A Search Engine for Images

Yujing Wen
Department of Computer Science
University of Maine
Advisors
Curtis Meadow
George Markowsky

Project Report

(4/28/2003)

Contents

Abstract

A search engine was designed and implemented to allow campus faculty and staff to access the Campus Collection of the University Of Maine Museum Of Art. The Campus Collection refers to the subset of the Museum art collection that is available for display on the University of Maine Campus. University faculty and staff can request items from the Campus Collection. With the search engine, users can easily and quickly search for and request desired items. In this paper, the necessity of designing an efficient search engine and the significance of this project will be addressed in considering the given drawbacks presented in the existing management system for the available art works. Then an overview of current image database search methods is presented, followed by explanation of the method that was selected for construction of the search engine and the reason for choosing the method. After this, the structure of the search engine and its implementation are described in detail. Finally, the work is summarized and limitations and future work are discussed.

Keywords: Search engine, Relevancy Ranking, Vector Space Model, Image database.

1. Motivation

1.1 Current Situation

Currently, there is a collection of about 3880 pieces of artwork in the University of Maine of Museum of Art. Information about each piece of artwork is stored in a relational database that has been designed for Museum staff access only. Normally, if users want to view or borrow an artwork, they have to go to the Museum and ask a curator to find it. Curators normally work through the database to find the reference number and storage location for the item.

1.2 Problems

Obviously, there are several drawbacks in the current artwork system.

- Time and Place Constraints

  In the current artwork management system, the artworks, having to be exposed to users directly, are stored in museum storage rooms, which means that users need to go to the museum to view these artworks during museum hours. As far as the user is concerned, this is inconvenient. From the museum point of view, this involves more requirements concerning the management of those artworks and the services provided to users.

- Some Difficult or Impossible Physical Access

  It will be difficult or impossible to access some artworks that maybe spread across a large physical area or composed of material that may require delicate handling.

- Safety Problem

  Some artworks are very valuable and precious. Exposing of these artworks directly to users probably will result in wear, contamination and even damage to artworks. Therefore, it is not safe to expose some real artworks to the public.

- Lack of Information

  Users have no way to view the campus collection as a whole and have little or no idea about what is contained in the collection.

1.3 Our Solution

The highly developed computer technology allows images of artworks to be digitized and stored in a database structure on a computer. We propose to design and implement a search engine, a bridge between a user and the artworks. Search engines were originally described as automated programs that compiled and updated databases without human intervention. At present, the definition has been expanded to be a group of tools that have been developed to index and retrieve information from a database [1].

Using a search engine, the problems incurred in the previous artwork system can be resolved in the following way:

- The search engine makes the artwork database accessible online. So users can visit the Museum any time of the day or night from the comfort of their home workstation.
- With the increase in the storage capacities, it is possible to digitize images and multimedia on a massive scale. This will eliminate the physical space limitation. Users can access the whole images in the database within seconds.
- Since the interfaces of search engine are very simple and user friendly, users do not have to be trained in order to use it.
- By using the very efficient and advanced compression techniques, we can digitize a very large image effectively. Therefore, a large piece of artwork can be easily presented to users. By providing a reasonable digitized facsimile to the original artworks, we avoid directly handling some delicate artworks.
- The search engine system isolates users from the real artworks, so we avoid casual exposure of precious artworks to public. Since these works are intended to be loaned to campus members, they are all actually susceptible to "exposure." Additionally, since the search engine is a client/server application and we put the original database on the server side, users on the client side will not see the original database. The users' right to access the data can be under control. Therefore, users can only browse the artworks but cannot delete or update the original data.
- Since original objects are digitized, users can interact with the surrogate. The ability to zoom in and out and compare close-ups will allow a user to make his/her own juxtapositions at various levels.

From the above discussion we can see that a search engine can successfully solve the problems present in the old system. In addition to this, the significant advantage of using a search engine is that artworks can be widely accessed; digitization of photographic images and web accessibility will inevitably result in more people viewing the surrogate images. These are the motivations for developing a search engine.

My graduate project is to design and implement a search engine (software application package), which will provide several web interfaces to the Museum of Art Campus Collection that will allow campus users to browse or search for items from the collection and to request placement of desired items

The Search engine consists of three parts, a Front-end, a Back-end and a set of searching algorithms. The Front-end provides interactive interfaces to users. The Back-end is the database that stores the original data. At the heart of the search engine are the searching algorithms.

There are various searching methods for image databases. In order to find the best searching method that meets our requirement, some research was conducted on image database searching in

three successive steps. For the first step, the research focused on the methods for searching image databases. On this level, text-based search was selected. For the second step, the research was centered on the text-based search algorithms. The Vector Space Model was sorted out of several searching models for text-based search. The final research was on implementation of Vector Space Model, in which an efficient method was selected.

This paper is organized as follows. In section 2, the background of image searching methods is reviewed. In section 3, the procedure for choosing our search method is discussed in detail. The implementation of the search engine is described in section 4. Finally, a summary is provided and limitations and future works are pointed out in section 5 and 6. The source code is appended in section 7.

2. Background (Overview)

In order to search an image database, we must make sure that the images are indexed in a manner that facilitates later retrieval. Most techniques for indexing images fall into two broad categories: indexing by annotation (text) and indexing by content, to which the corresponding searching methods are text-based and content-based search, respectively.

2.1 Text-based Search

The basic idea of text-based search is to associate keywords or text with each image, called indexing by annotation. Both query and images in search are represented by index terms. Queries are resolved by searching through the annotations. Users are required to enter a keyword or textual description of a desired image. Standard text retrieval techniques are then used to identify the relevant images [2].

Advantages:

- High-level abstraction and concepts can be easily expressed
- Query can be easily issued. The user interface is very simple.
- Standard text retrieval techniques can be used for searching. Text retrieval techniques have been highly developed in information retrieval system.

Disadvantages:

- It is tedious and time-consuming to associate keywords or text with each image.
- Some features are very difficult to describe with text, some special textures and complex shapes can't be clearly represented.
- Text descriptions are sometimes incomplete. It is possible that some image features may not be mentioned in a textural description.
- Text descriptions are sometimes subjective. Different indexes or even the same indexer may describe the same feature with different terms or different features with the same term [2]. It is up to the curator to select the terms for describing the object. The underlying reason for this phenomenon lies in the fact that most terms have multiple meanings on the one hand, and on the other hand, some concepts can be described by more than one term. This is called synonymy and polysemy problems.

The most critical problem is the last one in the above list of disadvantages. To address this problem, people try to limit the words for describing images. Structured vocabularies were developed. Some of these vocabularies include the *Library of Congress Subject Headings*, *ICONCLASS*, the *Thesaurus of Geographic Names*, the *Art & Architecture Thesaurus*, and the *Union List of Artist Names* [4].

Many methods are also developed to solve this problem. Some of these approaches are proposed based on dimensionality reduction and some on the feedback information from users with the latter known as Relevance feedback. The dimensionality reduction method is characterized by reduction of a high dimensional vector space to a low dimensional "semantic space" to capture concepts contained in an image collection. The latent semantic indexing (LSI) method, which is one of the widely used methods, reduces a high dimension vector space using singular value decomposition (SVD). As far as the relevance feedback is concerned, after a retrieval set is obtained, feedback information from a user is collected and used to refine the original query. Therefore, the retrieval performance can be significantly improved.

The overall inevitable problems involved with the above improved methods are:

- The computation expense increases greatly.
- The whole system becomes more and more complex.
- There are no very concrete conclusions about them and it is not clear yet how to extract the maximum benefits from them. A lot of on-going research is concentrating on this issue.
- Relevance feedback only works after a period of time in systems that accept large numbers of queries. It is not suitable for small systems.

2.2 Content-based Search (indexing by content)

Recently, research was conducted on content-based search due to the inherent problems with text-based search. Content-based indexing technique extracts information from an image by using artificial intelligence and pattern recognition. It compares image features directly.

A number of methods were developed to depict an image. One of these uses a data structure called 2D string, which attracts people's interests. In this structure, objects in an image and their associated position information are listed. Both documents and query are expressed in 2D string. Once 2D string query is issued, finding similar image becomes finding matching 2D substring.

Advantages:

- The enough information contained in 2D strings makes it easier to reconstruct a symbolic representation of the image.
- Because the information is only symbolic, the representation is much more compact.

Disadvantages:

- 2D strings can not be generated automatically, because it is difficult to identify objects represented in the 2D strings without human intervention. [13]
- Since objects are represented by points in 2D strings, some special relationships, such as DISJOINT, JOIN, PART-OVLP, CONTAIN and BELONG between two objects, may not be determined correctly.

Jacob proposed a process called "wavelet decomposition," which is used to generate signatures for both images in a database and a query image. The task for finding similar image to query becomes finding similar image signature to query signature.

Huang and Jean's [18] technique indexes images based on their "morphological skeletons." The image skeletons are extracted in two steps. First, the images are divided into subsections based on the features. Then the "bones" are represented with center points and radiuses. After users

issue queries by drawing a sample image, a “query skeleton” is derived using the same method. This query skeleton is then compared to the pre-stored skeletons of images in the database and matches are returned.

Other image indexing techniques extract feature information at various coordinates. The four most common features are color, shape, texture and edge [22]. The technique originates from the early experiments conducted by Kato [20], which is on the automatic retrieval of images by color and shape feature [21]. First, the features of each image are extracted using various pattern recognition techniques and saved in an index file. Later, Queries are expressed through visual examples, query by visual example (QVE), which can be either sample images from the database or user hand-drawn examples. In the former case, the features of the query image have already been computed; in the latter case, the features can be quickly derived because the hand-drawn image is not likely to be very complex. The features of the query image can then be compared to the precomputed features stored in the index file, and if similarities are found, the relevant images will be presented to the user.

Gray [2] proposed a simple content-based system that uses the color and edge extraction modules to construct a set of histograms and an edge map for each image, and applies histogram intersection to compare color distributions and sketch comparison to compare edge characteristics. After the system was evaluated with a real-world image database, a number of weaknesses come out.

For the color retrieval mechanism:

- Users can’t specify that the absence of a color from a certain region means that the image is irrelevant.
- The position information is not taken into account in the algorithm.
- Since exact color match is performed by the histogram intersection algorithm, score zero will be returned for images that do not contain the exact same shades of color.
- The efficiency of the histogram intersection is not good; the response time will be poorer for large databases.

For the edge retrieval:

- Many edge maps contain extraneous edges.
- The sketch comparison algorithm cannot calculate the similarity score very correctly.

Tang [13] pointed out that query-by-example techniques are often "not robust to different scales, rotations, and small changes in the location of objects".

Venters and Cooper [22] undertook a review of currently available content-based image retrieval (CBIR) systems. The report summaries the functionality of five commercial CBIR applications: Excalibur Visual RetrievalWare SDK, ImageFinder, IMatch, QBIC DK, and the Virage VIR Image Engine SDK and highlights the functionality for the increasing number of prototype research systems.

It points out the following problems

- Research to date is contradictory and the validity of the interaction methods continue to remain untested with real user populations.

- The analysis of the feature characteristics has the potential to be objective.
- Limited success was achieved in developing robust retrieval algorithms.
- The difficulties involved in developing effective and robust image retrieval systems are problematic e.g. image encoding, storage, compression, transmission, display, shape description and matching [22].

Summary of the content-based search:
Advantages:

- Can capture low-level image features.
- Accepts pictorial queries.

Disadvantages:

- Cannot capture high-level concepts.
- Bad querying performance, due to complex interfaces and pictorial query process is hard to start.
- Quite intense computation and very complex system.
  Whether indexing images by 2D string, or signatures, or feature vectors or according to their "morphological skeletons," they all need very expensive preprocessing and precomputation and make the whole system very complex.
- The technology still lacks maturity, and is not yet being used on a significant scale.

3. Our Selection

People always try to pursue a "perfect" method for both representation and retrieval algorithms, whereby only and all relevant documents are retrieved. However, from a practical point of view such methods would be very complex. Therefore, IR researchers suggest that the system should adopt fairly simple methods of representation and seek approaches that facilitate the ranking of documents [23].

Based on the above points and our present situation, the selection criteria were made as follows.

- Simple image and query representation.
- Less computation cost (no precomputation or preprocessing)
- Standard searching techniques are applicable.
- Required functions can be easily implemented
- System is as simple as possible
- The relevancy ranking of documents can be facilitated

To make the decision on the selection of search method, the two search methods are compared based on the above selection criteria.

- Text representation is much simpler than content representation. Whether we represent images by 2D string, or signatures or feature vectors or some "morphological skeletons", we need first to do very expensive preprocessing and precomputation to generate them; while it is very easy to generate the text representation without any preprocessing or precomputation.
- The computational complexity of content-based searches is much higher than that of text-based searches, since content-based searches need very expensive preprocessing and precomputation to represent images.
- The content-based searching technology is still very young. Some content indexing methods are not accurate, effective and robust, and it is very difficult to develop effective and robust content-based image retrieval systems; while the text searching techniques is very mature in information retrieval system. There are standard searching techniques for text-based search.

- Content-based search systems are much more complex than text-based search systems. To issue a query, users need to draw a picture, which requires a painting application to be added to the system; while text-based search does not need any extra application.
- It is very difficult to issue a high-level abstract concept query in the content-based search. However, it is very easy to do that in text-based search.

From the above comparison, it is clear that a text-based search meets our criteria better than a content-based search. In addition to this, the first drawback, which is associating text with images is tedious and time-consuming, is not a issue, because the keywords and text description have already been associated with each image by the Museum of Art, and the synonymy and polysemy problems are not very critical to us, since our database is not very large and it is a domain specific (artworks) database. Because of these reasons, we selected text-based search instead of content-based search.

Some research on text-based search was conducted. In an information (text) retrieval system, retrieval is based on some model. Various models have been proposed. Every other model ultimately relies on three basic models: vector space, probabilistic and Boolean [24].

The Boolean model compares Boolean query statements with the terms that represent a document. The drawbacks are:
1) User needs to figure out the correct logic and phrasing to get the correct answer, thus it is not convenient compared with natural language query. For example, if you want to find a lovely dog picture in Maine, you could enter the following natural language query: *lovely dog picture in Maine.*
A comparable query using Boolean logical syntax would be entered as: *lovely and dog and picture and Maine.*
2) It uses value 0 and 1 to represent the similarity between the query and the returned documents. Therefore, it is difficult to do relevance ranking on documents [24].

The probabilistic model computes the probabilities of relevance of a collection of documents. Although documents can be ranked, the ranking method does not improve the retrieval effectiveness. [25].

The vector-space model represents both the user's queries and the documents by a set of controlled terms (such as, the index terms). Query can be posed using nature language; it can be described in the same terms users would use in speaking naturally. The similarity degree can be measured by computing the score number for each document. The higher the score is, the more relevant the document is to the query, thus it is easy to realize relevance ranking search.

We know that the most powerful weapon in search engines is relevance ranking. Simply put, relevancy ranking arranges a set of retrieved records so that those most likely to be relevant to the request are shown first. Moving down the ranking list means toward records with less likelihood of relevancy. With a good relevancy ranking algorithm, users will spend less time reviewing search results before deciding whether they are satisfactory. Users do not care about how many records are retrieved, as long as they know that the best information floats to the top. In addition to this, users do not need to compose complex logic queries to try to reduce the number of retrieved items.

Because Vector Space Model can support the powerful relevance ranking better than the other two models, we selected the vector space model for the text search.

3.3 Vector Space Model (VSM)

In the VSM, each document $\vec{d}_j$ $(1 \leq j \leq n)$ is represented by a weighted vector,

$$\vec{d}_j = (w_{1j}, \ldots, w_{tj})^T.$$

where w_{ij} is the weight (or importance) of term i in representation of the document $\vec{d}_j$, t is the size of the indexing term set, and T is the transpose operator. The weight of a term can be determined in many ways. A common approach uses the so-called tf*idf method, in which the weight of a term is determined by two factors: how often the term j occurs in the document i (the term frequency $tf_{i,j}$) and how often it occurs in the whole collection (the document frequency df_j). Precisely, the weight of a term j in document i is: $w_{i,j} = tf_{i,j} * idf_j = tf_{i,j} * \log N/dfj$, Where N is the number of documents in the document collection and idf stands for the inverse document frequency. This method assigns high weights to terms that appear frequently in the document but infrequently in the rest of the collection [26]. The idea of an inverse document frequency is to measure how good particular terms are as a document discriminator-that is to distinguish the few documents in which they occur from the many from which they are absent. Both query and documents terms can be weighted to distinguish terms that are more important from those that are less important for retrieval purposes. A collection of n documents is then represented by a $t \times n$ term-document matrix D

$$D = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1n} \\ w_{21} & w_{22} & \cdots & w_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{t1} & w_{t2} & \cdots & w_{tn} \end{bmatrix}.$$

For queries, a similar matrix representation is provided as

$$Q = \begin{bmatrix} q_{11} & q_{12} & \cdots & q_{1l} \\ q_{21} & q_{22} & \cdots & q_{2l} \\ \vdots & \vdots & \ddots & \vdots \\ q_{t1} & q_{t2} & \cdots & q_{tl} \end{bmatrix}.$$

Where l is the size of the query vector set. q_{zi} is the weight of term z in representation of the query $\vec{q}_i$. When a query $\vec{q}_i$ is given as

$$\vec{q}_i = (q_{1i}, \ldots, q_{ti})^T.$$

The similarity between a document and a query both are measured using the following formula.

$$\mathrm{Sim}_{\mathrm{VSM}}(\vec{q}_i, \vec{d}_j) = \cos(\vec{q}_i, \vec{d}_j) = \frac{\sum_{z=1}^{t}(q_{zi} w_{zj})}{\sqrt{\sum_{z=1}^{t} q_{zi}^2}\sqrt{\sum_{z=1}^{t} w_{zj}^2}}.$$

The denominator in the formula is called normalization factor. It consists of two parts (see the formula)—square root of sum of query term's weight square and square root of sum of document term's weight square. To compute the first part, we need to access each query term. Since user's query is normally short, the computational cost is not very high for this part. However, to compute the second part, we need to go through every term in a document and documents are normally long, the computational cost is very high for this part. The function of the normalization is to remove the effect of document lengths on document scores. Thus, a document containing {a, b, c} will have exactly the same score as another document containing {a, a, b, b, c, c} because these two document vectors have the same unit vector [26].

3.4 VSM-Implementation

After selecting the vector space model, the next important task is how to implement it. One important characteristic of our "documents" is that the "document" is very short; it consists of the

concatenation of a number of columns of interests from our database. Based on this special "document" property, some research on the implementation of Vector space model was conducted.

For most cases, a single document only contains a small fraction of total number of terms indexed over the entire collection of documents, thus the full vector representing the document is always very long and sparse. Therefore a full document vector is rarely used. Instead, document vectors are stored in an inverted file. An inverted file consists of two parts, namely the index terms and the posting lists. Each index term corresponds to an indexed document term in the collection, and it is associated with a posting list. Each item in a posting list records the document that contains the term, and it may also include some other information such as the corresponding term frequency, depending on the retrieval environment. An inverted file is shown in Fig.1 [27], in which the document frequency of each term is stored together with the index term, and the term frequency of each term is stored together with the document. A query will be presented as a list of terms with associated weights. The posting lists corresponding the query terms are retrieved and, from the posting lists, the document scores can be computed as shown in the pseudo-code below [27].

Pseudo-codes:

For every query term q in Q **do**
  Retrieve the posting list for q from the inverted file
  For each document d indexed in the posting list **do**
    Score (d) = Score (d) + $tf_{d,q} \times idf_q$
  End {for}
    End {for}
    Normalize scores
    Sort document according to normalized scores

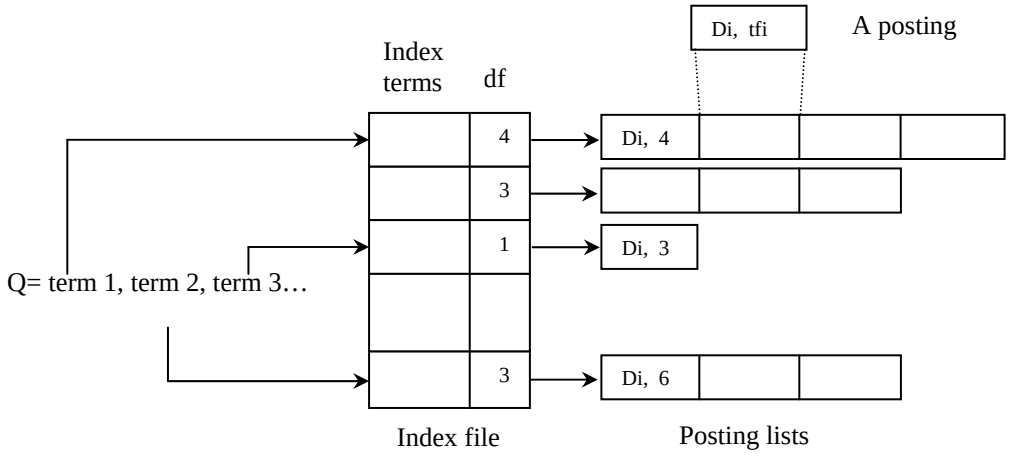

Fig. 1

With an inverted file, the number of posting lists accessed is equal to the number of query terms. The number of vector comparisons is greatly reduced [26]. The computational cost is acceptable for queries of reasonable size. We will implement an inverted file in our search engine.

Because the exact vector space model is expensive to implement, Dik Lee etc. [26] proposed a family of successively simpler approximations.

Method 1: Full Vector Space Model

This method uses inverted file to implement the full vector space model. Unfortunately, the computation of the normalization factor is extremely expensive because the term in the normalization factor requires access to every document term (see section 3.3 for detail). The normalization factor also cannot be precomputed, because every insertion and deletion on the document collection would change idf and thus the precomputed normalization factor. Since our database will be often updated, we will not select this method.

Method 2: Approximate Normalization

This method uses the square root of the number of terms in a document to approximate the normalization factor. While this still favors long documents, the effect of document size is not as significant as it is without any normalization. This normalization factor is much easier to compute than the original one; also, precomputation is possible. With the approximation, the formula becomes

$$sim(Q, D_i) = \frac{\sum_{j=1}^{v} w_{Q,j} \times w_{i,j}}{\sqrt{number\ of\ terms\ in\ D_i}}$$

Method 3: Dropping the Normalization Factor

This method lets us further simplify the computation by simply dropping the normalization factor.

$$sim(Q, D_i) = \sum_{j=1}^{v} w_{Q,j} \times w_{i,j}$$

Method 4: Ignore *idf*

This method only makes use of term frequencies in the calculation and ignores *idf*. It simplifies the computation as well as saving the inverted file structure needed for storing the *df* values.

$$sim(Q, D_i) = \sum_{j=1}^{v} w_{Q,j} \times tf_{i,j}$$

Method 5: Ignore *tf*

This method ignores the term frequency but retain the *idf* values in determining term weights.

$$sim(Q, D_i) = \sum_{j=1}^{v} w_{Q,j} \times w_{i,j} \qquad w_{Q,j} = \begin{cases} 1 & j \in Q \\ 0 & otherwise \end{cases} \qquad w_{i,j} = \begin{cases} idf_j & j \in D_i \\ 0 & otherwise \end{cases}$$

Method 6: Simplest one, ignore both *tf* and *idf*

This method is the simplest in the family. It ignores both *tf* and *idf* values and therefore measures the number of common terms in the document and query vectors.

$$sim(Q, D_i) = \sum_{j=1}^{v} w_{Q,j} \times w_{i,j} \qquad w_{Q,j} = \begin{cases} 1 & j \in Q \\ 0 & otherwise \end{cases} \qquad w_{i,j} = \begin{cases} 1 & j \in D_i \\ 0 & otherwise \end{cases}$$

In order to evaluate their retrieval performance of these six methods, some experiments were conducted on a set of collections. Six documents collection were used for the test. Five of them come from the Smart system developed at Cornell University. The last one called TREC (Text Retrieval Conference) subset contains 10,000 Wall Street Journal articles (see the paper [26] for detail information). Each collection has a standard set of queries. The TREC subset collection has two kinds of queries---narrative and concept query. The other five collections only provide narrative queries.
The average precision for each method was computed and a comparing graph was drawn.

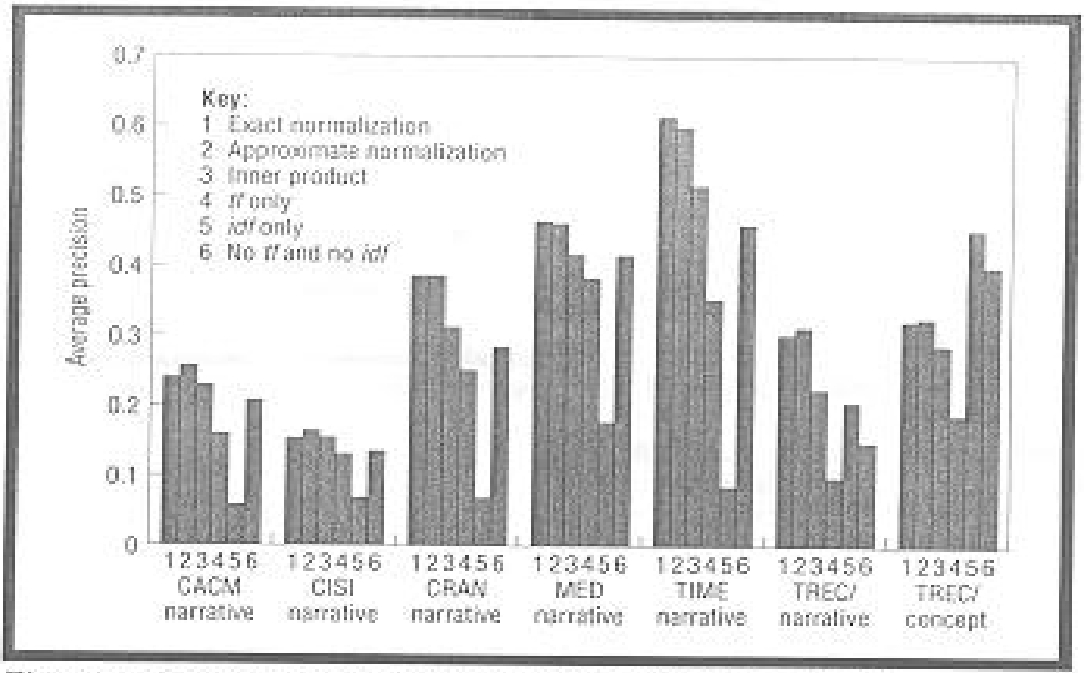

Figure 4. Average precision for the six ranking methods on each of the seven sample document collections.

Our selection among those six methods was based on two considerations, that is computation complexity and retrieval precision. For computational complexity, from method 1 to method 6, the computation cost gradually decreases; therefore, method 6 has the least computational cost. For retrieval precision, from the above graph, it is clear that the last three methods are not as good as the first three methods for natural language queries. However, method 6 gets better results than method 4 and 5. Particularly, method 6 performs extremely well for concept documents and query. Concept documents consist of concept terms. Concept terms are content descriptors. For example, a document can be described as follows in natural language:
A relevant document will identify an information retrieval system, identify the company or person marketing the system, and identify some of the characteristics of the system.
A comparable description using concept terms would be: *information retrieval system, storage, database, data, and query* [26]. The concept terms are very precise content descriptors, thus, the appearance of a concept term in a document will always reveal the document's content and relevance, regardless of the term frequency information. This is the exact reason that why method 6 performs extremely well for concept query.

From the above discussion, we can see that method 6 has the least computation complexity and extremely good retrieval performance for concept documents. As mentioned before, our documents were formed by combining several columns of a table from the database, such as artist, title, subject, category and description column etc. Each document is a short piece of text that is densely packed with potential search terms. The document consists of concept terms indeed**.** Therefore, method 6 should perform very well on our documents. Based on the above consideration, method 6 was selected.

The final selection for our search engine is the simplest approximation method (method 6) for Vector space model of text-based search. With the selection of a relevancy ranking search algorithm the initial research was complete.

4. Search Engine Implementation

As usual, our search engine consists of three parts, which are the Front-end (user interface), the Back-end (database) and the searching algorithm implementation. Several simple and user-friendly interfaces were designed for the Front-end part. The Back-end is just the database system that stores the original data. It will be explained in detail later. The heart of our search engine is the searching algorithm implementation. It will be discussed in section 4.3.

A typical searching procedure can be described as follows. First, a user sends a query to the search engine. After getting the query, the search engine maps the user's query into a query language appropriate to the database in order to find the specific images that meet the user's criteria. Finally, the matching results are presented to users. If the search is a text search, relevancy ranking will be conducted on those matching images, and the ranked results will be returned to the user. To clearly show the relationship between these three parts, the following picture was drawn.

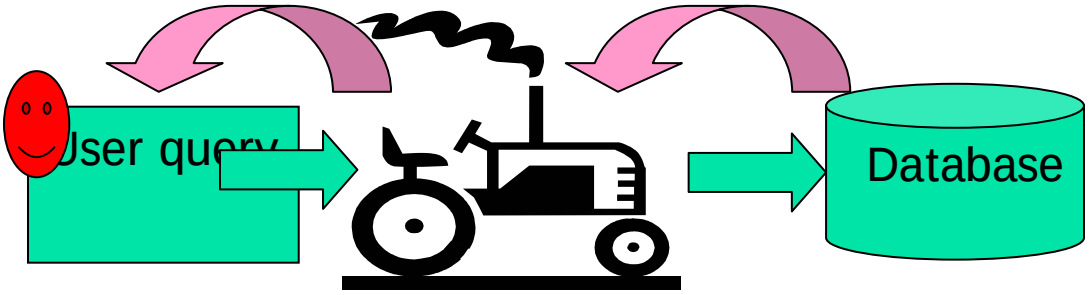

Fig 2 Search engine three parts relationship

4.1 Front End (User Interface)

Several simple search methods (such as search by author, subject, status and category etc.) and two advanced search methods (search by keywords and text) were implemented. Whether a simple search or an advanced search is conducted, six windows will pop up successively. Next the six windows will be described in detail.

1. The First Window

In the first window, all possible search methods are provided. They are organized by two categories--simple and advanced search (see Fig. 3).

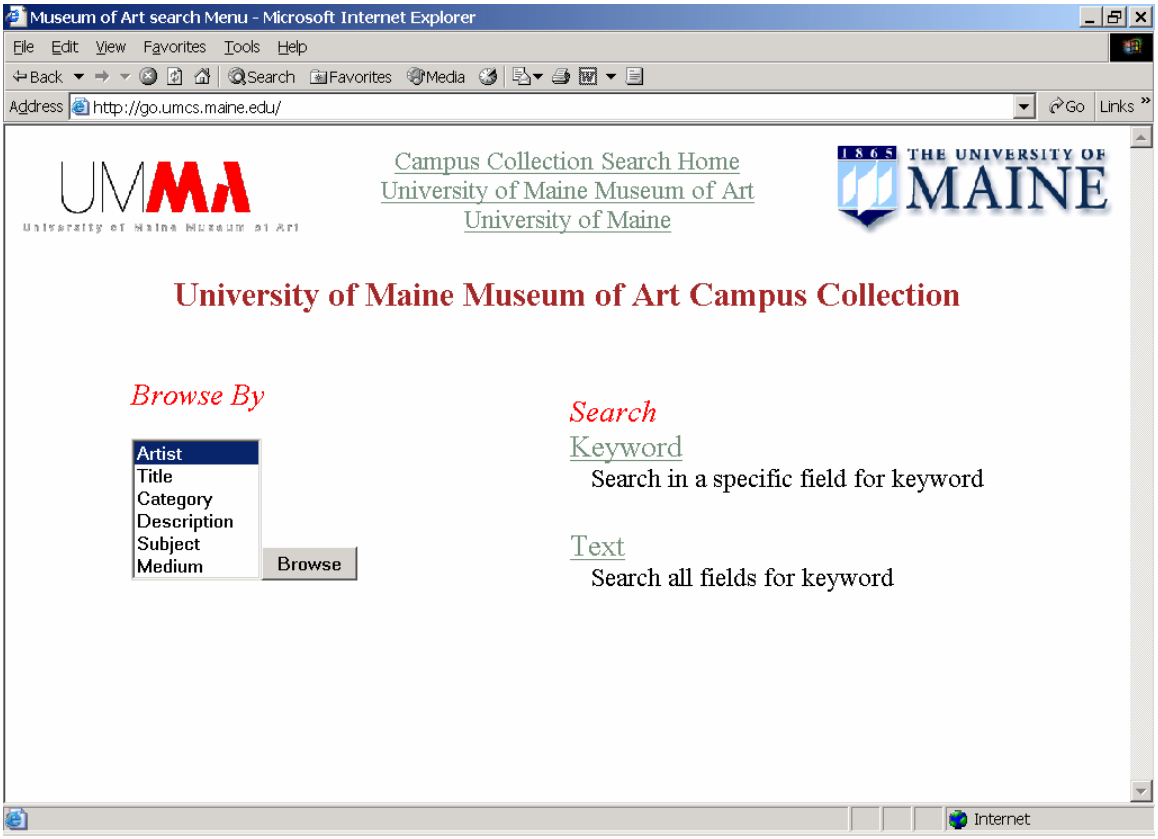

Fig. 3 The first window

2. The Second Window

Once a search method is selected, an input window (second window) will pop up. Three types of input boxes were designed. One was designed for simple searches (see Fig. 4). Another was designed for keyword search (see Fig. 5). The last one was designed for text search (see Fig. 6). For text searches, users can speak natural language; users can input one or several terms or phrase that express their main idea. The search engine will search all columns of all records for the term or terms and rank the results using a relevancy algorithm. Terms must be separated by spaces. Other symbols (such as colon, semicolon, etc.) will not be treated as separators. The search engine is not case-sensitive; upper and lower case characters are interpreted as equivalents. It does not search for stop words (see search tips in 4.4 for detail).

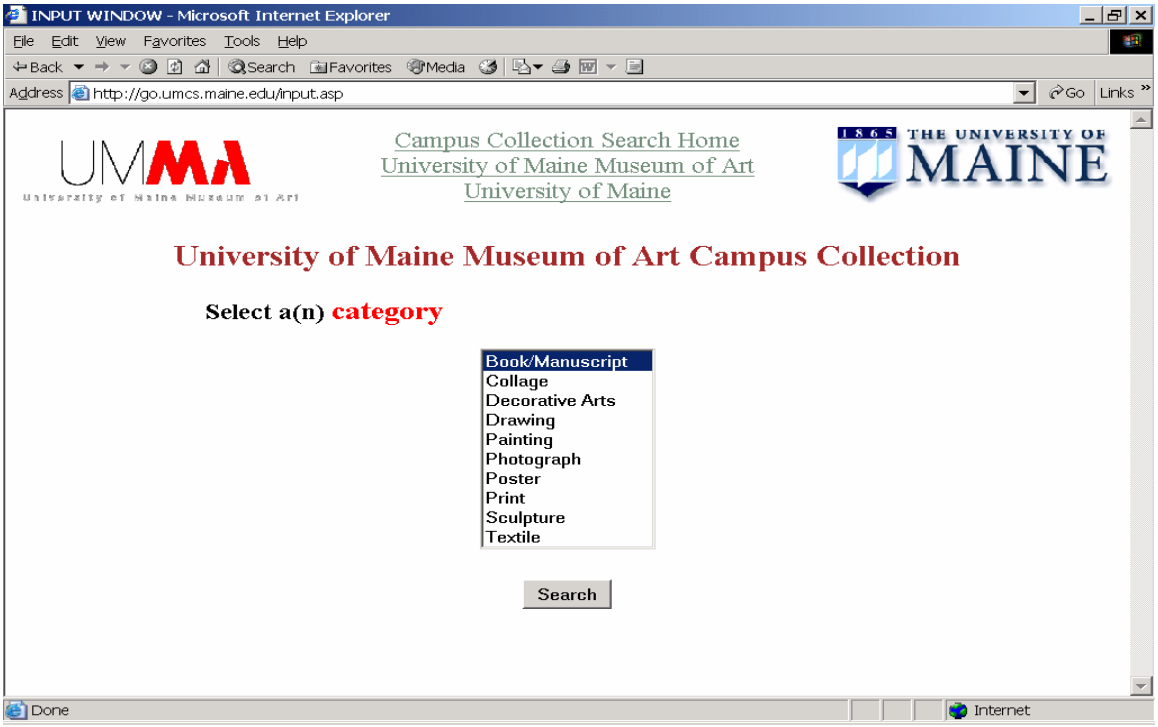

Fig. 4. Simple Search Input Window.

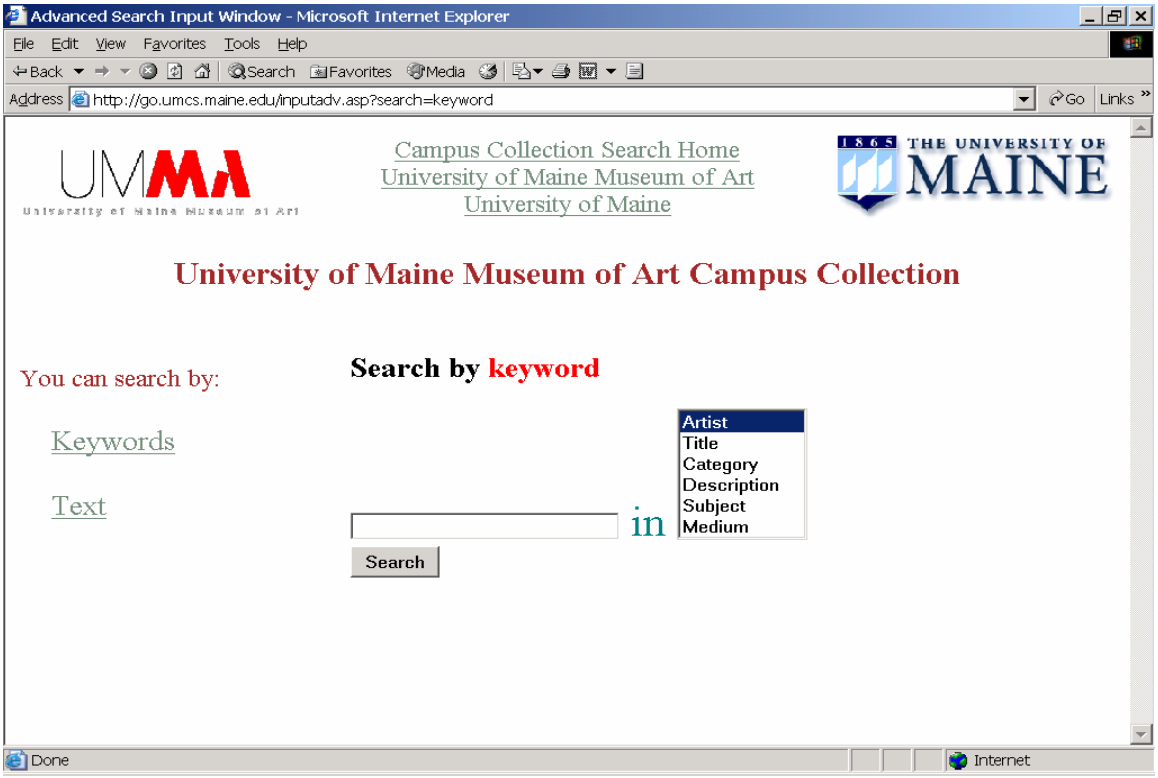

Fig. 5. Keyword search input window.

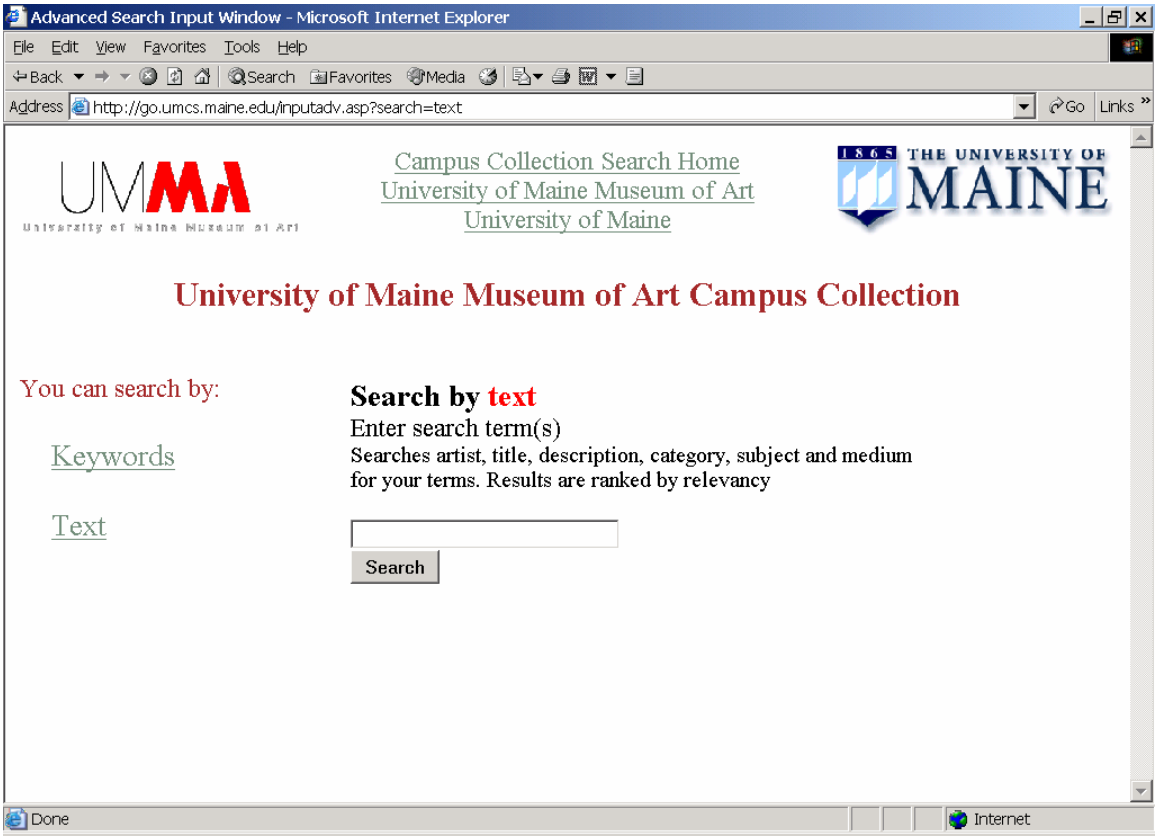

Fig.6 Text Search Input Window

The search engine also does input validation checking. If a search field was not selected or no keywords were inputted, a little warning window (see Fig. 7 and Fig. 8) will pop up to catch this kind of error. Users have to choose a field and input a keyword; otherwise users cannot continue the searching.

Fig. 7 No Keyword Input Warning Window

Fig. 8 No Field Selected Warning Window

3. The Third Window

The third window shows the searching results using a table. Except for the text search, the tables are the same (See Fig. 9). If a text search is conducted, the matching results will be ranked and the ranking scores will be displayed. Fig. 10 shows the table for text search.

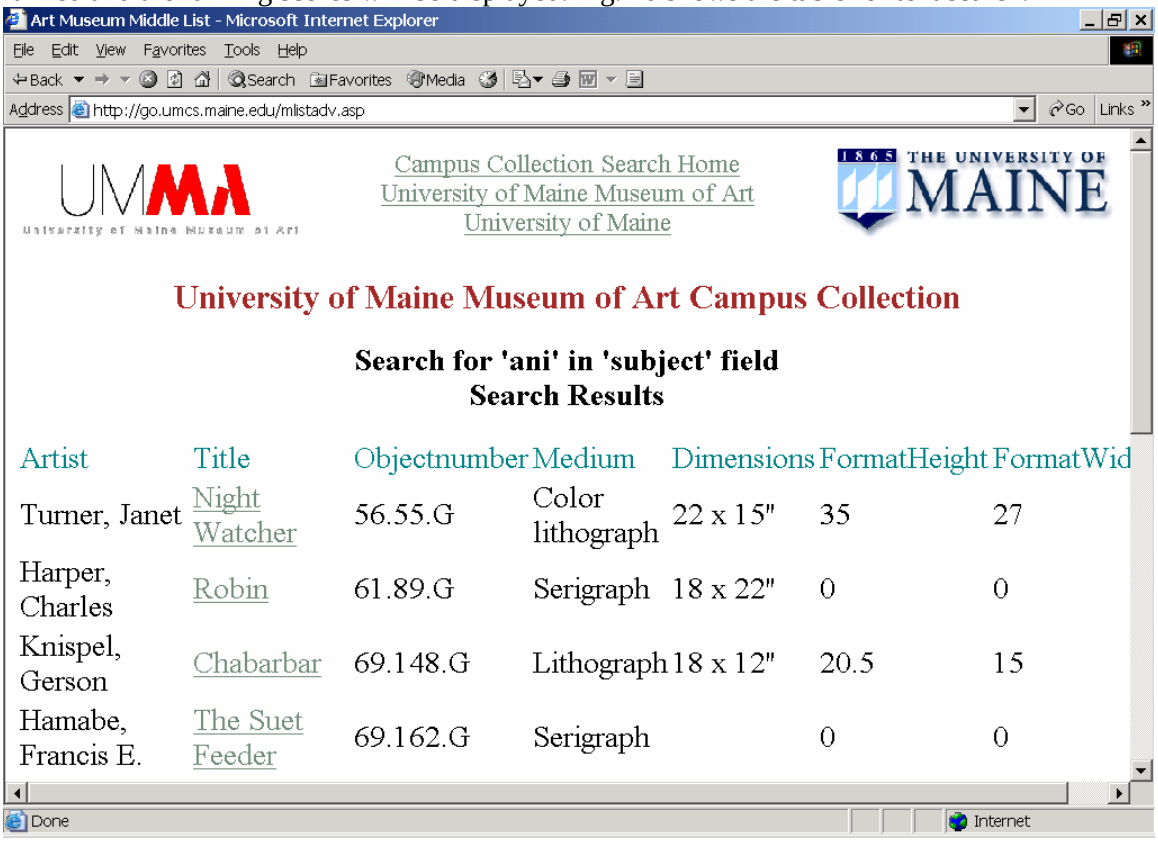

Fig. 9 The third window (not text search)

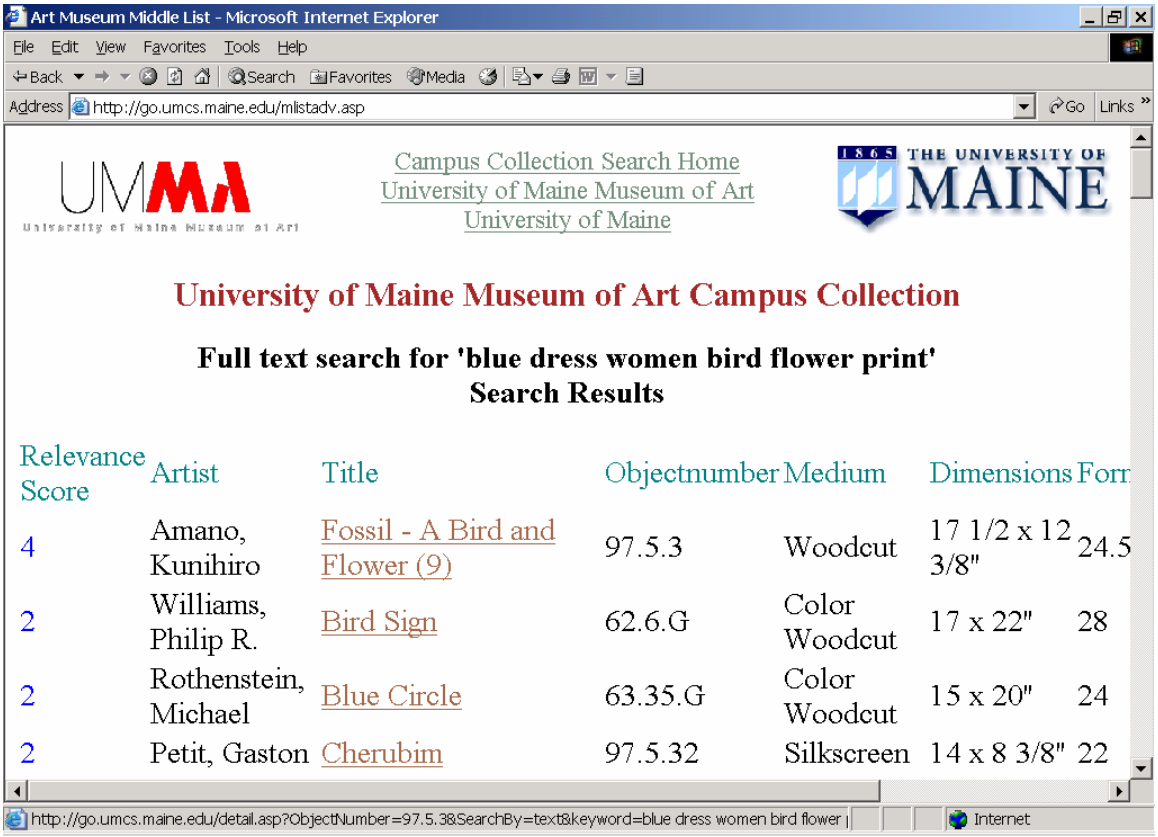

Fig. 10 The third window (text search)

If there are no matching results found, then the following window (Fig. 11) will pop up:

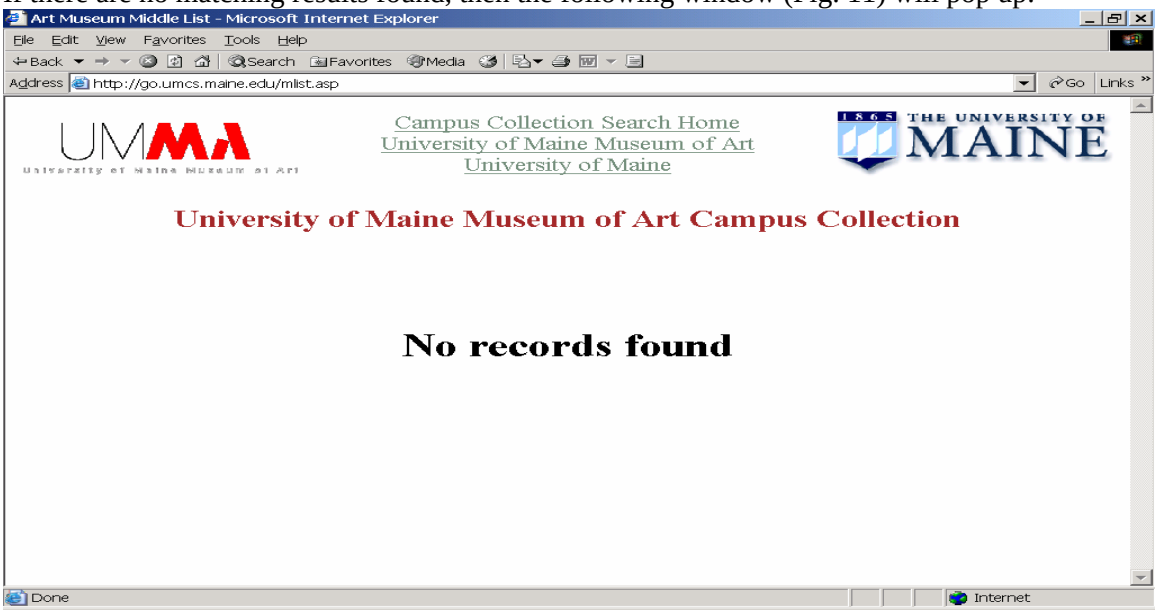

Fig. 11 No Result Found Window

4. The Fourth Window

In the third window, each result is associated with a link. To see the detail information (such as title, author, subject etc.) about an artwork, click its link, then the fourth window (see Fig. 12) will pop up, on which the detail information about this artwork will be shown.

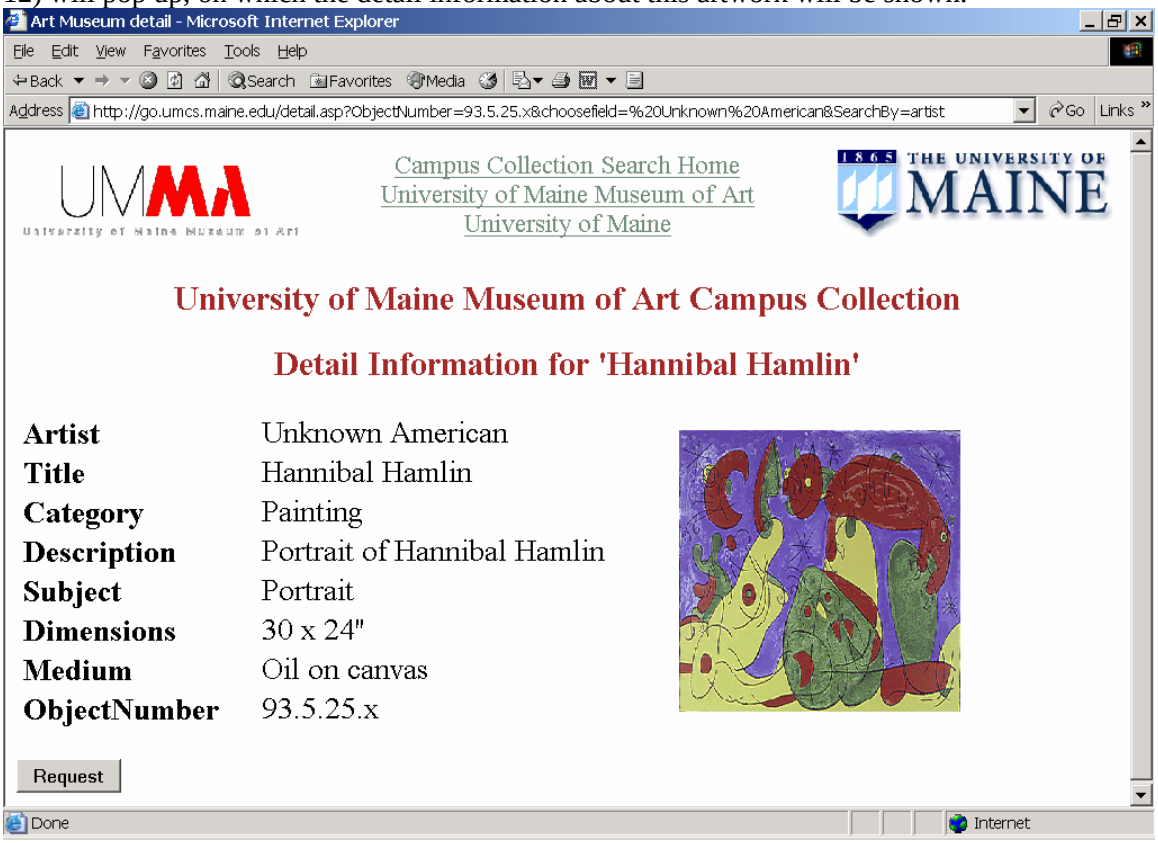

Fig. 12 The Fourth Window

5. The Fifth Window

The fifth window is designed to collect users' information, such as name, phone, email etc. The information such as Artist, Title and ObjectNumber about the requested artwork is shown for users' convenience. If the 'cancel' button is clicked, the detail window will pop up.

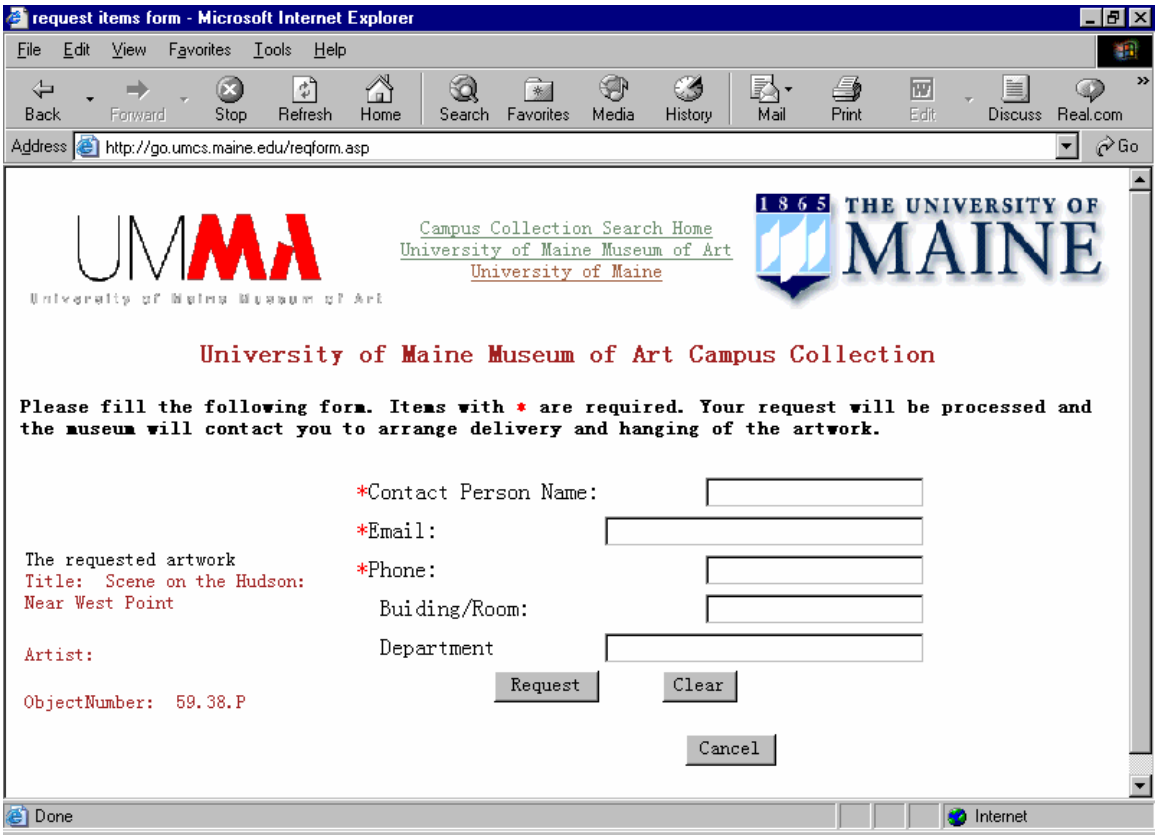

Fig. 13 The Fifth Window

6. The Sixth Window

The sixth window just notes you that your request has been sent to Museum.

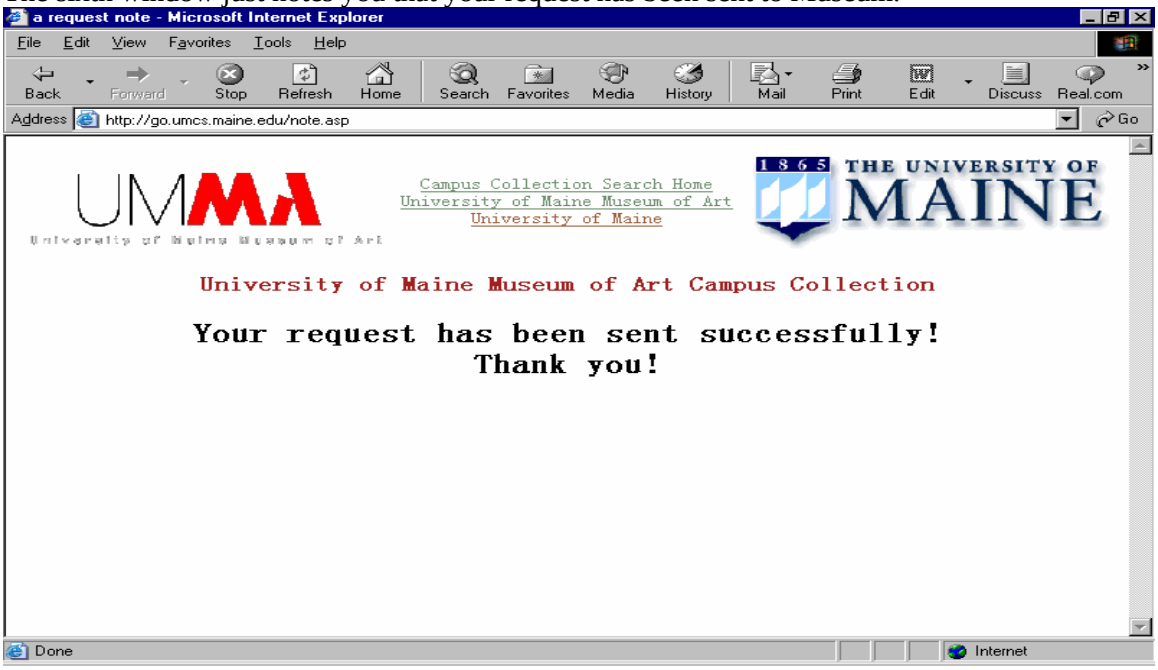

Fig. 14 The Sixth Window

4.2 Back End (Database system)

Information about images is stored in a relational database using MS Access. The database consists of ten tables. They are:

- Main
- Artists
- Subjects
- Categories
- Department
- Source
- Location History
- Status
- Buildings
- Borrower

The primary table Main stores all single-valued attributes of an artwork such as artist, subject, category, value, and dimensions.
Main-schema = {ObjectNumber, SortingNumber, AltNumber, Collection, Artist, DisplayDate, Category, EditionNumber, SignatureLocation, Subject, Description, Place, Dimensions, Medium, Formating, FormatHeight, FormatWidth, Source, Status, Department, Building, Room, Spot, DatePlaced, Value, ValuationDate, ContactID, IndexNum}

The following two screen dumps (Fig. 15 and Fig. 16) show part of the design and datasheet window of the 'main' table.

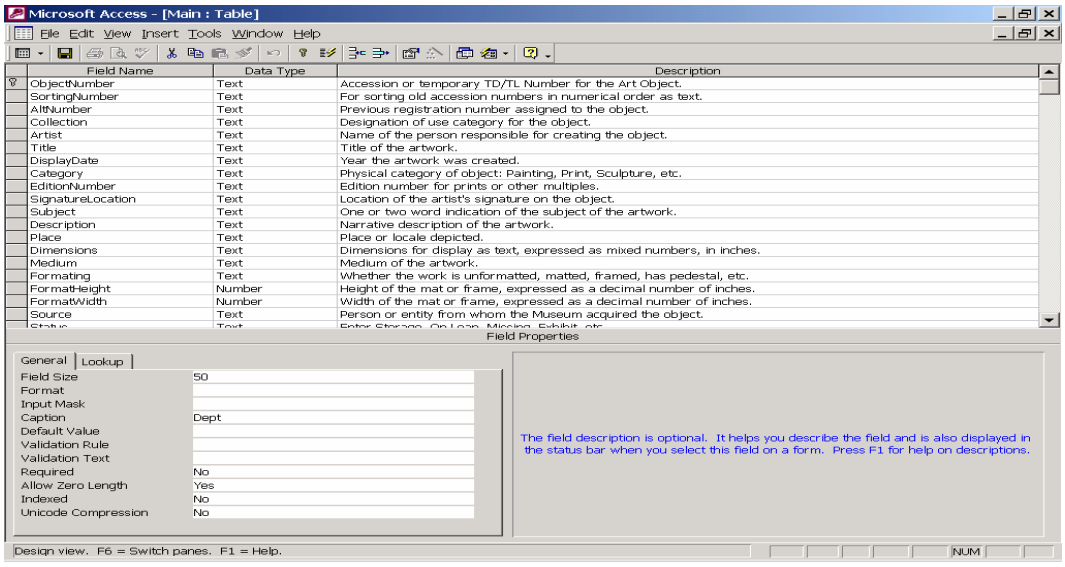

Fig. 15 Part of the Design of the 'Main' Table

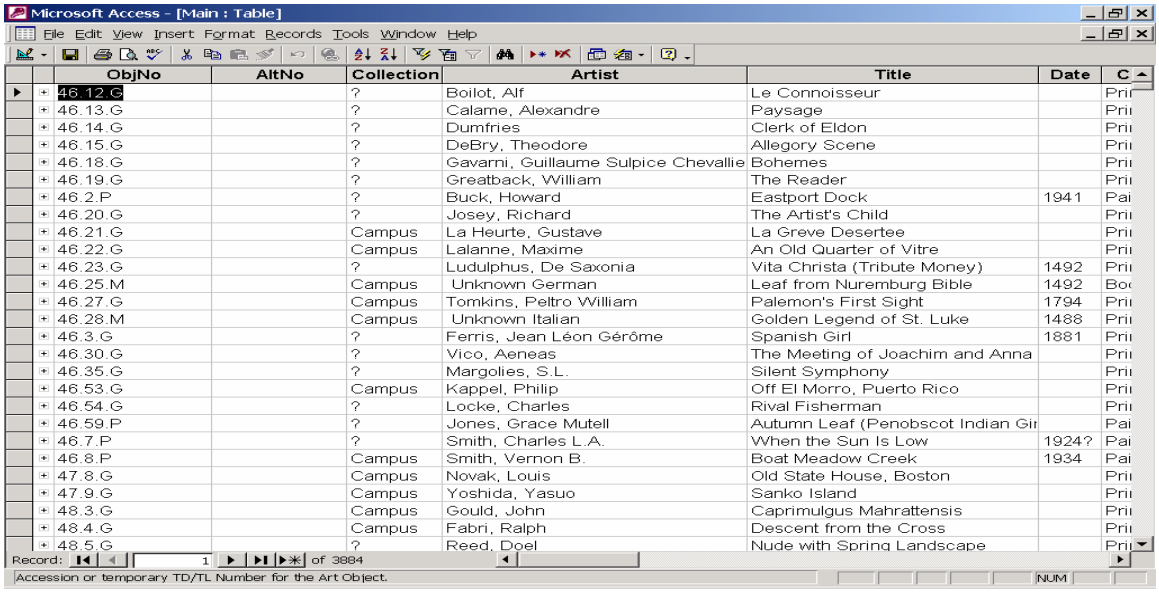

Fig. 16 Part of the Datasheet of the 'Main' Table

Of the remaining eight tables, six function as solely as lookup tables for enforcing controlled vocabularies in columns of the main artwork table. These are Artist, Subject, Category, Department, Source, and Status. The schemas are as follows. Location History table records the information about the place and date of an artwork movement. Building table records Building name for Campus Loans and Borrower name for Museum Loans and Date Building was last inventoried. The information (such as name, working place and phone etc.) about the person who borrowed an artwork are stored in Borrower table. The schemas for them are as follows.

Artists-schema = {artist}
Subjects-schema = {subject}
Categories-schema = {category}
Departments-schema = {department}
Source-schema = {source}
Location history-schema = {ObjectNumber, DatePlaced, Department, Building, Room, Spot, Status}
Status-schema = {status}
Buildings-schema = {Building, InventoryDate}
Borrower-schema = {ContactID, Contactperson, Department, Phone, Building, Room}

The following diagram (Fig. 17) shows the relationships.

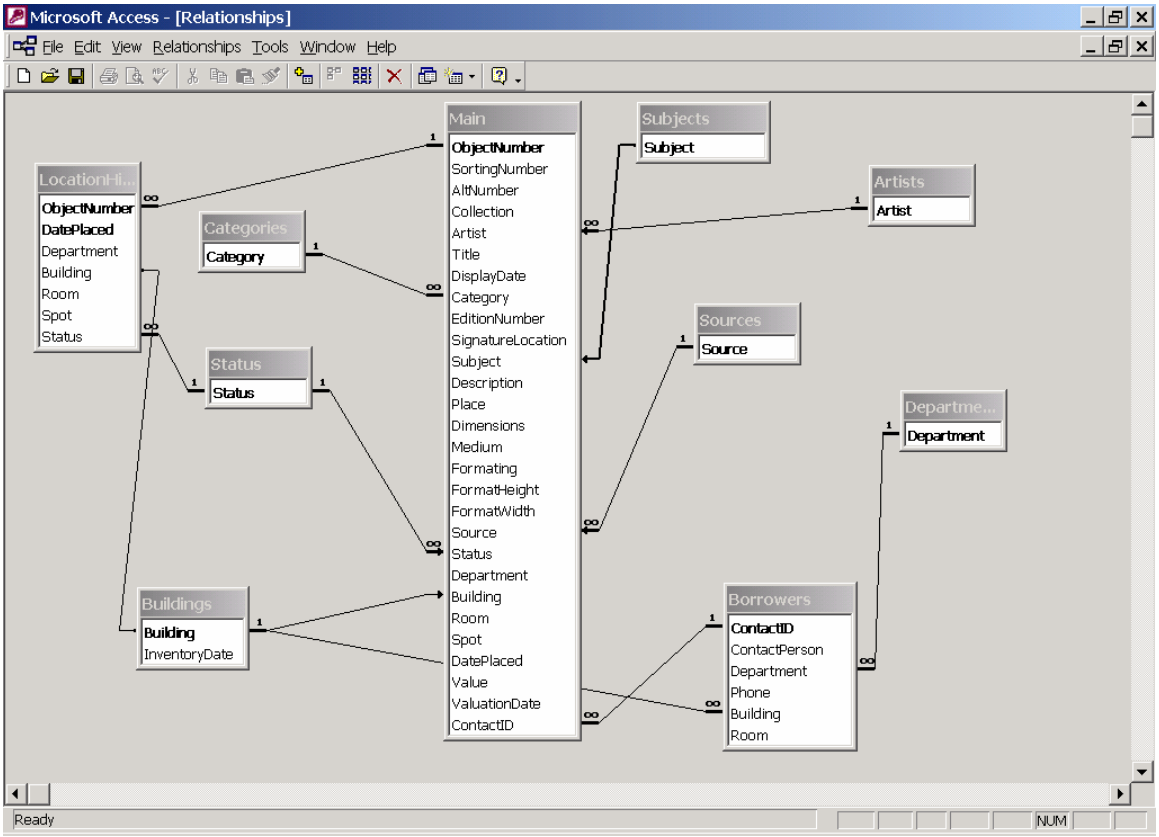

Fig. 17 Tables Relationships

To implement the relevance ranking algorithm for the text search, three new tables were added named MixTable, ComTerms and keywordIndex. The MixTable contains two columns—ObjectNumber and Mix. The Mix column is formed by concatenating all searchable columns from the Main table. An update query was created to fill these two columns by using the data from the Main table. The ComTerms table has one column ComTerm. It is a stop list that stores all common words such as the, we, has, where etc. The keywordIndex table has two columns--keyword and ListOfObnum; it is sorted by the keyword, which is also the primary key. These three tables will be explained in detail later.

Followings are the screen copies (Fig. 18, 19 and 20) for them.

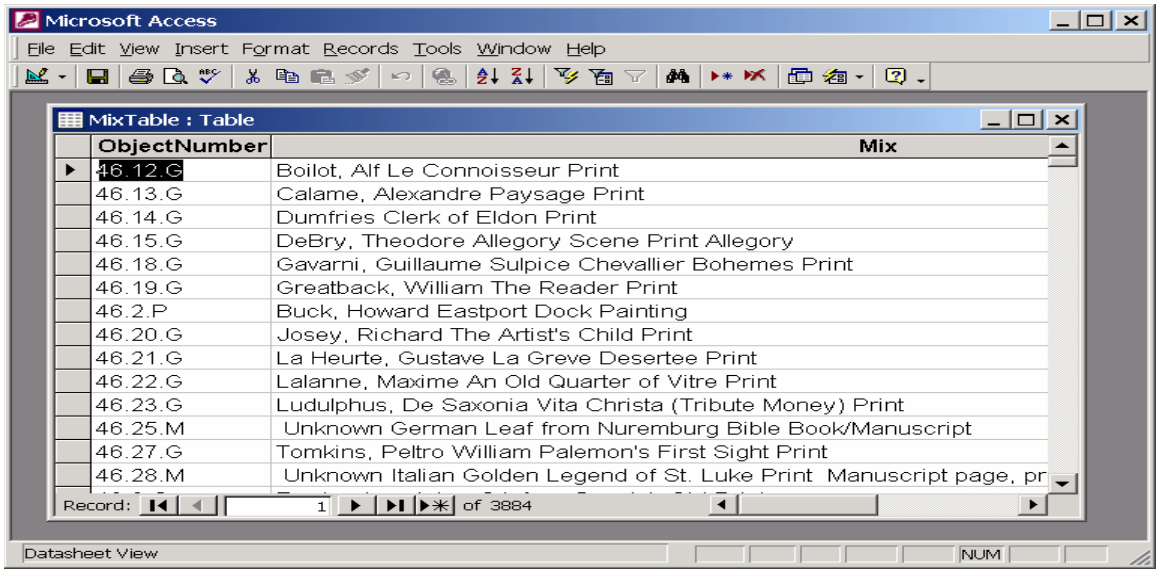

Fig. 18 MixTable

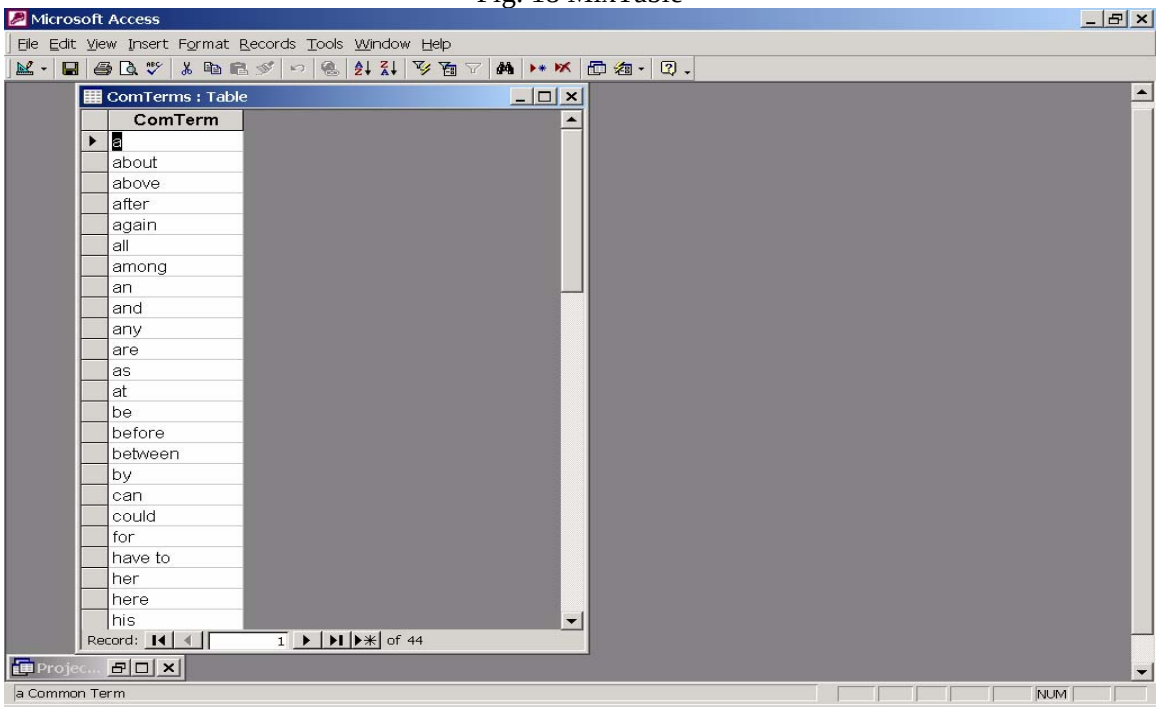

Fig. 19 ComTerms table

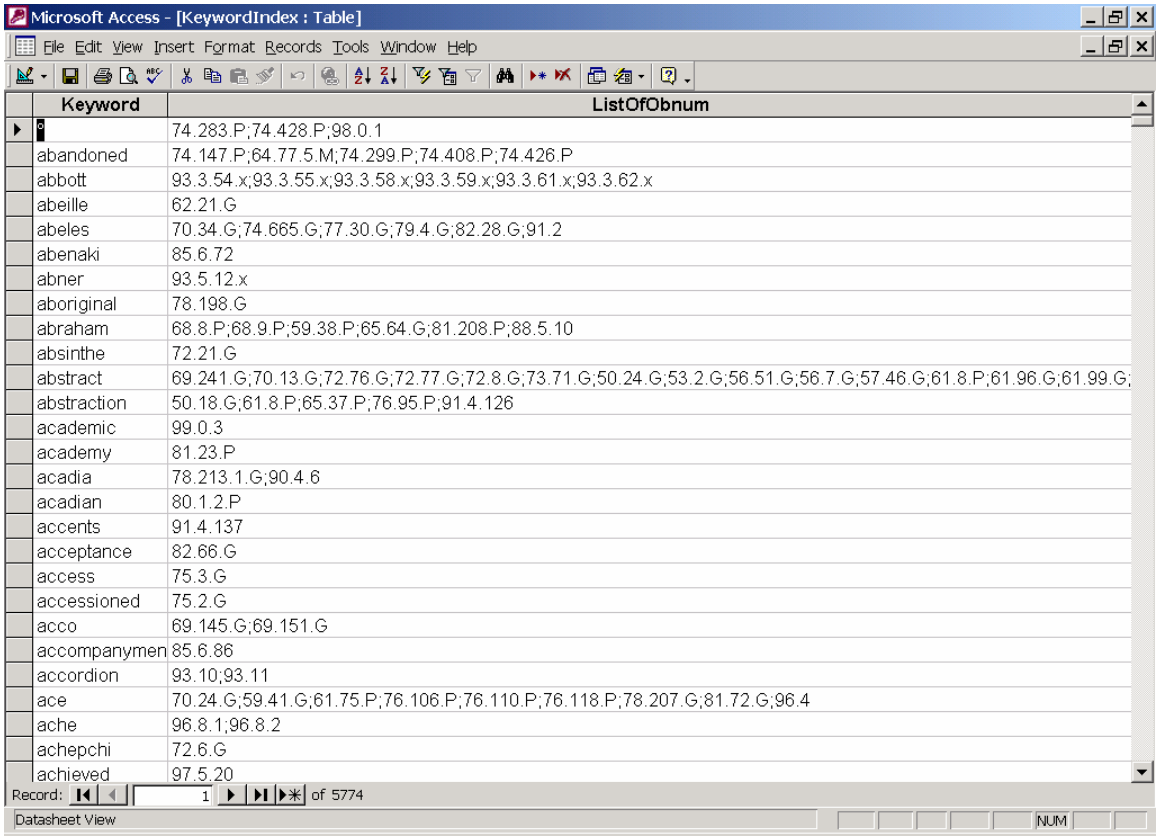

Fig. 20 KeywordIndex table

Our search engine is platform independent because the primary interface between the search engine and the back end database is ANSI standard SQL. Although the back end database was implemented using MS Access, the search engine can continue to function with minimal changes using a different backend database such as Oracle or MySQL.

4.3 Search Engine Implementation
HTML, VBScript, JavaScript, VBA, CSS and SQL were used together to develop the search engine. Source code is shown in Section 7.

As we knew from section 4.2, the front-end part has six kinds of windows. Actually it has eight kinds of windows, because the second and the third windows can be further categorized into two windows separately based on the complexity of the search (simple or advanced). To implement those functions involved in those eight windows, one VBScript program was written for each window, therefore eight files were written, which are Mainmenu.asp, input.asp, inputadv.asp, mlist.asp, mlistadv.asp, detail.asp, reqForm.asp and sendemail.asp. The VBScript programs produce HTML output. They are executed on the web server, which delivers the HTML output to the client browser. These eight files will be discussed below in detail. To format the HTML pages, a simple css file called format.css and Generalheader.asp were also written.

If a simple search is conducted, the files will be executed in the following order:
Mainnemu.asp→input.asp→mlist.asp→detail.asp→reqForm.asp
→Note.asp

If an advanced search is conducted, the order will be:
Mainnemu.asp→inputadv.asp→mlistadv.asp→detail.asp→reqForm.asp
→Note.asp

To clearly show the relationship between the six files, the following figure (Fig. 21) was drawn.

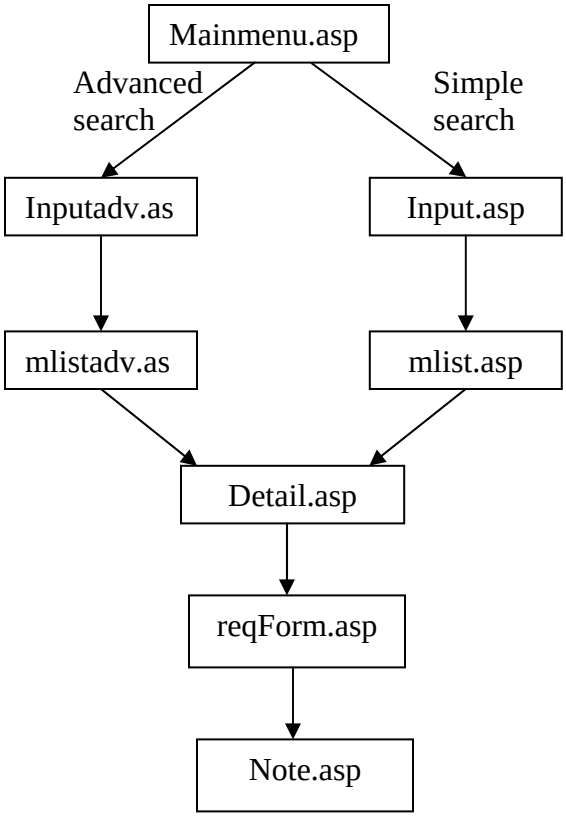

Fig 19 Relationships between Front-End Programs

1) Mainmenu.asp File

All files include a common header file called GeneralHeader.htm, which contains a table with one row and three columns (three cells). The University of Maine Museum of Art logo and the University of Maine logo fill the first and the last cell. In the middle cell, three links are provided for 'Campus Collection Search Home', 'University of Maine' and 'University of Maine Museum of Art' respectively.

Mainmenu.asp is used to implement the first window in the front-end part. A table was designed. It has one row and two columns (two cells). One cell stores a form that contains a selection box. Six searching available methods fill the entries of the selection box. The action attribute of the form is set to 'input.asp'. The other cell stores two advanced search methods. Each advanced search method associates with a link, which contains a parameter. For example, <A HREF = "inputadv.asp?search=text">text, is a link associated with the method 'text' and it has a parameter 'search,' which has a value 'text.' If this link is clicked, the parameter 'search' and its value 'text' will be sent to 'inputadv.asp' page. Parameters that are hardcoded into links such as this are returned to the server as the query string; this is equivalent to form submission by the HTTP "GET" method.

2) Input.asp File

This file is for the implementation of the input interface (second window in the Front-End part) for the simple search. It only has a HTML form, which contains an HTML <select> element filled by an entry list showing all available selection options. The entries will change with the selected search method. For example, if a user selected search by 'title,' then the entries will be filled by all distinct titles stored in the 'Main' table.

To fill the options of the selection box, we need to connect to our database. DSN-less OLEDB connection method was selected to do the connection. An application variable was created in a global.asa file. The variable was assigned by a DSN-less OLEDB connect string as follows.

Application("ConnectString") = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source="& Server.MapPath(".") & "\database\try.mdb".

If a page needs to connect to the database, it just creates an 'ADODB.Connection' object and then opens the application variable.The exact same connection method is used in all other pages.

This page gets an input parameter 'search' from the MainMenu.asp page. Based on the value of the parameter, a query is constructed in SQL and posed to our database to retrieve the matching records. Then it uses theresult set of the query to build the input entries in the combo box.

To complete those tasks, two sub-procedures named GetForm and BuildEntries were written. The GetForm sub-procedure sets up the connection to database, constructs a query, issues the query and gets matching records. The BuildEntries uses the records retrieved by GetForm to build the entries in the combo box.

3) Inputadv.asp File

This is similar to the input.asp. It is used for the implementation of the input interface of the advanced search, while the input.asp file is used for that of the simple search. It contains a one row and two column (two cells) table. The first cell is filled by two links, which link to keyword and text search. The second cell contains a form. Since we have two advanced search methods named keyword search and text search, two kinds of form were designed for each of them. For keyword search, the form contains a selection box, a submit button and an input text box. For text search, it has an input text box and a submit button.

The HTML code for those two forms was organized into two sub procedures-- BuildFields and InputText. The BuildFields sub-procedure builds a form, which contains a selection box and an input text box. The selection box is filled by Artist, Title, Category, Description, Medium and subject. The InputText sub procedure is very simple, which just provides an input text box.

This page gets input parameter 'search' from the MainMenu.asp page.
If the value of 'search' is Keyword, it will call BuildKeyword, otherwise call InputText.

4) Mlist.asp File

Two files (Mlist.asp and Mlistadv.asp) were written for the implementation of the third window in the Front-End part. Mlist.asp file is for the simple search and Mlistadv.asp is for the advanced search.

This file contains a table that is used to show the retrieved results to user. The table's columns (header) are filled with some column names from the 'Main' table. The table's rows are filled with the retrieved records.

To fill the table's header, a query is constructed in SQL and issued, and some column names of the "Main" table are retrieved, which are used for the table columns.
To fill the table's rows, a query is built in SQL based on the two input parameters named searchby and choosefield, which were posted by the input.asp file after a user did a selection in the input window. Then this query is issued and the matching results are retrieved, which are used to fill the table rows.

This page was organized into one main procedure and four sub procedures named ShowResults, ShowHeader, TableHeader and ShowFooter. The main procedure is very simple; it constructs a query based on the user's selection, then opens our database and retrieves the matching records. ShowResults, ShowHeader, TableHeader and ShowFooter show the retrieved results in a table, a HTML page header,
a table header and a HTML page footer respectively.

The program will be executed in the following way. The main procedure will call ShowResults if some matching results are obtained, otherwise tell users no results found. The ShowResults will call ShowHeader, TableHeader and ShowFooter in order.
.

5) Mlistadv.asp File
This is the most important and difficult file in this project. Keyword and text search were implemented here.
This file is similar to the Mlist.asp file. It also uses a table to display the search results. For the keyword search, the table header is the same as that in the Mlist.asp. To fill the table's rows, the same operations are performed as that in the Mlist.asp file. However, a different query is built in SQL, which is based on the two input parameters named searchkeyword and choosefield that were posted by the inputadv.asp file. For the text search, in addition to the headers in the keyword search table, another new header was added called relevance score. To compute the relevance score, the selected relevance ranking algorithm (method 6 in paper [26]) was implemented. A query was posed to retrieve the matching records, which are used to fill the table rows.

This page was organized into one main procedure and ten sub procedures namely KeywordSearch, ShowKeywordResult, KeywordTableHeader, TextSearch, ProcessRecord, InsertionSort, ShowTextResult, TextTableHeader, ShowHeader and ShowFooter.

The main procedure is very simple. It opens our database and calls KeywordSearch or TextSearch based on the value of the 'searchby' parameter that was posted by the inputadv.asp.

KeywordSearch, ShowKeywordResult, KeywordTableHeader ShowHeader and ShowFooter are called to perform a keyword search.
KeywordSearch connects to the database, constructs and issues a query in SQL, and retrieves matching results based on the keyword and the search field that the user selected. ShowKeywordResult, KeywordTableHeader, Showheader and ShowFooter display the retrieved results in a table, the keyword search table header, a HTML page header and a HTML page footer respectively.

The execution order can be shown as:
KeywordSearch→ShowKeywordResult→
ShowHeader→KeywordTableHeader→ShowFooter.
KeywordSearch calls ShowKeywordResult, which calls ShowHeader, KeywordTableHeader and ShowFooter in this order.

TextSearch, ProcessRecord, InsertionSort, ShowTextResult, TextTableHeader, ShowHeader and ShowFooter are written to implement the text search. TextSearch connects to our database, constructs a query in SQL based on the input words, which were posted by the inputadv.asp after a user clicked the submit button, issues this query and retrieves the matching records (documents). ProcessRecord computes the relevancy scores for those retrieved documents using our selected relevance ranking algorithm. Then it saves those scores and the corresponding documents in an array. InsertionSort sorts the array by the score number. ShowTextResult just displays the retrieved results by calling TextTableHeader, ShowHeader and ShowFooter in order.

The execution order can be shown as:
TextSearch→ProcessRecord→ InsertionSort→ShowTextResult
→ShowHeader→TextTableHeaderand →ShowFooter
The TextSearch calls ProcessRecord, which calls InsertionSort and ShowTextResult. ShowTextResult calls ShowHeader TextTableHeader and ShowFooter in order.

For the text search, an inverted table was built to implement our selected relevance ranking algorithm. The inverted table contains two columns. One is called Keyword, which stores all indexing terms contained in the campus collection. The other is called ListOfObNum, which is a list of artwork objectnumber. Since we selected the most simple approximation method (method 6), which ignores both the document frequency and the term frequency, we don't need to store their information in our inverted table. Each keyword associates with an objectnumber list. Each objectnumber in the list records the document that contains the keyword.

To build the inverted table, for each row of the mix column in the Mixtable, which is a long string that consists of some words, first, remove various kinds of symbol such as !, @, #, $, etc. and single characters such as A, b, c etc. Then remove the common terms stored in the stop list (the ComTerms table). Finally, for each word left, determine whether it has been added before. If yes, the objectnumber of current record is appended to the end of the objectnumber list, otherwise, a new keyword entry will be added and the objectnumber is added to the objectnumber list column.

To implement the above idea, several functions were written using VBA (Visual Basic for Applications). The main function is called BuildInvertedTable. It first calls PreProcessRecord sub function to remove the stop words and symbols. Then it populates the two columns of the KeywordIndex table by implementing the above idea. The sub function PreProcessRecord calls three sub functions named RemoveSymbols, RemoveChars and RemoveComns, which remove various kinds of symbols, single letters and stop words respectively.

The inverted table was created by running the VBA code. Therefore, it will be easy to do the corresponding change (by just running the VBA code again) if the database is updated.

The relevance ranking algorithm was implemented by using two loops, a 'Do While' loop and a 'For' loop. The 'For' loop is nested within the 'Do While' loop. The outer Do While' loop gets the retrieved objectnumber list one by one. The inner 'For' loop processes each objectnumber in the objectnumber list one by one. Each objectnumber actually corresponds to an artwork in our database. Its score is accumulated according to the following pseudo-code:

```
Do While there is an objectnumber list in the retrieved record
  Get a new objectnumber list
  For each document d indexed by the objectnumber in the list do
    score(d) = score(d) + 1
  End {for}
End {While}
```

After the score array is obtained, the results are sorted by the relevancy ranking score and are shown in a table to user.

6) Detail.asp File

This file is used to show the detail information about an artwork that was selected by a user. It gets a parameter--objectNumber, which was posted by the mlist.asp or mlistadv.asp page after a user clicked a link in the third window. Based on the value of this parameter, a query is constructed in SQL and issued to retrieve the detail information (title, artist, subject, image and description etc.) about this object. Then the results are presented in a table to the user.

7) reqForm.asp File

This file is used to implement the request form window. A one row and two columns (two cells) table was created for it. The left cell is filled by some information such as artist, title and objectnumber about the requested artwork. The right cell contains two forms. The first form is filled by several input text box, such as name, phone, email etc. which are used to collect user's information. The form's action attribute is set to 'sendemail.asp'. The second form just contains a 'cancel' submit button. The form's action attribute is set to 'detail.asp'.

8) sendemail.asp File

This file is to implement the sending email function. It constructs an email object, which has four paramers-- FromEmailAddress, ToEmailAddress, Subject, and Message. This file gets an objectnumber from 'reqForm.asp' file, which is used to build the message parameter. Finally, the note.htm is called, which just tells the user that the request has been received.

4.4 Search Tips for Advanced Text Search

1) Search Terms

- Consider all possible words or phrases that might be used to describe your desired artwork. These should include related terms; variations in word endings (e.g. singular, plural, adjectives); synonyms; variant terminology and alternative spelling forms.
- Choose specific search terms that are closely related to the interested artworks.
- Use terms you might use when discussing a topic with a curator or artist, including current jargon or buzzwords.
- The words should reflect ideas essential to your interest topic.
- Include alternative words and abbreviations
- Avoid words that are too general.

2) The Search Engine is not Case-Sensitive.

All words in the back-end database were lowercased and all input words are lowercased in the program, thus upper and lower case characters are interpreted as equivalents.

3) The Search Engine Does Not Search for Stop Words.

Stop words are common, frequently used words. While they may add clarity within the text, they do not add significant distinction to a search request. Since stop words have been

removed from the indexing terms, these words are not searchable and should be omitted from your search string.

Stop words include:

- Most articles (the, an, etc.)
- Personal pronouns (he, she, we, they, etc.)
- Most forms of the verb, "to be" (are, is, was, etc.)
- Some conjunctions (as, because, if, when, etc.)

4) Wildcard Characters
Use the wildcard character '%' to combine or eliminate search words to make the search simpler.

For example**:** *transplant%* would find transplant, transplanted, transplanter.

5) Special Characters Searching
When constructing a search request for words containing special characters, replace any special characters with the standard characters.

For example, accented characters should be entered as the same characters without the accent mark (even though the accent mark may appear on your keyboard). Such as, to search for the name, Fürst, enter: Furst

Search Strategy Checklist

1. Define the topic
2. Break it down into component concepts
3. Decide on the words/phrases to describe concepts

Think of possible alternatives:

- Terminology
- Spelling
- Synonyms

4. Decide on relationships
5. Try out the search
6. Display results
7. Refine search if necessary

BROADEN

- Put alternatives
- Reduce number of linked terms
- Change terminology

NARROW

- Link terms
- Add terms
- Change terminology

5. Summary

1) Research
First, Image database searching techniques have been researched and overviewed. After comparing the content-based and text-based searching methods and considering our real situation, the text-based search was selected.

Then research on text-based search techniques was conducted and summarized. Since the vector space model supports the relevance ranking technique much better than the other search algorithms, it was selected for the search engine.

Finally, research was conducted on the implementation of the vector space model. Six successively simpler approximations to the full vector space model were developed in the paper [26]. Based on the experimental result and our situation, the simplest approximation method (method 6) was selected.

2) Search Engine Implementation

An image database search engine has been designed and implemented.
It is a dynamic, data-driven, client/server software application.

The artworks were described by text strings, therefore searching a particular image is actually searching the corresponding text.

Several simple and two advanced search methods, and some simple and user-friendly interfaces were designed and implemented. For text search, Users can speak natural language to our search engine. The selected relevance ranking algorithm was implemented. The searching results are sorted by the relevance ranking score and displayed with the most relevant one floating on the top.

6. Limitations and Future Work

Limitations

- Users who are not familiar with the professional descriptors for artworks will have problems finding desired items precisely.
- The search engine only searches the exact words entered by users. It cannot identify the synonym and polysemy. It does not search for variations in word forms using stemming algorithms.
- The search engine does not have the ability to provide feedback to the user's query. In addition, the system does not remember the history of a query issued by users. The same query has to be re-entered for a repeated search.

Future Work

- Build Thesaurus Term List (Controlled Vocabulary) to make searching easier for the user.

  A Thesaurus Term List is a set of standard terms used to describe the contents of items found in a database. A term in a Thesaurus Term List may describe a person, an event, an idea, or a place. By putting together all items on the same topic under a single word or phrase, Thesaurus Term List can make searching for information much easier. Thesaurus Terms can help to get relevant information, and reduce the number of 'false hits!'

  See [41] for a good website about Art Thesaurus.

- Implement partial (heuristic search methods) search to improve searching performance.
  The basic idea of the partial methods is to process query terms one by one and accumulate partial scores for the documents, rather than compute the final score of a document completely before processing to the next documents [27].
- Consider efficiently combining text-based and content-based search together for a more powerful search.

- Consider GVSM (Generalized Vector Space Model) or SVSM (Semantic Vector Space Model) model instead of just VSM. Both GVSM and SVSM attempt to extend VSM by incorporating some contextual information in relevance prediction. While SVSM analyzes semantic case structure at the sentence level and encodes this information into its document/query representation, GVSM takes advantage of term co-occurrence patterns that are already captured in the document-term frequency matrix of VSM and builds the information into similarity computation [29].
- Consider using feedback information from a user to increase the effectiveness of the search

7. Source code (in a disk)
8. Acknowledgement

I am especially grateful to my graduate project advisor Curtis Meadow for offering a lot of very precious advice regarding my project source code and report. I thank Professor George Markowsky for providing important suggestions on the project presentation slides. I also thank Steve Ringle from Museum of Art for offering a lot of very good comments on the website designing. Finally, I would like to thank my husband Shuguo Ma and our lovely son DongZhe Ma for their support.

References

[1] D. S. Brandt, *What flavor is your internet search engine?*, Computers in Library, Vol. 17, No. 1, (1997) pp. 47-50.

[2] Robert S. Gray *"Content-based Image Retrieval: Color and Edges"*. IEEE Region 10's Annual International Conference, Proceedings. Proceedings of the 1994 IEEE Region 10's 9th Annual International Conference (TENCON'94). Part 1(2). Aug 22-26 1994, 1995,Singapore, Singapore.

[3] C. D. Rose, *Digital imaging*, http://charltonrose.com/school/digimg/

[4] H. Besser and J. Trant. *Introduction to imaging: Issues in Considering an image database.* The Getty Art History Information program, 1995.

[5] M.W. Berry , Large scale sparse singular value computations. *International Journal of Supercomputer Applications* **6** (1992), pp. 13–49.

[6] M.W. Berry, S.T. Dumais and G.W. O'Brien, Using linear algebra for intelligent information retrieval. *SIAM Review* **37** 4 (1994), pp. 573–595.

[7] Deerwester, S. Dumais, G.W. Furnas, T.K. Landauer and R. Harshman , Indexing by latent semantic analyses. *Journal of The American Society for Information Science* **41** 6 (1990), pp. 391–407.

[8] S. Dumais, G.W. Furnas, T. Landauer and S. Deerwester , Using latent semantic analysis to improve information retrieval. In: *Proceedings of CHI'88: Conference on human factors in computing*, ACM, New york (1988), pp. 285–288.

[9] Dumais, S. T. (1995). Using lsi for information filtering: TREC-3 experiments. In *Proceedings of the third text retrieval conference (TREC-3).*

[10] J.J. Rocchio , Relevant feedback in information retrieval. In: G. Salton, Editor, *The SMART retrieval system: Experiments in automatic document processing*, Prentice Hall, Englewoodcliffs, NJ (1971), pp. 313–323.

[11**]** Faloutsos, C.,Oard,D.(1995). *"A Survey of information retrieval and filtering methods"*,avail. As UMIACS-TR-95-33, College Park: U of MD

[12] S. K. Chang, E. junger and G. *Tortora, Spatial reasoning, image indexing and retrieval using symbolic projects*.

[13] A. K. L. Tang, *Content based image query*, Brigham Young University, Department of Computer Science. April 1996.

[14] S. Y. Lee and F. J. Hsu, *Spatial knowledge representation for iconic image database*.

[15] E. G. M. Petrakis and S. C. Orphanoudakis, *A generalized approach to image indexing and retrieval based on 2D strings*.

[16] J. C. R. Tseng, T. F. Hwang and W. P. Yang, *Efficient image retrieval algorithms for large spatial databases*.

[17] P. W. Huang and Y. R. Jean, *Reasoning about pictures and similarity retrieval for image information systems based on SK-SET knowledge representation*, Pattern Recognition, Vol. 28, No. 12, (1995) pp. 1915-1925,.

[18] C. E. Jacobs, A. Finkelstein and D. H. Salesin, *Fast multiresolution image querying*, Proceeding of the ACM SIGGRAPH 95: Computer Graphics Proceedings. 22nd Annual Conference Series, August 1995, pp. 277-286.

[19] P. W. Huang and Y. R. Jean, *Reasoning about pictures and similarity retrieval for image information systems based on SK-Set knowledge representation*, Pattern recognition (Pergamon), Vol. 28, No. 12, (1995).

[20] Kato T. *Database architecture for content-based image retrieval*. Proceedings of Society of the Photo-Optical Instrumentation Engineers: Image Storage and Retrieval, 1662, 1992. San Jose, California, USA, SPIE, 1992.

[21] Eakins, J. P. and Graham, M.E., *Content-Based Image Retrieval: A Report to the JISC Technology Applications Programme*. Institute for image Data Research: Newcastle-Upon-Type. 1999, pp.1-63.

[22] C. C. Venters and M. Cooper, *A Review of content-based image retrieval systems*, http://www.jtap.ac.uk/reports/htm/jtap-054.html.

[23] S. K. M. Wong, W. Ziarko, V. V. Raghavan and P. C. N. Wong, *On modeling of information retrieval concepts in vector spaces*, ACM Transactions on Database Systems, Vol. 12. No. 2, (1987) pp. 299-321.

[24] C. Y. Wang and P. A. NG, *A ranking-based document retrieval model: the TEXPROS approach*, Journal of Systems Integration, Vol. 8, (1998) pp. 379-404.

[25] G. Salton and M. J. McGill, *Introduction to modern information Retrieval.* McGraw-Hill: New York, 1983.

[26] Dik Lun Lee, Huei Chuang, and Kent Seamons, *"Document ranking and the Vector-Space Model",* IEEE Software, March/April 1997,P67-75.

[27] W. Y. P. Wong and D. L. Lee, *Implementations of partial document ranking using inverted files,* Information Processing & Management, Vol. 29, No. 5 (1993) pp. 647-669.

[28] S. A. Perry and P. Willett, *A review of the use of inverted files for best match searching in information retrieval systems,* Journal of Information Science, Vol. 6, (1983) pp. 59-66.

[29] G. Z. Liu, *Semantic vector space model: implementation and evaluation,* Journal of the American Society for Information Science, Vol. 48, No. 5, (1997) pp. 395-417.

[30] X. Tai, F. Ren and K. Kita, *An information retrieval model based on vector space method by supervised learning,* Information Processing and Management, Vol. 38 No. 6 (2001) pp. 749-764

[31] C. H. Chang and C. C. Hsu, *Enabling concept-based relevance feedback for information retrieval on the WWW,* IEEE Transactions on Knowledge and Data Engineering, Vol. 11, No. 4, (1999) pp. 595-609.

[32] H. Besser, *Visual access to visual images: the UC Berkeley image database project,* Library Trends, Vol. 38, No.4 (1990) pp. 787-798.

[33] W. R . Caid, S. T. Dumais and S. I. Gallant, *Learned vector-space models for document retrieval,* Infromation Processing & Management, Vol. 31. No. 3, (1995) pp. 419-429.

[34] H. Besser, *User interfaces for museums,* Visual Resources, Vol, VII, (1991) pp. 293-309

[35] Tat-Seng Chua, Swee-Kiew Lim, and Hung-Keng Pung. *"Content-based retrieval of images."* In Multimedia 94, pages 211-218, San Francisco, California, 1994.ACM.

[36] J. Griffioen, B. Seales and R. Yavatkar, *Content-based multimedia data management and efficient remote access,* http://www.uky.edu~kiernan/DL/brent.html.

[37] M. L. Larsgaard, *Content-based searching of large image databases,* http://www.uky.edu~kiernan/DL/larsgaard.html.

[38] H. Besser, *The transformation of the museum and the way it's perceived,* http://www-personal.si.umich.edu/~howardb/Papers/garmil-transform.html.

[39] B. Yumono, S. L. Y. Lam, J. H. Ying and D. L. Lee, *A world wide web resource discovery system,* http://www.w3.org/Conferences/WWW4/Papers/66.

[40] "Term Weighting and Ranking Algorithms" http://www.sims.berkeley.edu/courses/is202/f98/Lecture17/sld001.htm

[41] Art and Architecture Thesaurus On Line. See http://www.getty.edu/research/tools/vocabulary/aat/