

There are many systems for indexing, segmenting, feature selection, and dimensionality reduction. However the Q-DB system is an efficient system which uses all of the above mentioned algorithms, to present a more efficient system.

3.1 Segmentation

There are many methods to obtain features from the image which may be done through segmentation. Depending on the level of details to gather the features, a doctor may be necessary to manually segment. Since in our paper we try to show how dimensionality
reduction and feature selection may effect accuracy/time for the query, we did not use feature which require a physician to be present.

Basic features such as the area, circularity, orientation, (nodule density & cluster ability) of Body, Left & Right Lung, Nodules, trachea, tumors if any could be obtained automatically. One point to note about orientation is that it depends on the way person is sitting. Since most of the people will be sitting is the same approximate position, it is best to use orientation of Lungs, Trachea, etc… with respect to the body.

The popular snake algorithm [8] could be used to segment lungs. However that requires some details of the image to be known before hand which may not be consistent in all CT scan. These detailed include the fact that some lungs tend to join at the bottom, and the centroid around which to place snake points may not be known to different sizes of the lungs at different level which could cause the program to fail for a considerable number of images..

We used algorithms based on [4], and [10]. These use boundary method to trace the outline of the lungs, and trachea. It takes into consideration that there may be nodules/tumor (which nodules growing at higher rates than normal) at the edge of the lung. Thus it takes into consideration the edge a few steps ahead and sketch the outline accordingly.
This image is then shaded using region growing method to color different regions in the image and obtain features such as area, circularity, orientation with respect to body, nodule density, …

In order to color the image, first the image is scanned linearly. Then when it saw red, it will color red all the subsequent pixels and stop coloring when it sees an already painted red. The edges become very hard to paint this way. Another method explored was to select a point in the image, then go in vertical and horizontal direction from that points. If it encounters the same color in all four directions, that area would be colored that way.

The above mentioned ideas were not as efficient as the final one. In the last idea, the image was scanned for yellow, and non white pixel, which was the left lung. Same technique was applied for right lung, and trachea. Finally, all these regions were color
with the appropriate color using the region growing method. The basic region growing algorithm was modified so that it colors the region inside the lung, the nodules as pink or blue depending on which lung they were in, and the condition for the recursion of the region growing method to stop was it encountered its own color again.

The above region growing method is basically a hybrid between region expansion and thresholding.

Once those main regions were identified, the remaining image was either the body or nothing. In the original image this meant that either it was grey representing nothingness which was colored black, or the remainder white which represented the body. This was also completed using region growing method.
When the images are too noisy, or have too many abnormalities for the program to run, those images were considered as improper outliers, and were removed. These occasionally happened near the top/lower part of the body.

![Figure 3](image)

Figure 3

Another issue we had was that at different parts of the lungs, the lungs appeared to merge into one. This way only left or only right lung is detected by the algorithm described above. An example is in figure 4.
In order to resolve this issue, the first idea was to draw horizontal lines, offset by the orientation of the lungs, in the top-middle region of the image. Then measure vertical distance of lines which pass through the lung. The one with the minimum vertical distance is the place where “cut” should be made.

Another way is to solve this is to assume the boundary drawn as a sequence of points. Then comparing the points which are reasonably far in sequence yet close in their Euclidean distance, this is where the lungs should be separated for the program.
The way we resolved it was using projection. Assume there is a box in which the top half of the image was present. If you drop the image from 25% to 75% of the width, this area is likely to contain the area where the lungs are joined. With vertical projections, the place is found where there are fewest points, and this is where the cur is placed.

Figure 5

After this features mentioned above (area, circularity, orientation, of left lung, right lung, body, distances between each object, etc… are obtain)
3.2 Dimensionality Reduction & Feature Selection

At this point, for each image we have a lot of dimensions. There are 28 in our case, but in the field, when more features are extracted, it can easily reach a 100. With higher dimension, the time required to run a search query rises exponentially. If a linear scan is used, thus yielding results which may not feasibly run within the necessary time (usually, a few ms).

3.2.1 PCA (Principle Component Analysis)

PCA (or Karhunen-Loeve transformation) rotates the data about the axis so that the maximum number of variabilities are projected onto lower dimensions.

First, the mean of each dimension is subtracted with the each element in the corresponding dimension. Then, the covariance matrix is computed. For n dimensions, this covariance matrix will be n x n.

From this covariance matrix, the eigenvalues and eigenvectors are computed. The n components with the highest eigenvalues (which are sorted in order) are used to calculate n dimensional points. They are sorted because this way it would be a feature forming vector. This method can take considerable amount of time for the initial data set (the bulk of data being reduced for the first time).

This way PCA can effectively reduce the number of dimensions while retaining important information based on the pattern

So, when a person has to query, the covariance matrix is available and can be used to preprocess the query to reduce the number of dimensions based on this covariance matrix. Occasionally, after a lot new data points are added, a new covariance matrix may be necessary.
3.2.2 FastMap [5]

FastMap reduces the number of dimensions faster than PCA, since it does not have to compute the entire covariance matrix. FastMap is a metric analogue to the KL-transformation (PCA).

It works by first selecting two pivot points which are furthest apart. Then it computes a pseudo-projection of the remaining points along the “line” (or multi-dimensional plane) which was created in the first step. These points are projected into the orthogonal subspace recursively.

The pivot points should lie along the principle axes, thus must be far apart. For the selected pivot points in the first step, the remaining points are projected using the law of cosine to determine the relationship, and thus pseudo-projection – which is the first coordinate. Then recursively, all the other dimensions are obtained until the required number of dimensions is achieved. In each recursion, the distances within the orthogonal hyperplane are calculated using Pythagorean theorem.

The complexity of this algorithm is O(N) as compared to O{N^2} of Multi-dimensional scaling.

3.2.3 Feature Selection

A survey of Feature Selection algorithms can be found in [18]. Here I just give a brief outline of the processes involved.

Branch and Bound Algorithm (B+B) is based on an evaluation function. Based on this, and a given threshold value, the algorithm halts at the tree levels where the evaluation is lower than that of the threshold. The way this program works, it can grow exponentially.
The FOCUS algorithm computes each singleton feature set, (and then $2^N$, where $N$ is the number of iterations). It half when a suitable and consistent solution is found.

Las Vegas Filter (LVF) creates random feature subset, and calculates the evaluation measure. Based on this it determines all the solutions found.

Las Vegas Increment (LVI) tries to find a good solution using sampling rather than the entire data set. The suggested sample size is 10%, but this really depends on the user requirements.

Las Vegas Wrapper (LVW) uses LVF to create candidate subsets. Then the accuracy of the inducer function is used as an evaluation measure.

The Automatic Branch and Bound (ABB) algorithm is exponentially slow, and can take days to run on data sets of a few thousand elements. Quick Branch and Bound (QBB), a variant uses LVF to find suitable starting points for ABB this way, the time is cut down drastically. It is reported to be better than LVF, FOCUS and ABB.

Sequential Forward Generation (SFG) iteratively adds feature to an initial subset to improve a known measure. It takes into consideration the feature which are already present in that measure. It can sort the list to give the most useful measure in order.

3.3 Storage into Database

Once the dimensionality reduction algorithms are applied, we test the database the resulting data sets. The resulting datasets involve dimensionality reduction to different extents (going down from 28 to 2-12 dimensions), and databases of different sizes (1000-4500 elements). Then different indexing techniques are applied (Linear, R*-tree, and LSH) in order to test storage, and querying speeds, and decide how efficient each one is for such a system.
For each image, the key used was Patient ID, CT Scan Date, and CT slice number to uniquely identify any single scan. Since dimensions were reduced, the k-nearest neighbor points retrieved are also in reduced dimensions. Along with the Key, the data stored with the key are the actual coordinates in the original dimensions. Instead of this, a better technique which could save on storage (for PCA, FastMap) is to store the matrix used to reduce the dimensions. With the actual values stored as data long with the points, the actual values such as nodule density, lung area, etc… can be retrieved. This way information is not lost. The databases rely on Euclidean measure of distances.

There are many indexing which have been investigated to some extent. We have used, and explain the reasons for doing so. We try to present efficiencies, accuracy, and speed of various different types of storage techniques.

3.3.1 Linear Method

Linear method is probably the simplest form of method used for storage/querying. It gives the k-nearest neighbor (KNN) with perfect accuracy, and the results of this method are used to measure the accuracy of other indexing techniques. Tests were also done with Normalized version, since some data points are less than zero, while others are in thousands.

3.3.2 R*-Tree

The data points for each different feature are likely to be clustered together. The trachea is likely to vary within a small range, and so is the orientation. For this reason R-Tree is a natural choice as it uses clusters using minimum bounding rectangle around different database, and based on this storage structure can fast in querying as it does not have to go through all data points in the structure.

3.3.3 Hash-based Method: LSH (Locality Sensitive Hashing)
LSH is a popular method used because it is faster than most methods for high dimensional indexing. We used techniques presented in [11, 21, and 22]. It works by creating random partitions in a high-dimensional space into high dimensional data cubes (hash bucket). The points in the same buckets are the nearest neighbors. For a given query it determines in which bucket that query is located, and then returns the points in that buckets as the nearest neighbors. Since it hashes a point multiple times (using different hash functions), points close together at the edge of the cubes will be returned as the nearest neighbors.

The general overview of the system is given below (figure 6)

![Figure 6](image)

The overview on how to query is given in figure 7

![Figure 7](image)
The first step in the query may not be applicable at time. The system works best for running nearest neighbor queries of images. However, query may be run without the CT scan with those aspects of the image which might not have been selected by feature selection, in that case, a linear search or secondary indexing may be necessary as is the case in many systems.

4. Experimental Results & Analysis

Below we will present the tests which were done on data collected. We present the data in the manner which is best to read and understand to evaluate and analyze rather than an exhaustive list of everything.

This is followed by analysis about the results, and why they might be the way they are as well as their accuracy.

The conclusion and discussion of results are presented in Section 5.

Below we present the results we got. It should be noted that the presentation is followed for best demonstrative evidence, and conclusions drawn thereafter. In order to keep things clear, an exhaustive list of graphs was avoided as it was unnecessary.

4.1 Storage

Here I compare the time it would take to insert points into the r-tree. Rather than exploring time taken to add 1 point, a more reliable figures can be obtained for comparison with different methods.
Figure 8.1: Storage time vrs Dimensions (1000 points) for Linear scan method.

Figure 8.2: Storage time vrs Dimensions (4050 points) for Linear scan method.
Figures 8 & 9 show comparison between R*-tree and Linear based method for adding 4050 points into the structure. Linear scan as expected takes the shortest time.

The linear scan, to add currently just adds the element to the end of the array, and for k nearest neighbors would require k scans of N elements in the list. Thus it would traverse through k*N elements to get k nearest neighbors.

The time shown in Figure 8 is probably the base time, and all other methods would take longer to add elements to their structure for reason explained above. R*-tree takes comparatively a lot more time. This happens because after the element is read into the R*-tree structure, a lot of adjustments to different parameters need to be done so it remains optimized to run fast query. Basically the boundaries of hypercubes have to be readjusted, and points in different hypercubes need to be rescanned and allocated if they
are affected by this adjustment. This causes a lot of operations, thus increase in time requirements.

For LSH, time factor depends on the radius of the expected search query. For small radius, calculation of variables takes longer because collision into different buckets need to be taken into consideration. For higher radius means this can be more liberal this having more collisions into the same bucket, and the new hash functions that need to be calculated are less. Time takes longer for more dimensions, and points, but with higher radius, the time is decreased comparative to the same situation with smaller radius for search.

There is not much difference between times for Linear, and Linear with normalization, except, as expected, since linear with normalization has to do more operations and takes slightly longer.

R*-tree with 1000 points does not reduce the time drastically, but rather, just proportionally.

4.2 Querying

Here we compare the time it takes to query vrs the Number of Nearest Neighbor requested for different dimensions, database size & methods.

The convention followed for querying is that Best, Average and Worst represent the location of query in the stored database. Best is from the start of database, average is from middle, and finally, worst is from the end of database. The location of query in the database, of course, does not affect the speed of retrieval, and are almost the same.

The second aspect to note is that feasible time is considered to be the minimum time the query can take using different methods. Feasible time is almost generally considered to be small and measured in milliseconds.
Here is what the graph looks like for increasing number of nearest neighbors. The angle with the x axes is very small.

![Figure 10: Linear Query with 1000 points and 3 Dimensions & 9 Dimensions.](image)

For Linear scan, there is a small increase in time the required with increasing dimensionality. However, the increase is proportional. This increase is represented by the constant multiple (the slope) which is related to the number of extra calculations needed.

![Figure 11: Linear Query with 4050 points and 3 Dimensions & 9 Dimensions.](image)

From Figure 10 and 11 it can be seen that with the increase in the number of nearest neighbor, there is a slight but significant increase in time required. But as the number of points increase, this increase is likely to effect the operations even more. So if there are a
50 million points in the dimensions, the slope is likely to be more steep (approaching 1). This will render linear method ineffective.

In Figure 12, you can see that linear with normalization does not have much difference as compared to linear. This is because, the query have to be changed by the normalization vector which is one dimensional, thus takes same number of operations as linear method plus the number of operations need to normalize the query which depends on dimensions.

![Figure 12: Linear with Normalization Query with 4050 points and 3 Dimensions & 9 Dimensions.](image)

Comparatively, R-tree behaves in a similar manner to the Linear method discussed above to some extent. With increasing dimensions, there is slight increase in the time necessary, but the change is not as significant as it was in the Linear method. This R-tree would be a good choice for high dimensionality.
But, as in Figure 14, with increasing number of points, the time required increases proportionally with the number of points it has to store in the database. This probably happened because of the way R*-tree stores points in the structure, with more points more nodes need to be created. This will cause more points need to be read through by the program before it arrives at the result. But still R*-tree is still a better choice than Linear method because the effect of increase in points and dimensions is not as high proportionally to the Linear method. So if there are millions of points, it might be 3-4 times faster than Linear, but still, the time necessary would not be feasible.

For LSH, because the time range with different dimensions and points is comparatively less as compared to the above two methods (Linear and R*-tree), only extremes (dimensions and points) were tested.
LSH for 1000 points and 3 dimensions ranges from 0-14ms, R*-tree ranges from 0-13ms, and Linear methods take 0-150ms. For small number of dimensions and points, R*-tree will do just as well as LSH. But if time is compared with figure 16, with tripling of dimensions, and quadrupling of the number of points, LSH double (0-35 ms), while Linear method ranges (0-310ms), and R*-tree ranges from (10-100ms).

Here is a sample table to plot LSH graph for 3-dimensions and 1000 points. R is the radius used which retrieved nn number of nearest neighbors.

<table>
<thead>
<tr>
<th>t</th>
<th>nn (best)</th>
<th>t</th>
<th>nn(avg)</th>
<th>t</th>
<th>nn(worst)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.72</td>
<td>1 (r=10)</td>
<td>0.72</td>
<td>1 (r=10)</td>
<td>0.72</td>
<td>1 (r=10)</td>
</tr>
<tr>
<td>1.98</td>
<td>10 (r=1500)</td>
<td>3.3</td>
<td>10 (r=6000)</td>
<td>2.64</td>
<td>10 (r=4500)</td>
</tr>
<tr>
<td>2.34</td>
<td>20 (r=2600)</td>
<td>4.14</td>
<td>20 (r=7000)</td>
<td>2.56</td>
<td>20 (r=8500)</td>
</tr>
<tr>
<td>5.4</td>
<td>100 (r=15300)</td>
<td>6.6</td>
<td>100 (r=17000)</td>
<td>6.6</td>
<td>100 (r=15500)</td>
</tr>
<tr>
<td>7.86</td>
<td>200 (r=20000)</td>
<td>9.84</td>
<td>200 (r=24000)</td>
<td>8.4</td>
<td>200 (r=23000)</td>
</tr>
<tr>
<td>11.28</td>
<td>400 (r=45000)</td>
<td>12.06</td>
<td>400 (r=28500)</td>
<td>11.1</td>
<td>400 (r=28500)</td>
</tr>
<tr>
<td>14.58</td>
<td>500 (r=51000)</td>
<td>13.32</td>
<td>500 (r=30500)</td>
<td>10.8</td>
<td>500 (r=30800)</td>
</tr>
</tbody>
</table>
So, LSH is probably the best choice because its speed is likely to take least amount of time in the field. However, accuracy measure is just as vital as indexing. If a method indexing and retrieves fast, but lags in accuracy as compared to other methods, it will not be as useful. This discussion is done in section 4.3

It must be noted that when future data is added, there will be more results for the same radius in LSH

4.3 Accuracy

Accuracy is basically a subjective measure. If you look at one image and look at another image, the images may not necessarily correspond. This happens because when the nearest neighbor runs, it looks for the nearest neighbor respective all the dimensions (features) so if one image has very small lungs, and the other medium sized lungs, it
could be that the nodule density for the lungs was almost identical, thus causing the nodules to be returned.

In order to resolve this issue, a measure of dimensionality bias can be added, or features can be ignored which may be not useful or searched upon very frequently. This way a certain degree of visual uniformity can be obtained. Another way to handle is to further filter out the result based on different factors. So, after getting the result, a filter can be added to remove those subsets in which the lung areas are very different.

In the linear method, when some feature are more prominent (larger, or similar angle, …), it will probably return based on those features because the area is large, but orientation has a small range. So it is likely that nearby lungs with similar orientation may be returned.

In linear with normalization method, this factor is done away with, and that natural dimensionality bias is removed. But this causes all sort of random results to be obtained, because the dimensions are then, not controlled properly. The exact match queries will work perfectly in both methods, but will not work best when using nearest neighbor queries.

All the examples we give below are colored so the features are more prominent.

So with this image as the input query:
Although this situation can at times be fixed with filtration, dimensionality bias (controlled or naturally occurring within the dataset is best. With this in mind, the remainder of the discussion in mind is done in context of reduced dimensionality and their effect (useful or not).

**4.3.1 Feature Selection**

When feature selection algorithms are used, even though they select the best features, this causes their own set of features being used which may or may not includes the measure which a person might want.

In the field these algorithms would be run on sample database to identify which features to use, the remaining database would be built upon that. But it is essential to choose the most effect system from the start as modifying the database later could be costly.


FOCUS chose #1, 3
LVF
   with factor 0.05: #2, 4, 5, 9, 10, 14, 16, 19, 22
   with factor 0.95: #5, 6, 8, 10, 11, 14, 16, 18, 20, 22, 23, 24, 25, 26
LVI: (threshold: 5) #3, 5, 11, 17, 18, 20, 25
LVW: (threshold: 100) 4, 5, 7, 8, 14, 18, 20, 21, 22, 24, 25
QBB: #6, 8
SFG gives the rank and gain. Here are the features in sorted order with respect to gain,
except when gain is 0, the features is ignored: #3, 9, 13, 1, 5, 23, 22, 20, 17, 19, 18, 16,
14, 7, 21, 10, 4, 24.

From the features itself, it can be seen that some of these might not be as useful because
they do not take vital feature (e.g. LL Area) into consideration.

From the tests run on all these databases, it was seen that SFG is best. Results are similar
for upto four dimensions (LL area, LN area, RN area and Body area). This probably
happens because these values are large, thus dimensionality bias, which, fixing normally
would cause results to be unusable).

4.3.2 PCA & FastMap

Running test results, we can see that PCA can be reduced down-to two dimensions, an
FM to three to get the same results as SFG with 4 dimensions, or the original data set
with 26 dimensions.
Reducing dimensions Using these techniques yield similar results. However, if there are millions of points within the same range, this will increase the number of points nearby thus reducing the similarity of results when dimensions are reduced using this technique. It would generally be best to determine the maximum number of dimensions to be used based on system specifications, and speed required for retrieval of queries. Having more dimensions will reduce this problem, but using all dimensions may be unnecessary.

4.4 Conclusions

In terms of indexing and querying, the recent LSH algorithm is the best for use in the field as it eliminates “the curse of dimensionality” and gives results better than most of the algorithms available. Since it overcomes, and effectively return the nearest neighbor, if features are modified before indexing for dimensionality bias to some extent.

For feature selection algorithms, it would be best to use the feature identified by SFG, however, the system designer should be aware that feature selection algorithms may not necessarily work best with their database, thus should add features which may be important, but not chosen by the feature selector. This means that features have to be selected carefully.

However, with the comparison done above, dimensionality reduction is a much better choice than feature selection. Both reduction (PCA & FM) give similar results, but FM
(and SFG), comparatively run faster. Thus using FM & LSH, this will likely yield a powerful combination for the database to be used in the field in terms to accuracy, and time required for process the query and retrieving results.

When the run of SFG alone are compared to those of FM & LSH, the results are comparable, thus either of these three methods depending on how much dimensionality reduction is necessary.

A further improvement of the system is to use SFG or manual selection of features (if necessary), and then apply the dimensionality reduction algorithm which works well in out case, where we reduce 26 dimensions down to 3.

5. Future Work

Currently we segment based on 2-D scan, 3-D scan would be better with new feature such as volume, centroid in 3-D space, and 3-D circularity. This would change the perspective of the image, and could be make to work with 2-D image queries. Using 3-D features will reduce the storage space and the number of points because it will store each patients scan as one points, however, it will require more features, (edge, location details,… ) which would be interesting to observe on how this effects accuracy.

Environmental/social factors pay come into play for certain diseases. The database could be extended to contain “anonymous” history of patients which could be queries upon. These could include: weather the person uses drugs, pollen season, disabilities, drinking habits, etc…

New techniques for indexing and searching are presented in [13, 14, 15]. The method includes biased sampling, selectivity estimators, and on-line discovery which are methods very different from those investigated above, and thus worthwhile to investigate in a similar context.
The BoostMap [12] technique could be applied to the situation here. Since it is relatively a different technique, thus outside the scope of this paper, it will yield interesting results.

Data Mining could be used to find out interesting properties not known before should be interesting and a useful factor in tracing the source of common complaints.

7. Acknowledgments
We thank Jing Bin Wang for providing Lung segmentation code which uses boundary following method to identify the lungs and trachea. We also thank Jonnie Alon for providing FastMap implementation. And finally, thanks to Vasillis Athitsos & Jingbin Wang for helping me in getting to know the lab/system resources better. Thanks to Bill for suggesting me to use thresholding for nodule detection. The initial version of R-tree was obtained from Marios Hadjieleftheriou’s website. The Locality Sensitive Hashing code was written by Piotr Indyk.

All programs except for Locality Sensitive Hashing were written in Java. Locality Sensitive Hashing was written in C++. The performance shown by C++ and Java are comparable¹, and various independent forums. Even if a fixed factor is used to scale down the results for programs written in java, the above analysis still holds true.

System Resources
We used a PC with two Intel Xeon 2.4GHz processors, and 2GB of physical memory.

References


[6] Avi Kak, Christina Pavlopoulou, School of Electrical Engineering and Computer Engineering, Purdue University, Computer Vision Techniques for Content-Based Image Retrieval from Large Medical Databases


[16] L. Grewe and A.C. Kak, Purdue University, interactive Learning of a Multiple-Attribute Hash Table Classifier for Fast Object Recognition


[19] Efficient Content Based Indexing of Large Image Databases. ESSAM A. EL-KWAE University of North Carolina at Charlotte and MANSUR R. KABUKA, TOIS, Vol 18 Number 2, April 2000, p171-210
