Automated building of domain ontologies from lecture notes in courseware

Neelamadhav Gantayat
Department of Computer Science and Engineering
Indian Institute of Technology Bombay
Mumbai, India
Email: neelamadhavg@cse.iitb.ac.in

Sridhar Iyer
Department of Computer Science and Engineering
Indian Institute of Technology Bombay
Mumbai, India
Email: sri@it.iitb.ac.in

***Abstract*—Courseware repositories contain large amounts of lecture videos and text. When searching for lecture material on a given topic, it would be useful if the repository also indicates the topics that are pre-requisites. In this paper we present a technique that automatically constructs the ontology (dependency graph) from the given lecture notes. We show how this ontology can be used to identify the pre-requisites and follow-up modules for a given query (lecture topic). We also provide the user with a dependency graph which gives a conceptual view of the domain. Our system extracts the concepts using term frequency inverse document frequency (tf-idf) weighting scheme and then determines the associations among concepts using apriori algorithm. We have evaluated our system by comparing its results with the dependencies determined by an expert in the subject area.**

***Keywords*-Ontology; Tf-idf; Apriori Algorithm; Pre-requisites; Follow-ups; Dependency Graph.**

I. Introduction

Courseware repositories, such as OCW[1] and NPTEL[2], contain large amounts of data in the form of videos and text. A fine-grain (topic-level) search facility and automatic identification of pre-requisites and follow-ups for a given topic is desirable and would be useful to students. Such a feature (identification of pre-requisites of a given topic) is not available in these repositories. This feature could be built by manual tagging of the contents, but it is cumbersome to do so.

In this paper we present a technique that automatically constructs the ontology (dependency graph) from given lecture notes. We show how this ontology can be used to identify the pre-requisites and follow-up modules for a given query (lecture topic). In domain ontology, relationships between different concepts of a domain are identified. In our case, a concept corresponds to a lecture module and a relationship corresponds to whether it is a prerequisite or a follow-up of the topic. We also provide the user with a dependency graph which corresponds to a concept map and gives a conceptual view of the domain. People can often grasp ideas much more quickly by looking into the graphical representation than by reading them in a book [1].

Our system extracts the concepts using term frequency inverse document frequency (tf-idf) weighting scheme and then determines the associations among concepts using apriori algorithm. We have evaluated our system by comparing its results with the dependencies determined by an expert in the subject area.

Our technique is to extract the topics (keywords) from the given PDF files using "term frequency inverse document frequency (tf-idf) weighting scheme" (described in Section 3). Then we determine the associations among different concepts (topics) using "apriori algorithm" (described in Section 3). Then we arrange the relations in a hierarchical order. The implementation of our system is described in Section 4. For any user query, our system provides the link for the topic, and two topics above it as pre-requisites and two topics below it as follow-ups, from the hierarchy of the ontology. Our experiments to evaluate the performance of the system are shown in Section 5 and conclusions in Section 6. To the best of our knowledge, there is no such system to automatically determine dependencies of topics from a repository of lecture notes.

II. Background

This section describes courseware repositories, gives a brief overview of domain ontology and defines dependency graph.

A. Courseware Repositories

We surveyed OCW and NPTEL repositories and searched for topics like Threads, TCP/IP and Ethernet. Although these topics are covered in these repositories, the search results often gave links to more advanced topics than our desired topic. We believe that a user not already familiar with the domain would find it difficult to determine the sequence of links to follow.

1) OCW: MIT OpenCourseWare is an initiative by MIT and has over 2,000 courses in 36 academic disciplines [2]. This content is available from http://ocw.mit.edu/. A lot of the content is in the form of PDF documents. We used the existing search facility for the topic "Operating system Threads". The first link of the results was "Micro-kernels", which is an advanced topic and does not help a user who is not familiar with the domain.

2) NPTEL: The National Programme on Technology Enhanced Learning (NPTEL) is a project from the Ministry of Human Resource Development (MHRD), India, initiated in 1999 [3]. There are more than 260 courses, available in two modes: Some courses have video lectures,

[1]http://ocw.mit.edu
[2]http://nptel.iitm.ac.in

while others have lectures notes in the form of PDF files. We used the existing search facility at NPTEL and found that it is course-grain and limited to only course titles. Currently it does not support topic-level search.

Hence we believe that a facility to not only support accurate topic-level search but also to identify pre-requisites and follow-ups of a topic would be useful. In the next section we provide the background on domain ontology that forms the basis of the techniques used in our system.

B. Ontology

Ontology provides a mechanism to capture information about the ideas, concepts, and the relationships between them in some domain [4]. The aim of ontology is to develop knowledge representations that can be shared and reused. Guber [5] defined ontology as

"A formal explicit specification of a shared conceptualization."

Domain ontology provides particular meanings of terms as they apply to that domain. For example the word thread has many different meanings. An ontology for the domain of operating system would model process threads, while an ontology for the domain of textiles would model thread as "long object resembling a thin line". Main application areas of ontology are knowledge management, web commerce, electronic business, and e-learning [6].

The key difficulties in developing ontology are: (i) extensive knowledge about a subject is required and (ii) it is time-consuming. We have automated this process, in the context of lecture notes. We use domain ontology to represent relations between topics for a given course. Here we consider only one relation, that is *"follows"*. Topic-2 *follows* Topic-1 means that Topic-1 is a pre-requisite for Topic-2 and Topic-2 is a follow-up of Topic-1. In our system we first develop the domain ontology from the given set of notes. Then we refer the node which represents the user's desired topic and also provide two of its ancestor nodes as pre-requisites and two descendants as follow-ups.

The domain ontology developed by our system is also presented to the user by a graphical representation called dependency graph.

C. Dependency graph

A dependency graph is a directed graph which represents dependencies of several objects towards each other.

"Given a set of objects S and a transitive relation $R = S \times S$ with $(a, b) \in R$ modeling a dependency 'a needs b evaluated first', the dependency graph is a graph $G = (S, T)$ with $T \subseteq R$ and R being the transitive closure of T." [7]

Dependency graphs are represented in hierarchical order, i.e., most general concepts are at the top of the graph and the more specific and less general concepts in lower orders. Using dependency graphs we can represent the dependencies between different concepts as shown in Figure 1; concepts are shown by ellipses and dependencies by arrows.

A dependency graph being similar to a concept-map [1], enhances the learner's understanding of a given subject

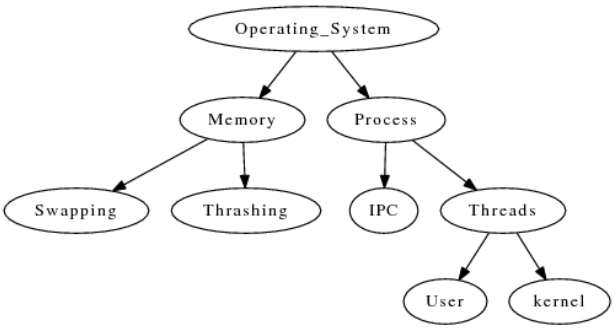

Figure 1. Dependency Graph

and is useful for providing summary of various interconnected and dependent topics. The key difference between a dependency graph and a concept-map is that: a concept-map can have any relation between two concepts, whereas in a dependency graph there is only one relation, that is, *depends*.

III. System Overview

A. Problem Statement

Given a repository of lecture notes for a particular subject (or the soft-copy of a text book), our aim is to build a system that provides the user with: (i) a mechanism to query for a desired topic, (ii) identify the pre-requisites and follow-ups for the topic and (iii) a dependency graph of the pre-requisites and follow-ups of all topics in the subject.

B. Solution Outline

The steps for our solution are as follows and also shown in Figure 2:

1) Convert the given PDF files into text, and index the text files.
2) Extract the keywords from the text files.
3) *For each keyword*: Identify its relation with other keywords. If there is a relation, then determine whether it is a pre-requisite or a follow-up.
4) Construct and store the ontology with the identified relations, in the form of a dependency graph in a hierarchical order.
5) Given a user query, lookup the ontology and provide the pre-requisites (ancestors) and follow-ups (descendants), as a reply to the query.

IV. Implementation Details

The implementation of our system has the following five phases:

1) Parsing: Parse the PDF files of the course to get the corpus text files, using PDFBox [8].
2) Indexing: Index the Text files, using Lucene [9].
3) Keyword Extraction: Extract the keywords, using tf-idf weighting scheme [10].
4) Ontology Construction: Find the relations between the keywords, using the apriori algorithm [11].

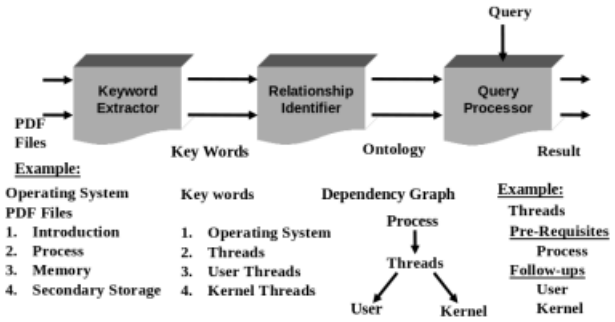

Figure 2. System overview

5) Dependency Graph Generation: Generate the dependency graph for the whole course, using DOT [12].

The flow of input to output is shown in Figure 3 and the details are described in the subsequent sections.

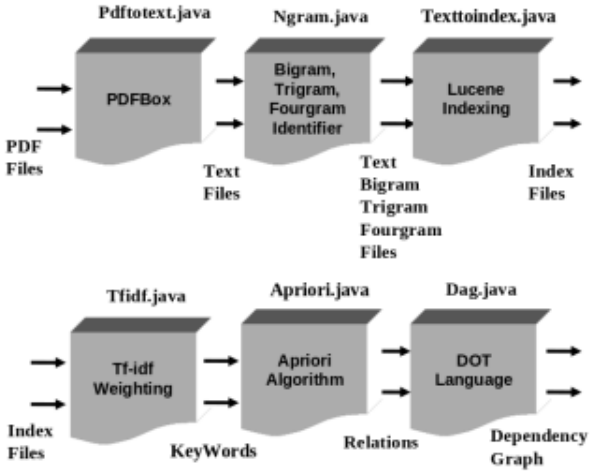

Figure 3. System Design

A. Parsing

We made use of a Nutch utility called PDFbox [8] to parse the PDF files and to convert them into text (as required by our indexing utility, Lucene). In case there is only one big PDF file containing all the topics (such as soft-copy of a book), we used PDFBox also to divide it into multiple PDF files, since our algorithm requires multiple PDF files for identifying relations.

B. Indexing

Lucene [9] is embedded in our system to index and search the given text documents. It lets one add indexing and searching capabilities to the application. Lucene can index any data that can be converted to textual format, and make it searchable. We used Lucene index to search and to calculate the tf-idf weight of each term (as described below), and also to identify the relationship between two keywords, i.e., whether one is a pre-requisite for other or a follow-up to the other.

C. Keyword Extraction

Keyword extraction is the process of extracting important phrases which can summarize the meaning of a document. Keywords can be extracted using linguistic techniques or machine learning techniques. Linguistic techniques make use of part-of-speech tagging or phrase chunking [13]. On the other hand, machine learning techniques use statistical or probabilistic data for keyword extraction. Machine learning based keyword extraction is once again divided into supervised and unsupervised techniques. Supervised techniques require some data for which the keywords are known (this is called training data) for its operation, while unsupervised techniques do not require any training data.

We used tf-idf (an unsupervised technique), to identify terms with high relevance to the document. We used "topia" [14] (which makes use of part-of-speech (POS) tagging technique), to find other keywords which are missed by tf-idf algorithm, if any.

1) Term Frequency Inverse Document Frequency (tf-idf) [10]: Term frequency inverse document frequency (tf-idf) is defined as the number of occurrences of a term in a given document multiplied by the inverse of the number of documents where the term appears. Given a document collection D, a word w, and an individual document $d \in D$

$$w_d = f_{w,d} * log(|D|/f_{w,D})$$

where

- $f_{w,d}$ equals the number of times w appears in d,
- $|D|$ is the size of the corpus (number of documents), and
- $f_{w,D}$ equals the number of documents in which w appears in D.

Logarithm of document frequency in the above formula is used for smoothing purpose. The tf-idf value is

- high when a term occurs many times within a small number of documents,
- low when the term occurs fewer times in a document, or occurs in many documents,
- lower when the term occurs in virtually all documents.

Using tf-idf weighting scheme, we scored each word (unigram, bigram, trigram, and fourgram) in the given text corpus. Then we found out the top unigrams, bigrams, trigrams, and fourgrams according to the tf-idf weights. *Ngrams* are groups of n written letters, n syllables, or n words. In this way, we extracted top keywords for the given text.

We defined our own set of stop words in order to increase the accuracy of the extracted keywords. Stop words are common words like numbers, digits, articulations, conjunctions etc. We did not allow any stop words in bigrams. We allowed stop words as a second word for conjunction purpose in trigrams. In fourgrams, the third and the fourth words can be stop words but the first and the last words cannot be stop words. All the keywords extracted are stored in a file called `tfidf.txt`.

We performed experiments to determine the optimal number of unigrams, bigrams, trigrams, fourgrams, and total number of keywords to include. These are discussed in the evaluation section.

D. Ontology Construction

To identify the relation between different keywords and to construct the ontology, we used apriori algorithm, a classical algorithm for learning association rules.

1) Apriori Algorithm: "Given a set of documents, generate all association rules that have support and confidence greater than the user-specified minimum support and minimum confidence respectively" [11]. Where *Association Rule:* $i_1, i_2, ..., i_k \rightarrow j$ means, "if a document contains all of $i_1, ..., i_k$ then it is likely to contain j". *Support* is the number of documents containing all the words $i_1, i_2, ..., i_k, j$ and *Confidence* of this association rule is the probability of j given $i_1, ..., i_k$.

We modified the apriori algorithm according to our requirement. Our modified version is as follows:

- **Step 1:** Read documents and count the occurrences of each item. It requires only memory proportional to the number of words in the documents. This step was modified by using tf-idf weight.
- **Step 2:** Read documents again and count only those pairs which were found in Step 1 to be frequent. So if we find n frequent keywords, then we will have $n(n-1)/2$ pairs. Now, find out the *frequent wordsets* with the given confidence. The *frequent wordsets* establish an *Association* between the two words.

The *support* is a configurable parameter. If the support is less (minimum number of documents in which the pair of words should be repeated), then the algorithm will find more number of relations. On the other hand, if the support is more then, the algorithm will identify less number of relations. We considered the confidence as 0%. That is, if the support is satisfied, then we consider it as a relation, though other documents may contain only word-1 but not word-2.

For each pair of keywords in `tfidf.txt`, we calculated the frequency among different documents, and listed the pair of words having greater frequency than the support provided as relations. All the relations are stored in a file called `relation.dot` in a form which can be read by DOT language.

E. Generating the Dependency Graph

Having found the relations between the keywords, we show them in a graphical representation, i.e., the dependency graph. The purpose of creating the dependency graph is to let the users decide what they should know before learning any specific topic. Using DOT language, we constructed a directed acyclic graph for the relations that we identified above, and converted the file `relation.dot` into `dag.pdf` which is the dependency graph.

V. Evaluation

We used Computer Networks and Operating Systems courses of NPTEL courseware repository for our testing purpose. The dependency graph for Computer Networks course is shown in the Figure 4. It has been generated using DOT language. DOT language orders the nodes in such a way that, node with less number of incoming edges comes at the top levels, and nodes with more number of incoming edges in the next levels. So, the order shown in the graph is not the exact output of our system, it is a graphical simulation of the output. In the dependency graph shown in Figure 4, concepts like *tcp, ospf, rip* are the keywords identified by our system. For a particular node the ancestors are pre-requisites and descendants are follow-ups. For example, the pre-requisites of the concept *tcp* is *topology* and the follow-up topic is *congestion control*.

We evaluated our system by manually comparing its results with the results given by an expert. We compared the keywords generated by our program with the expert generated keywords. Similarly we compared the relations generated by our system with those of the expert generated relations. We modeled a goodness metric as shown in the following equations. Here, $E_{pw}, S_{pw}, E_{nw}, S_{nw}$ denotes Expert Positive, Self Positive, Expert Negative, and Self Negative, respectively, for the keywords identified. Whereas, $E_{pr}, S_{pr}, E_{nr}, S_{nr}$ denote Expert Positive, Self Positive, Expert Negative, and Self Negative, respectively, for the relations determined. Expert positive is, the correct keywords given by system with respect to the expert results. On the other hand, self positive is, the correct keywords given by system with respect to keywords given by the system. Similarly, Expert Negative is the items given by expert but missed out by the system, and Self Negative is the spurious items reported by the system.

A. Goodness Metrics

Let

- W_e and R_e denote the total number of keywords and relations suggested by the expert, respectively,
- W_s and R_s denote the total number of keywords and relations generated by our system, respectively, and
- W_c and R_c denote the common keywords and relations in both expert given and system generated, respectively, then

Expert Positive (E_p)

$$E_{pw} = \frac{W_c}{W_e} \qquad E_{pr} = \frac{R_c}{R_e}$$

Self Positive (S_p)

$$S_{pw} = \frac{W_c}{W_s} \qquad S_{pr} = \frac{R_c}{R_s}$$

Expert Negative (E_n)

$$E_{nw} = \frac{W_e - W_c}{W_e} \qquad E_{nr} = \frac{R_e - R_c}{R_e}$$

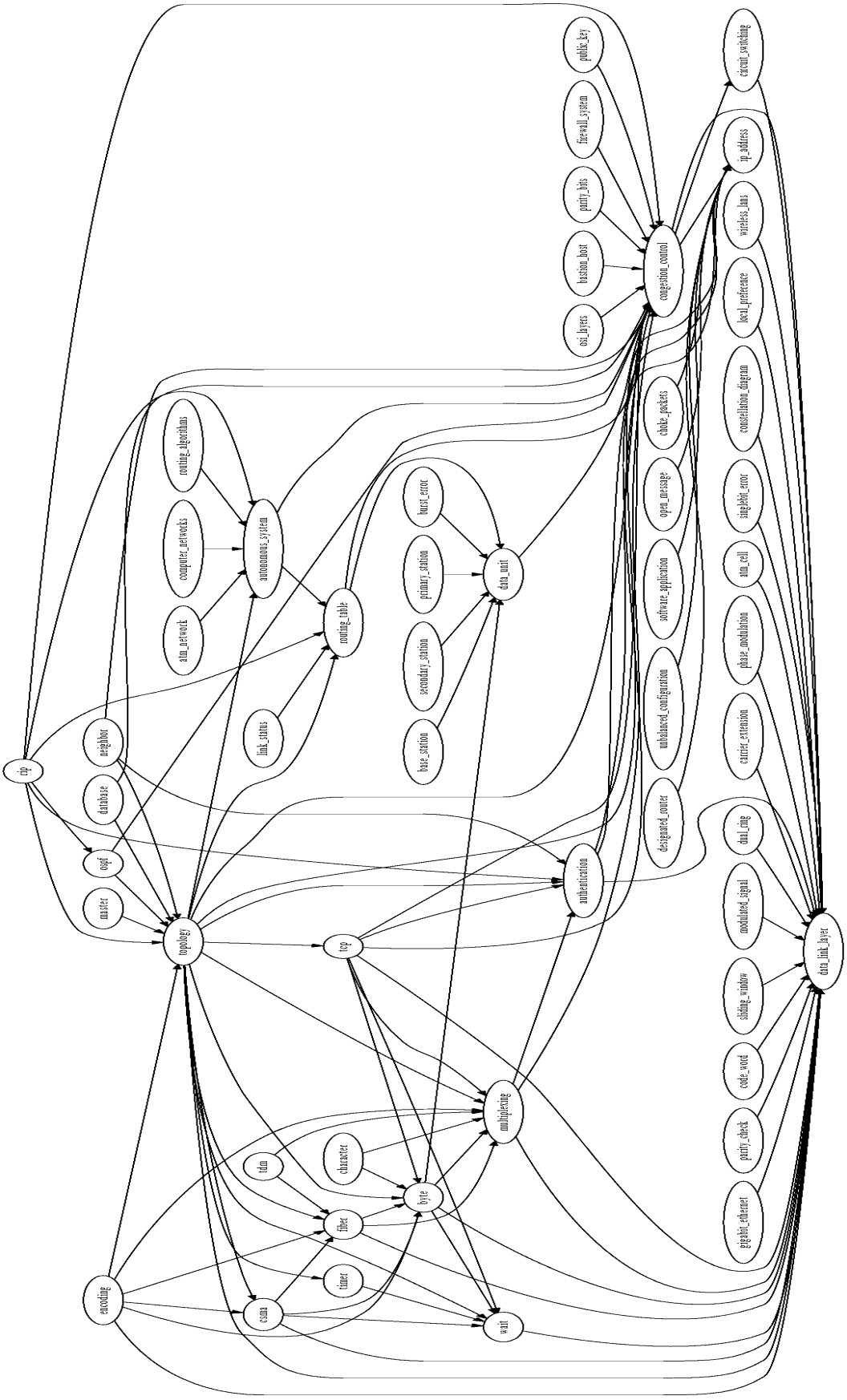

Figure 4. DAG for Computer Networks

Self Negative (S_n)

$$S_{nw} = \frac{W_s - W_c}{W_s} \qquad S_{nr} = \frac{R_s - R_c}{R_s}$$

For example, for a given subject as shown in the following Figure 5, let $w, x, y,$ and z be the keywords identified by an expert, and let $x, y,$ and p be the keywords identified by our system. In this case, True Positive is $\{x, y\}$, so the Expert Positive is $|\{x, y\}|/|\{w, x, y, z\}| = 2/4 = 0.5$, whereas the Self Positive is $|\{x, y\}|/|\{x, y, p\}| = 2/3 = 0.66$. False Positive is $\{p\}$, so the Self Negative is $|\{p\}|/|\{x, y, p\}| = 1/3 = 0.33$, and False Negative is$\{w, z\}$, so the Expert Negative is $|\{w, z\}|/|\{w, x, y, z\}| = 2/4 = 0.5$.

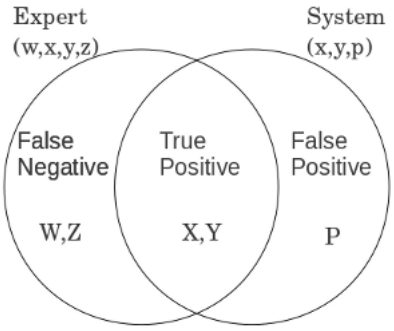

Figure 5. Classification Diagram

B. Performance Analysis

We manually identified the keywords for both the subjects Computer Networks and Operating Systems, then compared them with those of the results generated by our system. The results are given in the following tables. Table II shows the results for Computer Networks and Figure 6 is the graph for Computer Networks with respect to the results. In the same way, Table III and Figure 7 show the results for Operating Systems.

From the graph, we can observe that with increase in the number of system generated keywords, Expert Positive (E_p) increases. This is because, the expert given keywords (W_e) are fixed. At the same time with increase in number of keywords, Self Positive (S_p) increases upto a certain value. After this value if we further increase the number of keywords, it will result in more number of spurious keywords, than the number of legitimate keywords. So, it will start decreasing with increase in number of system generated keywords after this maximum value. We considered the value where Self Positive (S_p) is maximum as the optimum number of keywords. In the domain of "Computer Networks" the optimum value for number of keywords is 140. At this point 40.97% ($\frac{R_c}{R_s}$) of system answers are correct which happens to be finding 28.03% ($\frac{R_c}{R_e}$) of all correct answers. Similarly for "Operating System" domain the optimum value of keywords is 130.

The number of relevant keywords found was "fairly good". We have observed "some conflicts" when we compared the system results and the expert answers. "Some" relations which were generated by our system were not valid. And "some" important relations, which should have been obtained, were not generated by the system.

Table I
RESULTS FOR COMPUTER NETWORKS

keywords	Expert Positive(%)		Self Positive(%)		Expert Negative(%)		Self Negative(%)	
	E_{pw}	E_{pr}	S_{pw}	S_{pr}	E_{nw}	E_{nr}	S_{nw}	S_{nr}
69	18.02	14.09	44.92	40.82	81.97	18.03	55.06	59.18
92	22.09	18.02	41.3	37.1	77.9	22.1	58.69	62.9
101	26.74	21.14	45.54	41.39	73.25	26.75	54.44	58.61
114	30.23	23.21	45.61	41.09	69.76	30.24	54.38	58.91
127	30.23	25.09	40.94	37.18	69.76	30.24	59.05	62.82
140	36.04	28.03	44.28	40.97	63.95	36.05	55.70	59.03
143	34.3	28.03	41.25	36.95	65.69	34.31	58.70	63.05
148	34.3	29.4	39.86	36.02	65.69	34.31	60.08	63.98
153	38.37	31.57	43.13	39.18	61.62	38.38	56.86	60.82
167	38.95	32.19	40.11	36.51	61.04	38.96	59.87	63.49
174	38.95	32.19	38.5	34.51	61.04	38.96	61.49	65.49
183	38.95	32.19	36.61	33.41	61.04	38.96	63.35	66.59
195	39.53	32.93	34.87	31.27	60.46	39.54	65.12	68.73

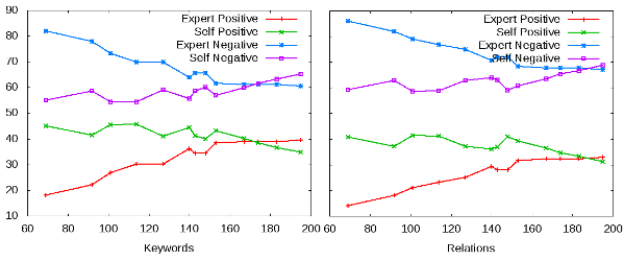

Figure 6. Computer Networks

Table II
RESULTS FOR OPERATING SYSTEMS

keywords	Expert Positive(%)		Self Positive(%)		Expert Negative(%)		Self Negative(%)	
	E_{pw}	E_{pr}	S_{pw}	S_{pr}	E_{nw}	E_{nr}	S_{nw}	S_{nr}
58	15.88	11.58	29.31	25.21	84.11	74.79	70.68	74.79
78	18.69	13.49	25.64	21.13	81.3	78.87	74.35	78.87
87	21.49	21.39	26.43	22.41	78.5	77.59	73.56	77.59
101	28.03	23.23	29.7	22.17	71.96	77.83	70.29	77.83
103	28.97	23.07	30.09	26.39	71.02	73.61	69.89	73.61
124	36.44	32.14	31.45	27.15	63.55	72.85	68.54	72.85
129	40.18	37.28	33.33	29.03	59.81	70.97	66.66	70.97
133	40.18	37.28	32.33	28.31	59.81	71.69	67.67	71.69
144	42.99	38.19	31.94	27.51	57	72.49	68.05	72.49
156	44.86	42.76	30.76	27.16	55.14	72.84	69.23	72.84
168	45.79	43.39	29.16	26.16	54.2	73.84	70.83	73.84
179	44.85	41.85	26.81	22.11	55.14	77.89	73.18	77.89
195	44.85	41.45	24.61	21.1	55.14	78.9	75.38	78.9

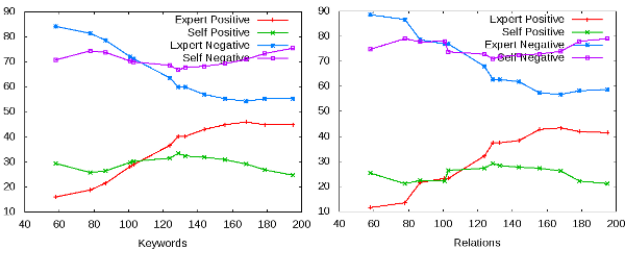

Figure 7. Operating Systems

C. Configuring the system

After conducting different experiments, we observed that Self Positive was maximum when the number of keywords was nearly equivalent to 140. The optimum value of Self Positive was obtained when the number of unigrams was 100, number of bigrams was 70, number of trigrams was 50, and number of fourgrams was 20. This is

because most of the keywords in any subject are unigrams, there are very few fourgrams, and there may be hardly any fivegrams. However, more experiments with lecture notes in various subjects are required before these results can be generalized. Moreover, the goodness metric can be improved by considering a higher number of keywords, at the cost of increase in execution time.

VI. Conclusion

We have developed an automatic ontology generator to get a dependency graph of topics in an area from a set of lecture notes. We evaluated our system by defining a goodness metric and found that its performance is comparable to the keywords and relations generated manually by an expert in the area. In our experiments we found 40% of our answers are correct, which happens to be finding 30% of all correct answers. We have used free and open source components for building our system and believe that such a system will be of use to courseware repositories.

References

[1] J. D. Novak and A. J. C. nas, "The theory underlying concept maps and how to construct and use them," in *Technical Report IHMC CmapTools 2006-01 Rev 01-2008*. Florida Institute for Human and Machine Cognition, 2008.

[2] MIT, "Mit open courseware - monthly reports," accessed 16-February-2011. [Online]. Available: http://ocw.mit.edu/about/site-statistics/monthly-reports/

[3] NPTEL, "Nptel — project document," Department of Secondary and Higher Education, Ministry of Human Resource Development, Government of India, New Delhi., July 2007. [Online]. Available: http://nptel.iitm.ac.in/pdf/NPTEL%20Document.pdf

[4] W. M.-j. YUN Hong-yan, XU Jian-liang and X. Jing, "Development of domain ontology for e-learning course," in ITIME-09IEEE international symposium, 2009.

[5] T.R.Guber, "Towards principles for the design of ontologies used for knowledge sharing," in *Int..J.Human-Computer Studies*. Florida Institute for Human and Machine Cognition,43(5-6), p.p 9.7-928, 1993.

[6] D. Fensel, I. Horrocks, F. van Harmelen, D. L. McGuinness, and P. Patel-Schneider, "Oil: An ontology infrastructure for the semantic web," *IEEE Intelligent Systems*, vol. 16, no. 2, 2001.

[7] Wikipedia, "Dependency graph — wikipedia, the free encyclopedia," 2011, [Online; accessed 16-February-2011]. [Online]. Available: http://en.wikipedia.org/w/index.php?title=Dependency_graph&oldid=408804604

[8] Apache, "The apache software foundation — pdfbox," accessed 16-February-2011. [Online]. Available: http://pdfbox.apache.org/download.html

[9] Wikipedia, "Lucene — wikipedia, the free encyclopedia," 2011, [Online; accessed 16-February-2011]. [Online]. Available: http://en.wikipedia.org/w/index.php?title=Lucene&oldid=414072215

[10] J. Ramos, "Using tf-idf to determine word relevance in document queries," Department of Computer Science, Rutgers University, 23515 BPO Way, Piscataway, NJ, 08855., 2002.

[11] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules in large databases," in *Proceedings of the 20th International Conference on Very Large Data Bases*, ser. VLDB '94. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1994, pp. 487–499. [Online]. Available: http://portal.acm.org/citation.cfm?id=645920.672836

[12] E. Gansner, E. Koutsofios, and S. North, "Drawing graphs with dot," Jan 2006. [Online]. Available: www.graphviz.org/Documentation/dotguide.pdf

[13] Wikipedia, "Terminology extraction — wikipedia, the free encyclopedia," 2011, [Online; accessed 16-February-2011]. [Online]. Available: http://en.wikipedia.org/w/index.php?title=Terminology_extraction&oldid=406585677

[14] Python, "package index — topia.termextract 1.1.0," accessed 16-February-2011. [Online]. Available: http://pypi.python.org/pypi/topia.termextract/