Identifying Similar Learning Objects Incrementally

by

Naveen Nelapudi

B.S (Computer Science), Osmania University, Hyderabad, India, 2001

Submitted to the Department of Electrical Engineering and Computer Science and the faculty of the Graduate School of the University of Kansas in partial fulfillment of the requirements for the degree of Master of Science.

University of Kansas

Table of Contents

Abstract

There are thousands of colleges and universities offering common courses. Although courses may share elements in common it is rare to find two courses from two institutions that share the same, and only the same, set of elements. Identifying the common elements of similar courses, and having their descriptions in one centralized database so that they are available to every course on that subject, results in a significant reduction in both the development time and cost.

For learning objects to be more widely used, a wide variety of learning objects must become readily available and educators need to learn more about them, knowing how to search for them and create them. The project aims to solve the first problem by automatically identifying and retrieving the most similar learning objects to the user query. Incremental Indexing is used for building a similarity matrix as opposed to batch processing which required the algorithm to be re-run whenever a new document was added to the collection.

1. Introduction

1.1. Learning Objects

According to the Institute for Higher Education Policy [1] 85 percent of four-year colleges offer courses online. A problem with the online courses is that they are not very flexible so they are difficult to repurpose. Many institutions offer similar courses with similar elements and each one ends up developing content for the course. As a result, there are probably hundreds of descriptions of similar topics floating around on the Internet.

Educational content is not inexpensive to produce. Even a plain Web page authored by a professor can cost hundreds of dollars. When you include graphics and a little animation, the price is doubled. Add an interactive exercise and the price is quadrupled. Instead, a course can be made of smaller units of content called “learning objects”. Each learning object can be made such that it is self-contained so that it can be used without depending on other learning objects. These learning objects can be reused by the other courses that need to convey the same concept rather than requiring development of their own description. This can save both the time and the expense.

A learning object can be almost anything. Any stand alone piece of information capable of teaching something can be a learning object. It can be a chapter in a book, a video, an image, a wiring diagram, an interactive application, a simulation and so on. As well as being of a flexible type, a learning object can be any size. A learning object should be a self-contained, reusable, smaller unit of learning that can be aggregated with other learning objects to produce more substantial units of learning. They are generally tagged with meta-data to allow them to be easily retrieved by a search.

The availability of computational power and network infrastructure that greatly facilitate distribution and sharing of learning objects coupled with their flexibility and re-usability creates a compelling economic rationale for learning objects.

In order to make such a system distributed and interoperable, we need to make sure that there is a common language that different systems understand and communicate in. eXtensible Markup Language, or XML, developed by the World Wide Web Consortium seems like the obvious solution for its two main reasons. First, it is structured so it is capable of representing an object hierarchy. Second, it is in plain text and easily machine-readable. Thus, it provides a means of distributing content to other systems no matter where they are located and no matter what program they are running. Thus, a piece of learning material, no matter where it is located, may be seamlessly integrated into an online course, provided the XML tags are employed consistently.

1.2. IKME

The Intelligent Knowledge Management Environment (IKME) is an ongoing project at the University of Kansas aimed at assisting the Defense Information Technology Testbed (DITT)/University After Next (UAN) by providing an advanced reach-back capability for commanders, staff, and other users who have time-critical needs. A knowledge management environment would facilitate the creation of extensible and reusable learning objects that would lead to faster delivery of content to knowledge users.

The project is based on the idea of using the Extensible Markup language as the data format for publishing. Knowledge creators use the environment to create learning objects which are stored as XML documents. These learning objects are based on an XML schema developed by the “Center for Army Lessons Learned”. These learning objects can be in turn reused to create lesson objects, which in turn are used to create a manual. This setup facilitates ease of creation and also faster delivery of content to the end users. Another main advantage of using XML as the data format is the separation of content and style. The same XML document can be represented in various styles and data formats using style sheets. For example a manual can be published online using an XSLT style sheet and also converted to a PDF file (for printing) by using the XSL-FO style sheet.

IKME contains the following capabilities:

- Create, modify, view and search learning objects.
- Create, modify, view and search lesson objects.
- Create, modify, view and search manual objects.

1.3. Goals and Contributions

- To use a Memory based approach for indexing as opposed to File based approach used for the earlier version.
- To help the user find the related learning object content for the creation of lesson objects and manuals.
- To incorporate Incremental Indexing into the similarity search instead of batch processing which required the algorithm to be re-run every time a new document is added to the collection.

2. Approach

2.1. Overview

The aim of the project is to generate similarity information between different documents so that the system can display the top *N* matching documents for a given document. Initially, each document is preprocessed and tokenized according to a set of predefined rules. Then these tokenized documents are indexed using the standard vector space model. As, a result Dictionary and Postings Files are created. Then the similarities between each document with all other documents in the collection are calculated. These results are stored as a similarity matrix on the disk for future use (when a new document is added). Then the top *N* similar documents to each document can be easily retrieved by a file look up operation. When new documents get added to the collection by create learning objects, the index and the similarity matrix get updated so that the changes to the similarities because of new documents are immediately reflected.

2.1. Existing Version

The existing version of IKME uses a file based indexing method in which each file is tokenized into an output file. So, if the document collection has 50 files, (1.in, 2.in,…., 50.in), we create 50 tokenized files (1.out, 2.out,…., 50.out). When the similarities are calculated, each tokenized file is opened, used for similarity calculation and closed. Once the similarity matrix is generated, it is stored to a persistent store and this matrix is used to display the top *N* most similar documents to a particular document.

Formula:

$$Similarity(d1, d2) = \sum_{i=1}^{N} wt_{id1} * wt_{id2}$$

where

N = number of tokens in the vocabulary

$wt_{id1} = tf_{id1} * idf_i$

tf_{id1}= *(frequency of token i in d1/ total number of unique tokens in d1)*

$idf_i = \log_2$ *(total number of documents/number of documents in which token i appeared)*

2.2. Problems with the existing version

The above method suffers from three problems. First, it uses a File based Indexing which is not fast enough for large scale processing. Second, it creates many files for calculating the similarity information. Finally the most glaring problem of all is that whenever a new document is added to the collection the whole algorithm needs to be re-run. So, it creates a similarity matrix from the scratch after each document addition. Since the algorithm is N^2, the problem

2.3. Overview of my enhancements

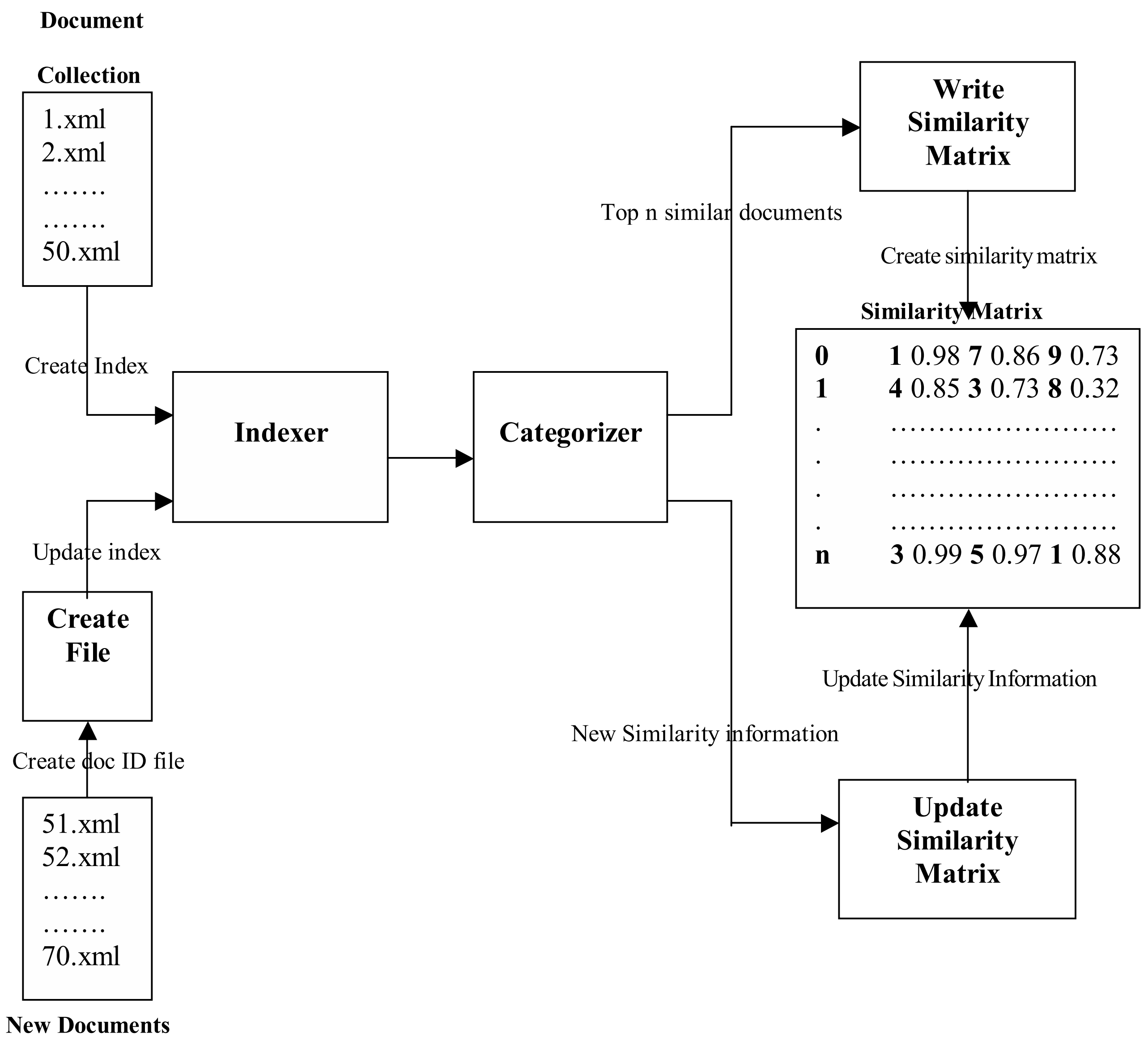
The new version uses a Memory based approach as opposed to file based approach, so the process of indexing is faster than that of the previous version. Since memory is getting cheaper, we should have enough space to fit the inverted file in main memory and take the advantage of faster seek times.

Incremental Indexing is used, so even if a new document is added to the collection, the algorithm need not be run again from the scratch. It first updates the index files (Postings and

Dictionary) and then uses the existing similarity information for updating the similarity matrix because of updated index.

Execution speed is the most important advantage gained by the enhancements which make it more suitable for large scale needs.

3. System Architecture



System Architecture

4. **Implementation Details**

4.1 Index.cc:

Inputs:

- Path of the dictionary file to be created.
- Path of the postings file to be created.
- Path of the documents file to be crated.
- Path to file specifying document pre-processing options.
- Directory containing the documents to be indexed/Path to file that specifies the filenames and document ids.
- Number of words expected in the document collection.
- Number of documents.
- Flag to turn on/off normalization.
- Flag to print indexing execution statistics.
- Flag to specify whether to read from the list of document name or to generate the file.

Outputs:

- Dictionary File
- Postings File
- Documents File

Purpose: To index the documents using standard term frequency-inverse document frequency (tf-idf) approach.

4.2 Categorize.cc:

Inputs:

- Path of the dictionary file.
- Path of the postings file.
- Path of the documents file.
- Path of the input document to be categorized.
- Output file containing category ids and corresponding weights.
- Number of words to be used for comparing documents.
- Number of documents to be returned.

Outputs:

- Returns a file that contains top n similar documents to the current document along with the weights.

Its format is <docid> <weight>

Purpose: Returns the top n similar documents to a given document in the descending order.

Sample Output:

Doc Id	Weight
10	0.98
8	0.85
6	0.75
3	0.43
11	0.12

4.3 WriteSim.cc:

Inputs:

- The document collection.
- The path where the resultant similarity matrix should be written.

Outputs:

- The similarity matrix is created and is stored as a file. It has all the documents in the collection and the top 5 matching documents to the corresponding documents.

Purpose: This program is responsible for creation of the similarity file for the purpose of retrieving top n similar objects to a given document. The similarity matrix is a flat file with fixed record format, so that retrieval of a particular record can be done with ease.

Algorithm:

```
For all documents in the collection {
        Process the document for tokenizing it.
        Add the tokenized document to the index (Dictionary, Postings and Documents files).
        Retrieve the top 'n' similar documents to the current document.
        Eliminate the similarity value of a document with itself.
        Open the temporary output file in Read Mode.
        Open the similarity file in Write Mode.
        Print the top n results from temporary file to the Similarity Matrix.
        Close the temporary file.
        Close the similarity file.
}
Close the Directory of documents.
```

Sample Output:

Document	Id	Weight	Id	Weight	Id	Weight	Id	Weight	Id	Weight
0	1	0.2213	13	0.2122	35	0.0632	54	0.0570	4	0.0529
1	0	0.5287	13	0.4383	7	0.2250	25	0.1844	21	0.1839
2	52	0.1439	17	0.1351	14	0.1057	13	0.0987	12	0.0812
3	57	0.2211	50	0.1459	55	0.1385	7	0.1305	51	0.0979

4.4 CreateFile.cc:

Inputs:

- New documents added to the document collection.
- The Similarity Matrix file.

Outputs:

- A file with the document names and the ids in the new directory.

Purpose: The resultant docID file has the document names and the docID (starting from the last document id in the docID file created by indexer). It is needed for updating the index files with the new documents.

Algorithm:

Open the Similarity Matrix in read mode.

Open the docID file in write mode.

```
Seek till the end of Similarity File to find the last document id.
Open the directory or newly added documents.
For all the documents in the directory {
    Print the location of the file and the incremented document id into the docID file.
}
Close the directory.
Close the docID file.
Close the Similarity File.
```

Sample Output:

Doc Name	Doc Id
../inputfiles/urban_energy.xml	51
../inputfiles/urban_environment.xml	52
../inputfiles/urban_finance.xml	53
../inputfiles/urban_infrastructure.xml	54
../inputfiles/urban_population.xml	55
../inputfiles/urban_network.xml	56
../inputfiles/urban_satellite.xml	57
../inputfiles/urban_segments.xml	58
../inputfiles/urban_supersurface.xml	59
../inputfiles/urban_systems.xml	60

4.5 Update.cc:

Inputs:

- The location of Similarity Matrix.
- New documents added to the document collection.

Outputs:

- Updated Similarity Matrix with corresponding changes made to all the corresponding documents.

Purpose: Update the Similarity Information, because of the newly added documents to the collection.

Algorithm:

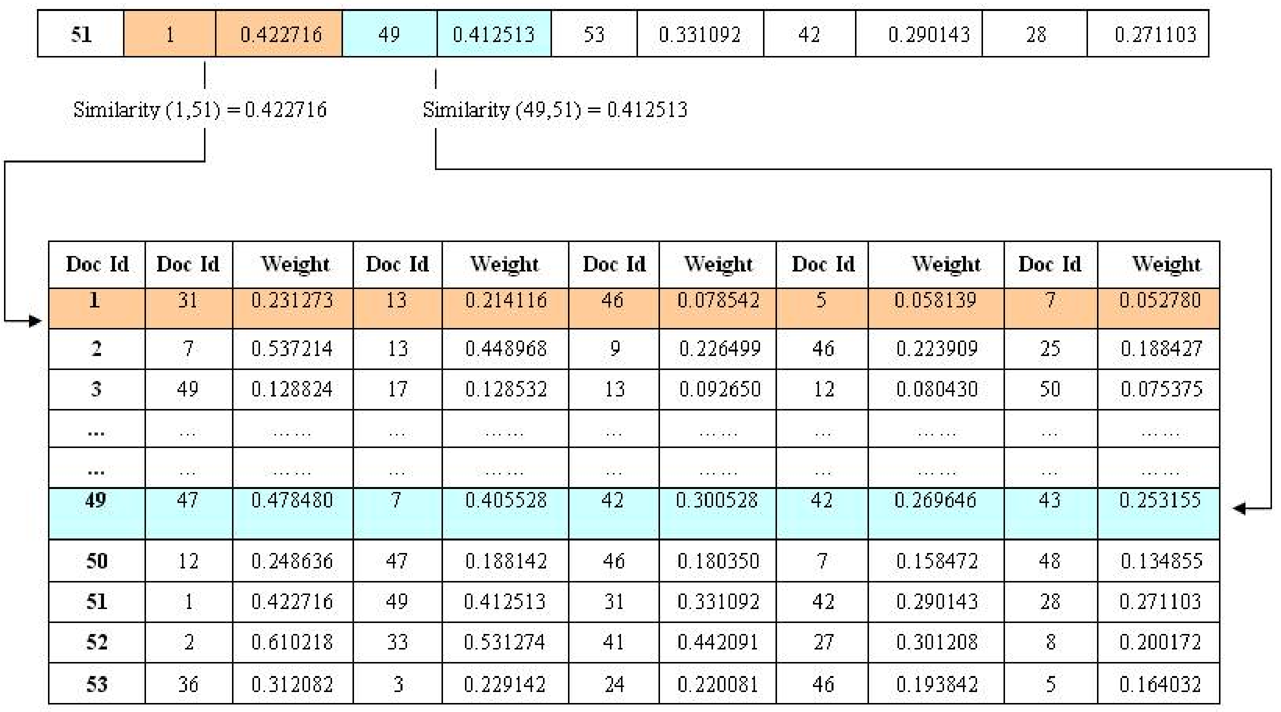
```
typedef struct {
        int id;
        float wt;
} recordtype;
recordtype UPDATERECORD [UPDATERECORDSIZE];
recordtype RECORD [RECORDSIZE];


Open the Similarity Matrix in read and write mode.
Seek till the end to find the last document id and save it in the memory.
For all the files in the newly added directory {
        Process the document for tokenizing it.
        Retrieve the top ‘n’ similar documents to the current document.
        Open the output stored in the temporary file in Read Mode.
        Open the similarity file in Write Mode.
        Print the top n results from temporary file along with the document id, to the Similarity
        Matrix.
        Copy the top 3n results into the UPDATERECORD Array of doc_ids and weights.
        Close the temporary file.
        For all the elements in UPDATERECORD (i=0 to 3n) {
            If the doc id of the ith element of UPDATERECORD is greater than or equal to
            current document id
                Continue
            Seek to the start of the record of the corresponding document according to the ith
            location in the UPDATERECORD Array.
            Store the record entries and the current docid and weight into the RECORD array.
            Perform Insertion Sort on the weights of the 6 entries in RECORD array.
            Seek to the start of that record again and write the top 5 entries of the sorted
            Array.
            Close the temporary file.
        }
    Close the similarity file.
    }
```

Step 1: Adding new record(s) to the similarity matrix

Doc Id	Doc Id	Weight	Doc Id	Weight	Doc Id	Weight	Doc Id	Weight	Doc Id	Weight
1	31	0.231273	13	0.214116	46	0.078542	5	0.058139	7	0.052780
2	7	0.537214	13	0.448968	9	0.226499	46	0.223909	25	0.188427
3	49	0.128824	17	0.128532	13	0.092650	12	0.080430	50	0.075375
...	...	......	...	......	...	......	...	......	...	......
...	...	......	...	......	...	......	...	......	...	......
49	47	0.478480	7	0.405528	42	0.300528	42	0.269646	43	0.253155
50	12	0.248636	47	0.188142	46	0.180350	7	0.158472	48	0.134855
51	1	0.422716	49	0.412513	31	0.331092	42	0.290143	28	0.271103
52	2	0.610218	33	0.531274	41	0.442091	27	0.301208	8	0.200172
53	36	0.312082	3	0.229142	24	0.220081	46	0.193842	5	0.164032

Step 2: Updating the corresponding records' similarity information



51	1	0.422716	49	0.412513	53	0.331092	42	0.290143	28	0.271103

Doc Id	Doc Id	Weight	Doc Id	Weight	Doc Id	Weight	Doc Id	Weight	Doc Id	Weight
1	31	0.231273	13	0.214116	46	0.078542	5	0.058139	7	0.052780
2	7	0.537214	13	0.448968	9	0.226499	46	0.223909	25	0.188427
3	49	0.128824	17	0.128532	13	0.092650	12	0.080430	50	0.075375
...	…	……	…	……	…	……	…	……	…	……
...	…	……	…	……	…	……	…	……	…	……
49	47	0.478480	7	0.405528	42	0.300528	42	0.269646	43	0.253155
50	12	0.248636	47	0.188142	46	0.180350	7	0.158472	48	0.134855
51	1	0.422716	49	0.412513	31	0.331092	42	0.290143	28	0.271103
52	2	0.610218	33	0.531274	41	0.442091	27	0.301208	8	0.200172
53	36	0.312082	3	0.229142	24	0.220081	46	0.193842	5	0.164032

OLD RECORD OF DOC 1

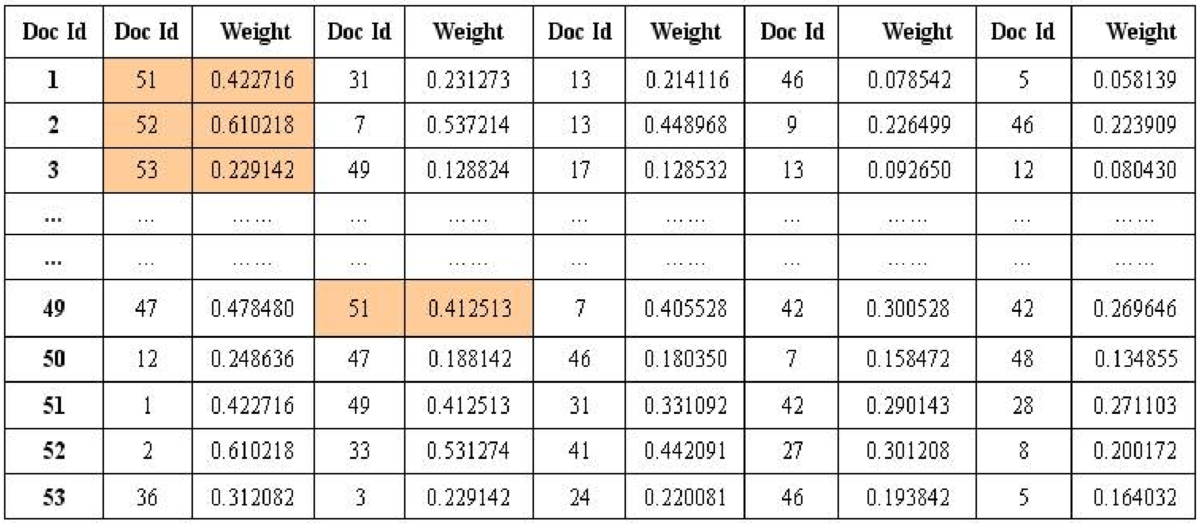
Doc Id	Doc Id	Weight	Doc Id	Weight	Doc Id	Weight	Doc Id	Weight	Doc Id	Weight
1	31	0.231273	13	0.214116	46	0.078542	5	0.058139	7	0.052780

With the addition of new document 51 with Similarity (1,51) = **0.422716**

NEW RECORD OF DOC 1

Doc Id	Doc Id	Weight	Doc Id	Weight	Doc Id	Weight	Doc Id	Weight	Doc Id	Weight
1	51	0.422716	31	0.231273	13	0.214116	46	0.078542	5	0.058139

New Similarity Matrix



Doc Id	Doc Id	Weight	Doc Id	Weight	Doc Id	Weight	Doc Id	Weight	Doc Id	Weight
1	51	0.422716	31	0.231273	13	0.214116	46	0.078542	5	0.058139
2	52	0.610218	7	0.537214	13	0.448968	9	0.226499	46	0.223909
3	53	0.229142	49	0.128824	17	0.128532	13	0.092650	12	0.080430
...	...	……	...	……	...	……	...	……	...	……
...	...	……	...	……	...	……	...	……	...	……
49	47	0.478480	51	0.412513	7	0.405528	42	0.300528	42	0.269646
50	12	0.248636	47	0.188142	46	0.180350	7	0.158472	48	0.134855
51	1	0.422716	49	0.412513	31	0.331092	42	0.290143	28	0.271103
52	2	0.610218	33	0.531274	41	0.442091	27	0.301208	8	0.200172
53	36	0.312082	3	0.229142	24	0.220081	46	0.193842	5	0.164032

5. Evaluation

Comparisons between the existing version and the new version when new documents are added to the collection:

Number of Documents	**New Version** Time (in seconds)	**Earlier Version** Time (in seconds)
54	26 (7.31u, 12.28s)	52
72 (18 documents added)	11 (2.48u, 3.98s)	62
172 (100 documents added)	52 (14.13u, 23.10s)	220

For 500 documents (450 added to an initial collection of 50), it took around 243 seconds (68.34u, 124.36s) for the new version.

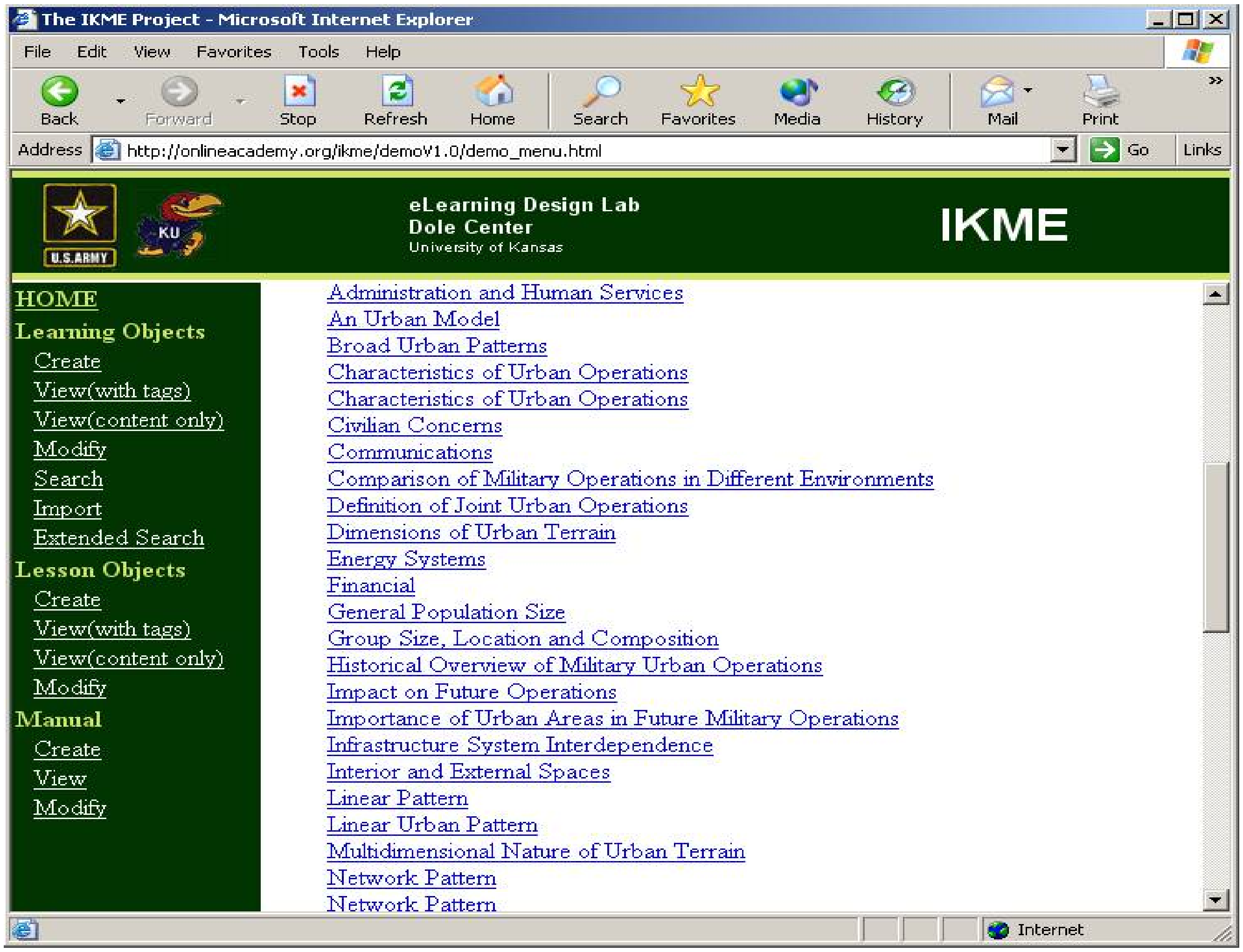
The execution speeds stand as a testimony for the efficiency and the need for Incremental Indexing in real-time searches for learning objects stored in huge databases.

Comparisons between the existing version and the new version when new documents are added to the collection but the similarity information is calculated at once, instead of using the incremental update:

Number of Documents	**New Version** Time (in seconds)	**Earlier Version** Time (in seconds)
54	26 (6.98u, 12.09s)	52
72	37 (9.418u, 15.85 s)	62
172	91 (24.21u, 38.09s)	220

Clearly the usage of memory based method is an improvement over the file based method. The decrease in the execution time validates the efficiency of new version.

6. Screenshots


The IKME Project - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites Media History Mail Print
Address http://onlineacademy.org/ikme/demoV1.0/demo_menu.html
Go Links
U.S.ARMY
KU
eLearning Design Lab
Dole Center
University of Kansas
IKME
Internet

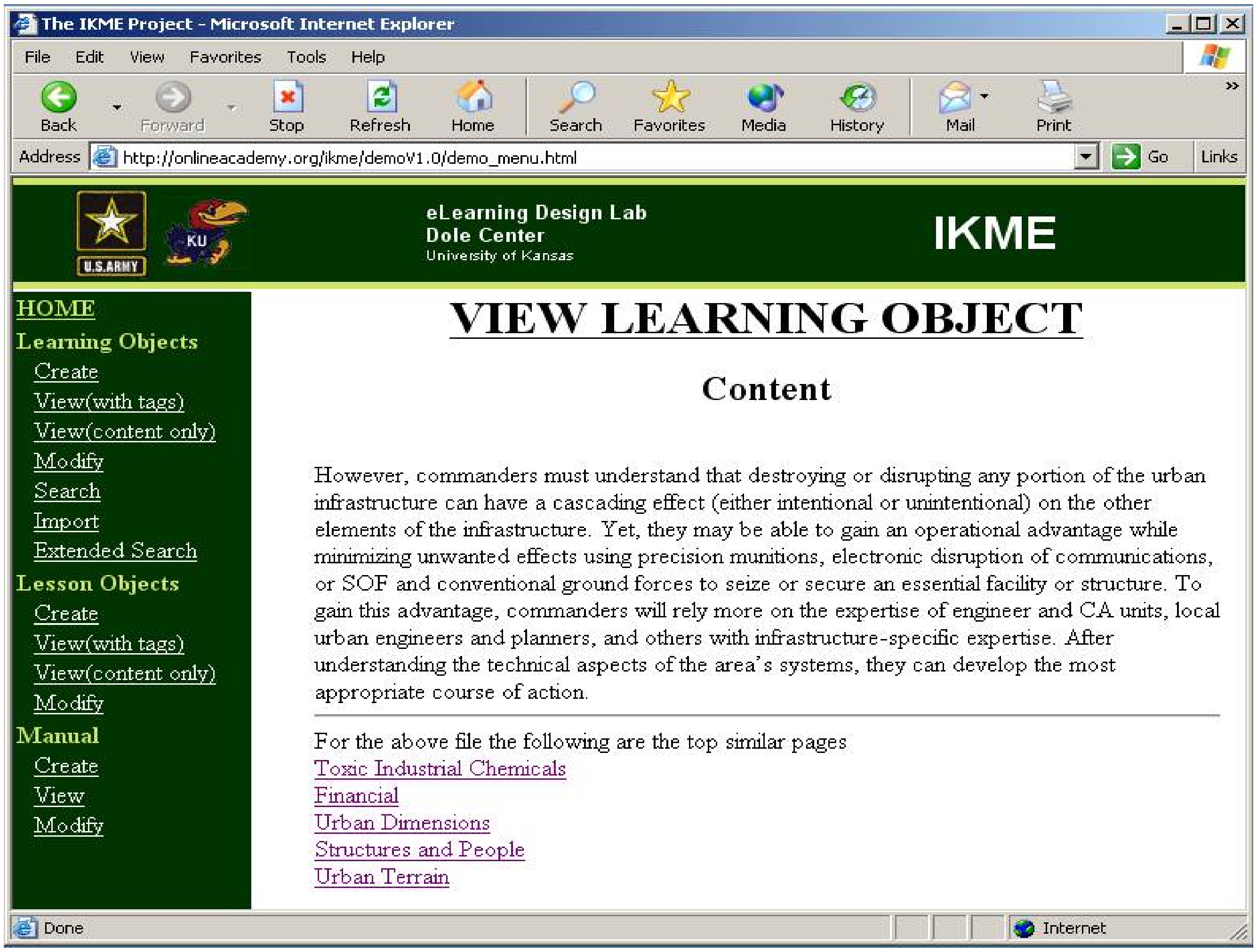
HOME
Learning Objects
- Create
- View(with tags)
- View(content only)
- Modify
- Search
- Import
- Extended Search

Lesson Objects
- Create
- View(with tags)
- View(content only)
- Modify

Manual
- Create
- View
- Modify

Administration and Human Services
An Urban Model
Broad Urban Patterns
Characteristics of Urban Operations
Characteristics of Urban Operations
Civilian Concerns
Communications
Comparison of Military Operations in Different Environments
Definition of Joint Urban Operations
Dimensions of Urban Terrain
Energy Systems
Financial
General Population Size
Group Size, Location and Composition
Historical Overview of Military Urban Operations
Impact on Future Operations
Importance of Urban Areas in Future Military Operations
Infrastructure System Interdependence
Interior and External Spaces
Linear Pattern
Linear Urban Pattern
Multidimensional Nature of Urban Terrain
Network Pattern
Network Pattern

List of Learning Objects stored in the repository.



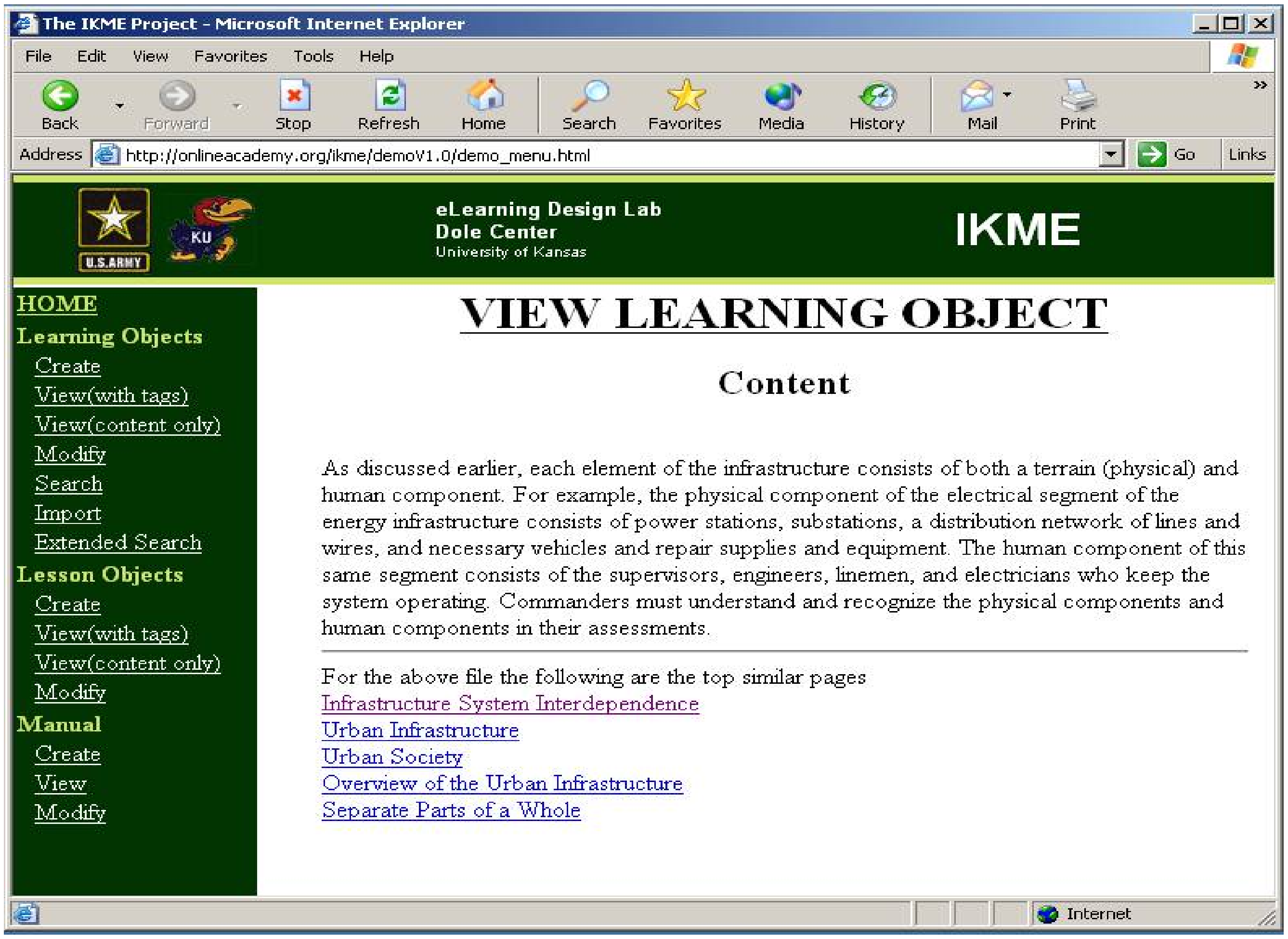
VIEW LEARNING OBJECT

Content

However, commanders must understand that destroying or disrupting any portion of the urban infrastructure can have a cascading effect (either intentional or unintentional) on the other elements of the infrastructure. Yet, they may be able to gain an operational advantage while minimizing unwanted effects using precision munitions, electronic disruption of communications, or SOF and conventional ground forces to seize or secure an essential facility or structure. To gain this advantage, commanders will rely more on the expertise of engineer and CA units, local urban engineers and planners, and others with infrastructure-specific expertise. After understanding the technical aspects of the area's systems, they can develop the most appropriate course of action.

For the above file the following are the top similar pages
Toxic Industrial Chemicals
Financial
Urban Dimensions
Structures and People
Urban Terrain

Top 5 similar learning objects to a particular learning object



VIEW LEARNING OBJECT

Content

As discussed earlier, each element of the infrastructure consists of both a terrain (physical) and human component. For example, the physical component of the electrical segment of the energy infrastructure consists of power stations, substations, a distribution network of lines and wires, and necessary vehicles and repair supplies and equipment. The human component of this same segment consists of the supervisors, engineers, linemen, and electricians who keep the system operating. Commanders must understand and recognize the physical components and human components in their assessments.

For the above file the following are the top similar pages
Infrastructure System Interdependence
Urban Infrastructure
Urban Society
Overview of the Urban Infrastructure
Separate Parts of a Whole

Top similar learning object to a particular object

7. Conclusions

- The primary goal of incorporating Incremental Indexing into the similarity search has been achieved.
- The algorithm need not be re-run even if a single document is added to the collection. Only the required parts of the index and the similarity matrix are updated.
- This will provide a faster way to search for similar learning objects and help the educators in creating new lessons using existing learning objects rapidly and inexpensively.

8. Future Work

- Investigating similarity formula (i.e., weighting different fields differently when calculating the match between the objects)
- Learning best differential weighting scheme.

9. References

[1] **All About Learning Objects.**

http://www.eduworks.com/LOTT/tutorial/learningobjects.html

[2] **Learning Objects 101: A Primer for Neophytes**

http://online.bcit.ca/sidebars/02november/inside-out-1.htm

[3] **Introducing Reusable Learning Objects**

http://media.wiley.com/product_data/excerpt/56/07879649/0787964956.pdf

[4] “**Automatically Identifying Related Learning Objects**” Mahesh Vulpala, Masters Project. University of Kansas 2003.