Effective Metadata Extraction from Irregularly Structured Web Content

Baoyao Zhou, Wei Liu, Yu Yang, Weichun Wang, Ming Zhang

HP Laboratories
HPL-2008-203

Keyword(s):
Information Extraction, Metadata, Online Course Organization, Logical Structure Model.

Abstract:

Metadata extraction is one crucial module for domain specific Web content discovery and management, because the accuracy and completeness of the extracted metadata would directly affect the quality of subsequent domain information services. Our Online Course Organization project aims to build an online course portal to serve the course information obtained from the Web. Since most course pages are irregularly structured, most existing approaches are not effective for extracting course metadata. In this paper, we proposed a novel hierarchical clustering approach to generate a web page semantic structure model from the DOM tree, called Logical Structure Model, such that the hidden patterns and knowledge can be revealed and used to facilitate identifying course metadata. The experimental results have shown that our solution can achieve effective metadata extraction.

External Posting Date: November 21, 2008 [Fulltext] Approved for External Publication
Internal Posting Date: November 21, 2008 [Fulltext]

© Copyright 2008 Hewlett-Packard Development Company, L.P.

Effective Metadata Extraction from Irregularly Structured Web Content

Baoyao Zhou, Wei Liu
HP Labs China
{baoyao.zhou, wliu}@hp.com

Yu Yang, Weichun Wang, Ming Zhang
Peking University, China
{yuyang.db, wangwch, mzhang_cs}@pku.edu.cn

ABSTRACT

Metadata extraction is one crucial module for domain specific Web content discovery and management, because the accuracy and completeness of the extracted metadata would directly affect the quality of subsequent domain information services. Our Online Course Organization project aims to build an online course portal to serve the course information obtained from the Web. Since most course pages are irregularly structured, most existing approaches are not effective for extracting course metadata. In this paper, we proposed a novel hierarchical clustering approach to generate a web page semantic structure model from the DOM tree, called Logical Structure Model, such that the hidden patterns and knowledge can be revealed and used to facilitate identifying course metadata. The experimental results have shown that our solution can achieve effective metadata extraction.

Keywords

Information Extraction, Metadata, Online Course Organization, Logical Structure Model.

1. INTRODUCTION

The World Wide Web serves as a huge, widely distributed, global information service center. Many general purpose search engines, like Google, Yahoo! and MSN, have been developed to assist users to find relevant web pages effectively. However, they still cannot give accurate and complete answers for more specific queries regarding a particular domain. For example, "Which models of HP photo inkjet printers are discussed popularly on digital product review forums?", "Which textbooks are popular for Information Theory courses?" and "What CS courses are provided by UC Berkeley?" To answer such questions effectively, further research needs to be carried out to develop technologies for analyzing and understanding the semantics of Web content, such as who, what, where, when they are talking about. For Business Intelligence, such technologies can potentially help companies to analyze customer online feedback/comments, competitive product information on the Web and online news about their events to drive product/service improvements and targeted marketing programs. Our Online Course Organization project is aimed at using online courses as an example domain to develop these technologies for domain specific Web content discovery and management, especially how to extract semantic metadata from irregularly structured Web content. Meanwhile, we also attempt to build an online course portal to serve the course information we obtained from the Web. In addition, we plan to generalize this solution to other specific domains for more general purposes, especially for benefiting HP's business.

Different from other domain pages with regular structures, such as product information pages and academic publication pages, most online course pages are irregularly structured. The main reason is that most course pages are designed and maintained by the course instructors in their specific ways. Our work focuses on how to effectively extract useful metadata from a course page returned by our course page crawler and classifier. Such course metadata include Course_ID, Course_Name, Course_Time, Teacher_Name and Teacher_Email. However, most existing approaches for extracting metadata from regular pages are not appropriate for course pages any more. Fortunately, most course pages are designed in very simple HTML, which makes it possible to deduce the approximate semantic structure from their DOM (http://www.w3.org/DOM/) tree, which then can be used to facilitate extracting metadata.

In this paper, we firstly propose a novel hierarchical clustering approach to discover the Logical Structure Model of web pages. Different from other existing web page content structure models, the proposed Logical Structure Model can present more detailed and comprehensive structure information of web page content. Based on the Logical Structure Model, the course metadata then can be extracted easily by some heuristic rules. We apply such solution to achieve course metadata extraction, but it is also possible to extend to other domains, such as online news, product review information, even Blogs.

The rest of this paper is organized as follows. In Section 2, we review the related work on metadata extraction and web page semantic structure discovery. Sections 3 and 4 present our proposed approach for discovering web page Logical Structure Model and extracting course metadata. Performance evaluation of the proposed approach is given in Section 5. Finally, Section 6 concludes the paper.

2. RELATED WORK

Metadata (http://en.wikipedia.org/wiki/Metadata) is data about data, more specifically a collection of key information about a particular content, which can be used to facilitate the understanding, use and management of data. Metadata extraction, especially from irregularly structured Web content, is still a challenging issue for the research area on content management.

Existing metadata extraction solutions can be mainly classified into three categories: wrapper induction [1][2], sliding window and boundary finding model [3][4], finite state machines (Hidden Markov Models [5][6], Conditional Random Fields [7][8][9]). However, most approaches are only effective for structured Web content, i.e., web pages with similar layout templates or semantic structures, but not appropriate for irregularly structured Web content. The main reason is that the semantic or logical patterns of metadata are usually not obvious in such irregularly structured Web content. For example, Teacher_Name may appear nearby Course_Name or in the content of "Instructor:" heading, but such logical pattern is hidden in course pages with various HTML implementations by different instructors. One promising solution to solve this problem is to construct a brief semantic structure model for web pages such that the hidden patterns and knowledge can be revealed and used to facilitate metadata extraction. That is exactly the main idea of our solution.

Many web page content structure models have been proposed, such as FOM [10], VIPS [11], PAS [12], etc. However, they only can describe the partition layout structure of a web page, i.e., a hierarchical tree structure, in which each node represents a page segment called *block* and all child nodes of a certain node represent a more detailed partition of the corresponding block, but not the hierarchical structure of semantics in Web content. In addition, most existing web page content structure analysis or web page segmentation approaches usually define the leaf nodes (mainly include text nodes, <IMG> and other specific objects) in the DOM tree as the basic objects or tokens, i.e., the smallest and undividable units. However, sometimes, one leaf nodes, especially text nodes, may contain several tokens with different semantics. For example, "CS102: Data Structure" consists of two tokens, i.e., Course_ID "CS102" and Course_Name "Data Structure". Therefore, existing web page content structure models do not satisfy the need to support metadata extraction from irregularly structured Web content.

3. WEB PAGE LOGICAL STRUCTURE DISCOVERY

A web page (i.e., a HTML document) can be parsed as a tree-based structure model called Document Object Model (DOM in short) by a standard HTML parser. Although DOM was initially designed to define the logical structure of documents, it cannot exactly represent the true inner semantics of HTML documents due to the flexibility of HTML syntax. However, it is still possible to deduce the approximate logical structure of HTML documents according to their DOM, especially for web pages written in simple HTML, such as most of course pages. Here, the web page logical structure means a document structure model that can represent the actual hierarchical relationships among segments with certain semantics in a web page. A good logical structure model can be used to facilitate pattern discovering from web page content. In this section, we propose a novel hierarchical clustering approach to deduce the approximate logical structure of a web page from its DOM tree.

Compared with other existing web page content structure models, the proposed Logic Structure Model (*LSM* in short) has the following innovations and contributions:

1. The definition of tokens (the smallest and undividable units) is independent to the leaf nodes in the DOM tree. In other words, one text node may be divided into several tokens, or several adjacent text nodes may be jointed into one token. Therefore, the proposed *LSM* can represent more detailed semantic and logical information of web page content.
2. The proposed *LSM* can represent the hierarchical section outline structure of a web page, i.e., not only the partition blocks but also the hierarchical relationships among them.

We define the *LSM* of a web page as follows.

Definition 1: A web page W can be represented as a tree structure model called Logical Structure Model $LSM = (C, R)$, where $C = \{c_1, c_2, \ldots c_n\}$ is a set of all tree nodes also known as token clusters, and $R = \{<c_i, c_j>\}$ is a set of all parent-child relationships among token clusters. Each token cluster has a type, which is one of PAGE, SEGMENT, HEADING and CONTENT. The PAGE cluster is the root node of *LSM*. All SEGMENT clusters are the second level nodes of *LSM*, i.e., children of the PAGE cluster. Both PAGE and SEGMENT clusters own all tokens in their descendant clusters as their tokens. Others are HEADING or CONTENT clusters, each of which has a list of their own tokens. According to the types of the parent and child clusters, each parent-child relationship can represent one of *page-segment*, *segment-heading*, *segment-content*, *heading-subheading* and *heading-content* relationships between two clusters.

Figure 1 shows an example of the *LSM* constructed from a DOM tree of a sample course page. The sample course page (PAGE cluster) is divided into two segments (SEGEMENT clusters). Each segment has a tree of its own token clusters (HEADING and CONTENT clusters) with their hierarchical relationships. For example, Node 3 represents a HEADING cluster with two tokens "CSD 2005" and "Geometric Mechanics". And the relationship between Node 5 and Node 6 indicates that the heading of content "Jerrold Marsden" is "Instructor:". Based on the constructed *LSM*, metadata should be much easier to be extracted from a course page than based on the original DOM. For example, it is difficult to identify which one of the two email addresses in the sample course page is the teacher email based on the DOM tree, because both of them and the reference heading text "Instructor:" are leaf nodes and we cannot ensure which one belongs to the content of "Instructor:". However, in *LSM*, it is obvious that the teacher email is "marsden@cds.caltech.edu", because only this email address appears in the descendant clusters of "Instructor:".

To generate *LSM* from DOM, we need to perform following tasks:

Step 1. *Preprocessing*: To extract all text nodes associated with their location information in the DOM tree and other specific HTML properties (heading level, font size, font style, link, etc.).

Step 2. *Text Node Generalization*: To generalize each text node to form one or more initial tokens with predefined attributes (*number*, *alphabetical text with number*, *alphabetical text with capital letter*, *time*, *person title*, *name*, *email* and *others*).

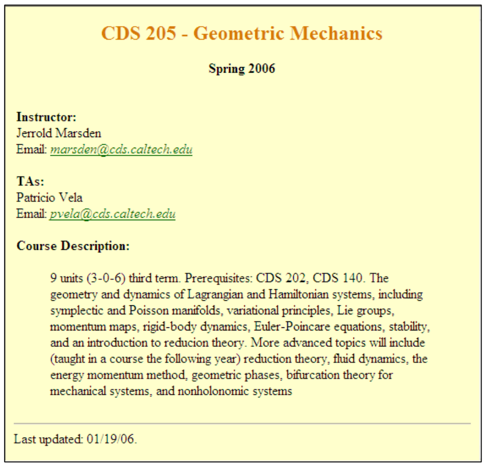
Step 3. *Hierarchical Clustering*: To generate the initial clusters with one cluster including one initial token, and then, recursively group tokens in similar clusters into bigger clusters and deduce the semantic logical relationships between dissimilar clusters by considering their locations in the DOM tree and specific HTML properties

The main idea also can be adapted to construct *LSM* for other domain web pages. In the subsequent sections, we will introduce the details of the proposed approach.

3.1 Preprocessing

After parsing the input HTML document as a corresponding DOM tree, all HTML element and text nodes can be visited easily by a traversal of the DOM tree. In this research, we only consider the textual contents, i.e., all text nodes, and separator objects (<BR>, <HR>, etc.) in course pages.

A sample course page



CDS 205 - Geometric Mechanics

Spring 2006

Instructor:
Jerrold Marsden
Email: marsden@cds.caltech.edu

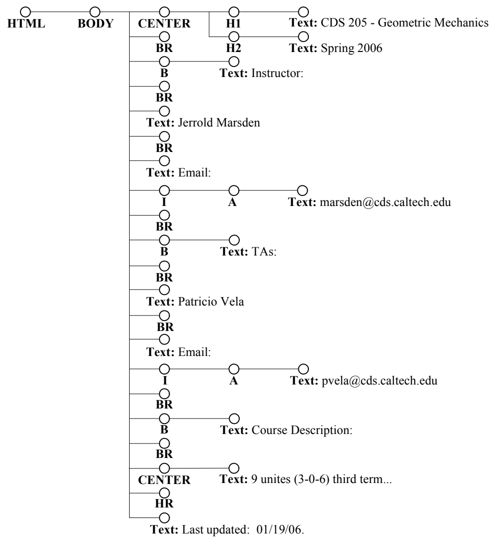
TAs:
Patricio Vela
Email: pvela@cds.caltech.edu

Course Description:

9 units (3-0-6) third term. Prerequisites: CDS 202, CDS 140. The geometry and dynamics of Lagrangian and Hamiltonian systems, including symplectic and Poisson manifolds, variational principles, Lie groups, momentum maps, rigid-body dynamics, Euler-Poincare equations, stability, and an introduction to reducion theory. More advanced topics will include (taught in a course the following year) reduction theory, fluid dynamics, the energy momentum method, geometric phases, bifurcation theory for mechanical systems, and nonholonomic systems

Last updated: 01/19/06.

DOM tree


HTML
BODY
CENTER
H1
Text: CDS 205 - Geometric Mechanics
BR
H2
Text: Spring 2006
B
Text: Instructor:
BR
Text: Jerrold Marsden
BR
Text: Email:
I
A
Text: marsden@cds.caltech.edu
BR
B
Text: TAs:
BR
Text: Patricio Vela
BR
Text: Email:
I
A
Text: pvela@cds.caltech.edu
BR
B
Text: Course Description:
BR
CENTER
Text: 9 unites (3-0-6) third term...
HR
Text: Last updated: 01/19/06.

Logical Structure Model

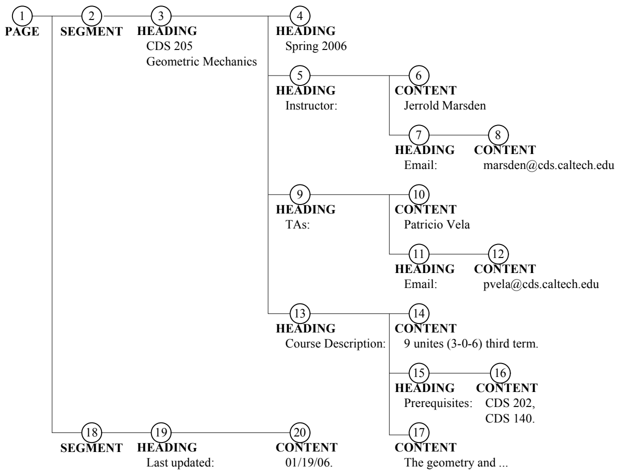

1 PAGE
2 SEGMENT
3 HEADING CDS 205 Geometric Mechanics
4 HEADING Spring 2006
5 HEADING Instructor:
6 CONTENT Jerrold Marsden
7 HEADING Email:
8 CONTENT marsden@cds.caltech.edu
9 HEADING TAs:
10 CONTENT Patricio Vela
11 HEADING Email:
12 CONTENT pvela@cds.caltech.edu
13 HEADING Course Description:
14 CONTENT 9 unites (3-0-6) third term.
15 HEADING Prerequisites:
16 CONTENT CDS 202, CDS 140.
17 CONTENT The geometry and ...
18 SEGMENT
19 HEADING Last updated:
20 CONTENT 01/19/06.

Figure 1. The Logical Structure Model of a sample course page.

In general, text nodes close to each other in the DOM tree are usually close when shown in the Web browser. Therefore, it is crucial to keep the location information for each text node, which can then be used for measuring the semantic and logical relationship among text nodes. In this research, we adopt the Absolute Location Path to represent the location of each text node in the DOM tree. An Absolute Location Path is an XPath (http://www.w3.org/XPath/) expression consisting of a sequence of all location steps separated by "/" from the <HTML> element to the target node (here is each text node) in document order [1], where each location step is denoted as a HTML element name with its position code. For example, an Absolute Location Path "/HTML[1]/BODY[1]/H1[2]/text()[1]" represents the text node that belongs to the second <H1> element in the children of the first <BODY> element in the children of the <HTML> element. Its mathematical definition is given as follows.

Definition 2: An Absolute Location Path can be denoted as a sequence of Location Steps, i.e., $ALP = /s_1/s_2/\ldots/s_n$, where each Location Step $s_i = element\_name_i[position\_code_i]$ for $1 \le i \le n$.

Apart from the location information, we also keep some important HTML properties for each text node, which include the heading level, font size, hyperlink and font style. Such information can be easily obtained from the ancestor element nodes of the corresponding text node.

According to the text pattern and HTML properties, each text node is classified into HEADING text or CONTENT text as its text type using the following heuristic rules:

1. Text node with an ancestor heading element (such as <H1>~<H6>);
2. Text node with an ancestor highlight element (such as <B>, <BIG>, <EM>, <I>, <STRONG>, <U> and <FONT> with font size larger than normal size), and with a parent paragraph element (such as <P>, <DIV>, <TD>, <CENTER>, etc.) or followed by <BR>, and with at most 10 words;
3. Text node with capital (initial) letters, and ending with a colon, and with at most 10 words;

After preprocessing a web page, we have obtained a list of all text nodes associated with their Absolute Location Paths, text type and some specific HTML properties.

3.2 Text Node Generalization

As mentioned earlier, most existing approaches consider the leaf nodes in the DOM tree as the basic tokens. Sometimes, one text node may be further divided into several tokens with different semantics. For example, a text node "Chem 24ab Introduction to Biophysical Chemistry" contains one Course_ID token "Chem 24ab" and one Course_Name token "Introduction to Biophysical Chemistry", and another text node "Instructor: Wennberg, Seinfeld" contains one Teacher_Heading token "Instructor:" and two Teacher_Name tokens "Wennberg" and "Seinfeld". To distinguish different semantic tokens in one text node, we regard all single words (texts separated by white-space) of each text node as the smallest units. We also define a total of eight basic attributes by simple regular expressions: *number, alphabetical*

[1] Document order orders element nodes in order of the occurrence of their start-tag in the XML.

text with number, alphabetical text with capital letter, time, person title, name, email and others. Each word is classified into one of these eight categories as its attribute. And adjacent words with the same attributes are jointed together to form a phrase, sentence or paragraph. As a result, each text node forms one or several tokens called *initial tokens*.

After text node generalization, we obtained a list of initial tokens with their basic attributes, Absolute Location Paths and specific HTML properties inherited from corresponding text nodes.

3.3 Hierarchical Clustering

In this section, we propose a novel hierarchical clustering algorithm to group similar tokens into clusters and deduce hierarchical relationships among clusters. The input of the algorithm is a list of initial tokens, and the output will be a *LSM*.

The main tasks of the proposed hierarchical clustering algorithm are listed as follows:

1. Construct initial cluster list by putting each initial token into one different initial cluster. And each initial cluster inherits the Absolute Location Path and specific HTML properties from its initial token.
2. Scan the current cluster list from tail to head. Compare the current cluster c_β with each of its previous candidate clusters c_α (i.e., clusters in the rightmost branch of the previous clusters from its rightmost cluster) until no previous candidate cluster left or one of the following operations can be applied to combine c_α and c_β. Which operation is required to be applied in practice depends on the distance between c_α and c_β, the text types (HEADING or CONTENT) and other properties (heading level, font size and font style) of c_α and c_β.

   If the distance between c_α and c_β, is less than or equal to a given minimum distance d_{min}, we need to

   a) merge current cluster c_β into the previous candidate cluster c_α,
      - if c_α and c_β are both HEADING clusters with the same heading level AND c_α has no child cluster.
      - if c_α and c_β are both CONTENT clusters with the same font size AND c_α has no child cluster.

   b) append current cluster c_β as the last child of the previous candidate cluster c_α,
      - if c_α and c_β are both HEADING clusters AND c_α has higher heading level than that of c_β.
      - if c_α and c_β are HEADING and CONTENT clusters respectively AND c_α has no child cluster.

   c) append current cluster c_β as the last sibling of the previous candidate cluster c_α,
      - if c_α and c_β are both HEADING clusters with the same heading level and c_α has child clusters.
      - if c_α and c_β are HEADING and CONTENT clusters respectively AND c_α has child clusters.
      - if c_α and c_β are both CONTENT clusters with the different font size.
      - if c_α and c_β are both CONTENT clusters AND c_α has child clusters.

   The cluster list needs to be updated after each operation. The updated cluster list should contain a list of root nodes of cluster trees.
3. Repeat Step 2 to recursively process the updated cluster list, until there is only one cluster left in the current cluster list or no two adjacent clusters can be combined.

To merge similar clusters together and distinguish dissimilar clusters, we should define the similarity or distance between two clusters. In general, the distance between two text nodes in the DOM tree can be defined as the length of the shortest path between them, i.e., the minimum number of edges needed to connect these two text nodes, which can be calculated based on the Absolute Location Paths of two text nodes. Such definition also can be used to evaluate the distance between two clusters.

The mathematical definition on the distance between two clusters is given as follows.

Definition 3: For two token cluster c and c' with Absolute Location Paths $ALP = /s_1/s_2/\dots/s_n$ and $ALP' = /s'_1/s'_2/\dots/s'_m$ respectively, there must exist some $1 \le k \le min(n, m)$, such that $s_i = s'_i$ for $1 \le i \le k$. Among them, the maximum k is denoted as k_{max}. The distance between these two clusters is defined as $d(c, c') = n + m - 2 \times k_{max}$.

When two clusters need to be merged together, we remain the shorter *ALP* of these two clusters as the *ALP* of the new cluster.

The algorithm for constructing *LSM* based on an initial token list is given in Algorithm 1 and 2.

Algorithm 1. Construct_LSM(T_L)

```
Input:
  $T_L$ = {$token_i$} – The initial token list
Output:
  Page – The page cluster of the LSM
Process:
1. Construct initial cluster list $C_L$ from initial token list $T_L$
2. $d_{min} \leftarrow 0$
3. $d_{max} \leftarrow$ The average length of all branches in the DOM tree
4. Page ← Hierarchical_Clustering($C_L$, $d_{min}$, $d_{max}$)
5. return Page
```

Note that if the size of cluster list does not reduce after one scan, we need to increase d_{min} by 1, otherwise initialize it as 0 to do another scan.

4. COURSE METADATA EXTRACTION

Based on the generated *LSM* of a course page, course metadata can be extracted more effectively. For Course_ID and Course_Name, we also apply special strategies to improve the accuracy of extractions, which include taking into account information in web page titles and making use of a predefined course name list.

4.1 Course ID Extraction

The ID of a course is usually denoted in the form of one or several discipline abbreviations plus an alphanumeric code, such as CS134b, Ph 136 and ChE/BE 210. Course_ID is a very important kind of metadata for Online Course Organization, because the discipline abbreviations in it can provide the explicit evidence for

Algorithm 2. Hierarchical_Clustering(C_L, d_{min}, d_{max})

```
Input:
  $C_L = \{c_i\}$ – The current cluster list
  $d_{min}$ – The minimum distance
  $d_{max}$ – The maximum distance
Output:
  Page –The page cluster of the LSM
Process:
1.  if $| C_L | = 1$ or $d_{min} > d_{max}$ then
2.      Create empty cluster Page with Page.type ← Page
3.      for all cluster $c_i \in C_L$ do
4.          Create empty cluster Seg with Seg.type ← Segment
5.          Seg.appendChild($c_i$)
6.          Page.appendChild(Seg)
7.      end for
8.      return Page
9.  end if
10. Cluster number $n \leftarrow | C_L |$
11. for $i = n$ to 2 do
12.     The previous candidate cluster $c_\alpha$, the rightmost cluster of
        $c_{i-1} \in C_L$
13.     The current cluster $c_\beta \leftarrow c_i \in C_L$
14.     while $c_\alpha$ != null and combine_flag = false do
15.         if $d(c_\alpha, c_\beta) \le d_{min}$ then
16.             if $c_\alpha$.type=Heading and $c_\beta$.type=Heading then
17.                 if $c_\alpha$.headingLevel = $c_\beta$.headingLevel then
18.                     if $c_\alpha$ has no child cluster then
19.                         Merge $c_\beta$ into $c_\alpha$, $C_L$.remove($c_\beta$)
20.                     else
21.                         $c_\alpha$.appendSibling($c_\beta$), $C_L$.remove($c_\beta$)
22.                     end if
23.                 else if $c_\alpha$.headingLevel > $c_\beta$.headingLevel
24.                     $c_\alpha$.appendChild($c_\beta$), $C_L$.remove($c_\beta$)
25.                 end if
26.             else if $c_\alpha$.type=Heading and $c_\beta$.type=Content then
27.                     if $c_\alpha$ has no child cluster then
28.                         $c_\alpha$.appendChild($c_\beta$), $C_L$.remove($c_\beta$)
29.                     else
30.                         $c_\alpha$.appendSibling($c_\beta$), $C_L$.remove($c_\beta$)
31.                     end if
32.             else if $c_\alpha$.type=Content and $c_\beta$.type=Content then
33.                 if $c_\alpha$.fontSize = $c_\beta$.fontSize and $c_\alpha$ has no child
                    cluster then
34.                     Merge $c_\beta$ into $c_\alpha$, $C_L$.remove($c_\beta$)
35.                 else
36.                     $c_\alpha$.appendSibling($c_\beta$), $C_L$.remove($c_\beta$)
37.                 end if
38.             end if
39.         end if
40.         $c_\alpha \leftarrow c_\alpha$.getParentCluster();
41.     end while
42. end for
43. if $| C_L | = n$ then
44.     $d_{min}$ ++
45. else
46.     $d_{min} \leftarrow 0$
47. end if
48. return Hierarchical_Clustering($C_L$, $d_{min}$, $d_{max}$)
```

subject-based course classification. For example, for a course "Cellular Engineering" with Course_ID "ChE/BE 210", it is obvious that this course can be classified into the subjects "Chemistry Engineering" and "Biological Engineering". Using Course_ID as a significant clue, the course classification should be much more applicable than most existing approaches, which are mainly based on technologies of text mining and web link analysis. In addition, Course_ID is also a key reference for locating other metadata, especially Course_Name, which usually occurs following Course_ID.

Any tokens with the attribute of *alphabetical text with number* in the generated *LSM* can be regarded as Course_ID candidates. To identify the actual Course_ID, one possible way is to enumerate all discipline abbreviations (e.g., CS for Computer Science, Ph for Physics and EconS for Economic Science), match all candidates with them, and then, label the most probably ones as Course_IDs. However, such task should be very costly in practice. In addition, sometimes, we may extract several different Course_IDs or no Course_ID. For example, if the course "CS101" discusses the course "EE123" in its content, then both "CS101" and "EE123" will be identified as Course_IDs. For another example, if a course has Course_ID "ERS 100", but "ERS" is not included in the predefined discipline abbreviation list, then "ERS 100" will not be identified as Course_ID. In practice, more than 80% of course page authors like to include course id in the HTML title of course pages. Upon the above fact, we implement the efficient and effective Course_ID extraction using the course page titles.

At first, the short alphabetical text with number in the HTML title is extracted as the potential course id. Then, we only need to compare all candidates with this potential course id one by one. If a matching candidate can be found, we can confirm that it is the actual Course_ID. Such simple method can easily and accurately identify Course_ID. However, sometimes, we still extract no Course_ID due to author's editing mistakes. For example, some course pages were modified from other web pages, but the author forgot to update the HTML title. We have seen a course page of "ECL 290", but the course id included in its HTML title is "ASE 110C". To deal with such cases, we have to refer to the predefined discipline abbreviation list to extract Course_ID again.

4.2 Course Name Extraction

Course_Name is not only the most important metadata for Online Course Organization, but also the key reference for locating other metadata in the generated *LSM*. For example, a course page may contain several names and emails, but only some of them in the descendant clusters of the cluster with Course_Name token in *LSM* are most likely Teacher_Name and Teacher_Email.

Any tokens with the attribute of *alphabetical text with capital letter* in the generated *LSM* can be regarded as Course_Name candidates. At the beginning, we attempt to use the similar idea for extracting Course_ID to identify Course_Name. But, unfortunately, only less than 10% course pages include the actual course name in their HTML title. Therefore, in most cases, we cannot find the potential course name from the HTML title. To extract Course_Name accurately, a predefined Course Name List is used to evaluate the probabilities of Course_Name candidates. Since it is impossible to enumerate all valid course names in the world, we need an approximate matching strategy to reduce the influence from the incomplete course name list.

In this research, a total of 5,928 course names have been extracted respectively from the MIT OpenCourseWare website (http://ocw.mit.edu/OcwWeb/web/courses/courses/), the UIUC website (http://courses.uiuc.edu/cis/catalog/urbana/2007/Fall/) and the English version of China Discipline Classification and Code. Basically, each course name can be defined as a sequence of keywords.

Definition 4: A sequence of keywords is denoted as $S = w_1w_2\ldots w_n$ ($n \geq 1$). A sequence of keywords $S' = w'_1\ w'_2\ldots w'_m$ is called a *sub-sequence* of $S = w_1w_2\ldots w_n$, denoted as $S' \subseteq S$, if and only if $\exists$ k ($0 \leq k \leq n-m$), such that $\forall$ $1 \leq i \leq m$, $w'_i = w_{i+k}$. We call that a sequence of keywords S_1 *approximately matches* a sequence of keywords S_2, if and only if $\exists$ a sequence of keywords S_0, such that $S_0 \subseteq S_1$ and $S_0 \subseteq S_2$.

According to the above definition, two sequences of keywords would approximately match each other, if and only if they have a common sub-sequence.

Definition 5: A *Course Name List* is a set of *Course Names*, denoted as $N_L = \{N_i\}$, where each *Course Name* N_i is a sequence of keywords. A sequence of keywords P is called a *Course Name Pattern*, if and only if $\exists$ $N_i \in N_L$, such that $P \subseteq N_i$.

Definition 6: A Course_Name candidate C is a sequence of keywords. We call that C *approximately matches* a *Course Name Pattern* P, if and only if $P \subseteq C$.

According to the above definitions, a Course_Name candidate can find an approximate matching course name pattern from a given course name list, if and only if there exists a course name in the course name list, which has a common sub-sequence as that the course name candidate has.

We propose a suffix-tree based model, called Course Name Pattern Tree (*CNP-tree* in short), to compactly store all course name patterns and facilitate the approximate matching. Except one virtual root node, each node in the *CNP-tree* represents a keyword. And each path from the root node to a certain node can be regarded as a sequence of keywords, i.e., a course name pattern. In addition, the support value associated with a certain node indicates how many times the corresponding course name pattern occurs in the given course name list. To improve the efficiency of pattern matching, all children of a keyword node are stored in a hash table.

Assume a course name list includes 3 course names, i.e., "Advanced Data Structure", "Data Structure" and "Data Mining", Figure 2 shows the process for constructing the corresponding *CNP-tree*. The path "Root → A(1) → D(1)" denotes a course name pattern "Advanced Data" with the support value of 1, and another path "Root → D(3) → S(2)" denotes a course name pattern "Data Structure" with the support value of 2.

The algorithm for constructing the Course Name Pattern Tree from a course name list is given in Algorithm 3. Each course name, i.e., a sequence of keywords, and its all suffix sequences are inserted or merged into the *CNP-tree* one by one. Meanwhile, the support value of each pattern is calculated.

We also compute a score for each Course_Name candidate according to its matching results in the given *CNP-tree*. In general, the Course_Name candidates matching the longer patterns with the greater support values should have the higher scores.

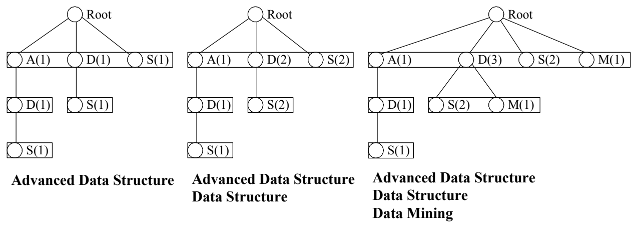


Figure 2. An example of constructing *CNP-tree*.

Algorithm 3. Construct_CNP-tree(N_L)

```
Input:
   $N_L = \{N_i\}$ – Course Name List
Output:
   Root – The root node of the CNP-tree
Process:
1.  Create an empty node Root as the root of the CNP-tree
2.  for all course names $N_i = w_1w_2\ldots w_n$ ($n \geq 1$) $\in N_L$ do
3.      for $i$ = 1 to $n$ do
4.          node ← Root
5.          for $j = i$ to $n$ do
6.              if node.children_hashtable.containsKey($w_j$) then
7.                  node ← node.children_hashtable.get($w_j$)
8.                  node.support++
9.              else
10.                 Create new_node with keyword ← $w_i$ and
                    support ← 1
11.                 node.children_hashtable.put($w_i$, new_node)
12.                 node ← new_node
13.             end if
14.         end for
15.     end for
16. end for
17. return Root
```

According to the above criteria, the course name score is defined as follows.

Definition 7: The score of a course name candidate C based on a given *CNP-tree* is defined as

$$Score(C) = \begin{cases} L_{\max} + \left(1 - \dfrac{1}{Support_{L_{\max}}}\right), & L_{\max} > 0 \\ 0, & otherwise \end{cases},$$

where $L_{\max}$ is the maximum length of all matching patterns and $Support_{L_{\max}}$ is the maximum support value of all matching longest patterns.

The main task for computing the score is to match all suffix sequences of a Course_Name candidate with the given *CNP-tree*.

Algorithm 4 shows the algorithm for computing the score of a Course_Name candidate based on a given *CNP-tree*.

For a Course_Name candidate "Advanced Data Mining", two matching 2-*length* course name patterns "Advanced Data (1)" and "Data Mining (1)" can be found in the *CNP-tree* given in Figure 2. Then, *Score*("Advanced Data Mining") = 2. For other examples, *Score*("Data Structure") = 2.5 and *Score*("Advanced Data Structure") = 3.0.

Algorithm 4. Compute_Course_Name_Score(*C, Root*)

```
Input:
  C – A course name candidate, i.e., a sequence of keywords C
      = $w_1 w_2 \ldots w_n$ ($n \geq 1$)
  Root – The root node of the given CNP-tree
Output:
  Score – The course name score of C based on the CNP-tree
Process:
1.  $L_{max} \leftarrow 0$, $Support_{Lmax} \leftarrow 0$
2.  for $i = 1$ to $n$ do
3.      if $(n-i+1) \geq L_{max}$ then
4.          node ← Root, pattern_length ← 0, support ← 0
5.          for $j = i$ to $n$ do
6.              if node.children_hashtable.containsKey($w_j$) then
7.                  node ← node.children_hashtable.get($w_j$)
8.                  support ← node.support
9.                  pattern_length++
10.             else
11.                 break
12.             end if
13.         end for
14.         if pattern_length > $L_{max}$ then
15.             $L_{max}$ ← pattern_length
16.             $Support_{Lmax}$ ← support
17.         else if pattern_length = $L_{max}$ then
18.             if support > $Support_{Lmax}$ then
19.                 $Support_{Lmax}$ ← support
20.             end if
21.         end if
22.     end if
23. end for
24. Score ← 0
25. if $L_{max} > 0$ then
26.     Score = $L_{max} + (1 - 1 / Support_{Lmax})$
27. end if
28. return Score
```

Apart from the course name score, we also consider other properties of the Course_Name candidates and their locations in the generated *LSM*. Such approach is also appropriate to extract other metadata like product name, company name, address, etc.

All heuristic rules used in the proposed approach for extracting Course_Name based on the *LSM* of a course page are listed in priority order as follows.

1. Token with the attribute of *alphabetical text with capital letter* can be Course_Name candidates;
2. Candidate with course name score ≥ 1 may be Course_Name;
3. Candidate with a higher header level or bigger font size is more probably Course_Name;
4. Candidates with higher score is more probably Course_Name
5. Candidate in the more frontal of the web page and close to Course_ID token is more probably Course_Name.

4.3 Other Course Metadata Extraction

After extracting Course_ID and Course_Name, other course metadata, such as Course_Time, Teacher_Name and Teacher_Email, can be located and extracted easily from the *LSM*. For example, Teacher_Name usually appears following a Teacher_Heading, i.e., a token with a keyword of "Instructor", "Lecturer" and so on. And Teacher_Heading is generally a sub-heading of Course_Name. Then, according to the relative locations of tokens in the *LSM* and other simple patterns (keywords included, font style, etc.), we can extract other course metadata successfully.

Heuristic rules used in the proposed approach for extracting other metadata based on the *LSM* of a course page are listed in priority order as follows.

For Course_Time:

1. Time token in the cluster including Course_ID or Course_Name token;
2. Time token with its parent cluster including Course_ID or Course_Name token;
3. Time token with its previous sibling cluster including Course_ID or Course_Name token.

For Teacher_Name:

1. Name token with its parent cluster including Teacher_Heading token, which is a token with a keyword of "Instructor", "Lecturer" and so on.;
2. Name token in the cluster including Course_Name token;
3. Name token with its parent cluster including Course_Name token;
4. Name token with its previous sibling cluster including Course_Name token.

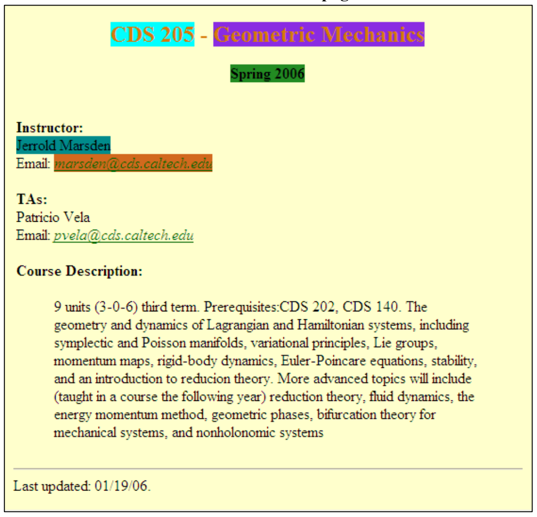
For Teacher_Email:

1. Email token with its parent cluster including Teacher_Heading token;
2. Email token with its parent cluster including Teacher_Email_Heading token, which is a token with a keyword of "Email" and its parent cluster including Teacher_Heading token;
3. Email token in the cluster including Teacher_Name token;
4. Email token with its parent cluster including Teacher_Name token;
5. Email token with its previous sibling cluster including Teacher _Name token.

Figure 3 shows the labeled sample course page and its *LSM*. And

Figure 4 shows some automatically labeled course pages with various metadata patterns.

The labeled course page



CDS 205 - Geometric Mechanics

Spring 2006

Instructor:
Jerrold Marsden
Email: marsden@cds.caltech.edu

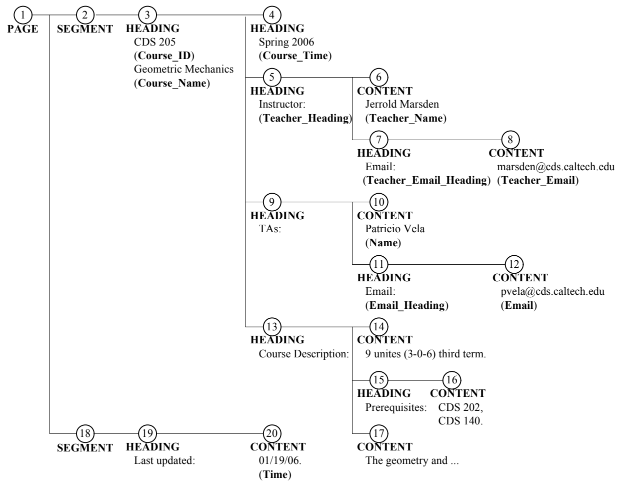
TAs:
Patricio Vela
Email: pvela@cds.caltech.edu

Course Description:

9 units (3-0-6) third term. Prerequisites:CDS 202, CDS 140. The geometry and dynamics of Lagrangian and Hamiltonian systems, including symplectic and Poisson manifolds, variational principles, Lie groups, momentum maps, rigid-body dynamics, Euler-Poincare equations, stability, and an introduction to reducion theory. More advanced topics will include (taught in a course the following year) reduction theory, fluid dynamics, the energy momentum method, geometric phases, bifurcation theory for mechanical systems, and nonholonomic systems

Last updated: 01/19/06.

Labeled Logical Structure Model



1 PAGE
2 SEGMENT
3 HEADING CDS 205 (Course_ID) Geometric Mechanics (Course_Name)
4 HEADING Spring 2006 (Course_Time)
5 HEADING Instructor: (Teacher_Heading)
6 CONTENT Jerrold Marsden (Teacher_Name)
7 HEADING Email: (Teacher_Email_Heading)
8 CONTENT marsden@cds.caltech.edu (Teacher_Email)
9 HEADING TAs:
10 CONTENT Patricio Vela (Name)
11 HEADING Email: (Email_Heading)
12 CONTENT pvela@cds.caltech.edu (Email)
13 HEADING Course Description:
14 CONTENT 9 unites (3-0-6) third term.
15 HEADING Prerequisites:
16 CONTENT CDS 202, CDS 140.
17 CONTENT The geometry and ...
18 SEGMENT
19 HEADING Last updated:
20 CONTENT 01/19/06. (Time)

Figure 3. The labeled course page and its *LSM*.

5. PERFORMANCE EVALUATION

We have implemented the proposed course metadata extractor in Java (JRE 1.6.0.). The experiment was performed on a computer with Intel Pentium 1.86GHz CPU and 1.49GB RAM, running Microsoft Widows XP Professional with Service Pack 2. To evaluate the performance of the proposed solution, we randomly selected a total of 326 course pages from websites of five US universities (California Institute of Technology, University of California at Davis, University of California at Irvine, University of Connecticut and Washington State University) and manually labeled all required metadata in the course pages. Then we run the implemented metadata extractor to identify the course metadata in the course pages automatically. In addition, we modified the code of the HMM-Based Text Mining and Extraction Tool (http://www.cs.toronto.edu/~ssanner/software.html), which is an implementation of hierarchical hidden-Markov model (HMM) text extraction from web pages as proposed by [5]. And then, we run it to label the course metadata by using our dataset as both training set and testing set.

The performance of extractions is evaluated by *Precision*, *Recall* and *F*1 score. An extracted metadata is considered correct if it has the same content (ignore redundant white-space and special characters) as that in the corresponding labeled metadata. Then, the *Precision P* is defined as the number of correct extractions divided by the total number of extractions, while the *Recall R* is defined as the number of correct extractions divided by the total number of labeled course page. The *F*1 score is defined as $2PR$ / (P+R), i.e., the harmonic mean of P and R. In addition, we use *Total Accuracy* to evaluate the overall performance of these two solutions, which is defined as the number of correct extraction course page (i.e., course page with all valid metadata extracted correctly) divided by the total number of course pages.

Table 1 shows the experimental results, which have shown that the proposed LSM-based solution has achieved much better effectiveness for most course metadata than the HMM-based solution. The HMM-based solution always has a higher Recall value than the LSM-based solution, because it usually labels many tokens to cover most metadata, but most of labeled tokens are not correct metadata.

Table 1. The experimental results.

Metadata	LSM-based			HMM-based		
	P	*R*	*F*1	*P*	*R*	*F*1
Course_ID	93.09%	86.81%	89.84%	30.70%	89.90%	45.80%
Course_Name	77.30%	77.30%	77.30%	32.10%	98.60%	48.50%
Course_Time	66.88%	73.05%	69.83%	19.20%	95.50%	32.00%
Teacher_Name	64.43%	55.80%	59.81%	58.60%	96.20%	72.90%
Teacher_Email	79.25%	65.63%	71.79%	19.00%	94.30%	31.60%
Total Accuracy	50.61%			6.1%		

6. CONCLUSIONS

In this paper, we proposed a novel hierarchical clustering approach to generate Logical Structure Model of a web page from the DOM tree, which is then used to facilitate extracting metadata from course pages. The experimental results have shown that our solution can achieve effective extractions for course metadata.

We will continue improvement for the course metadata extractor, which includes enhancing the precision and recall of extractions, extending to other course metadata, such as Course_Intro, Course_Outline, Course_Resource and Course_Literature, and integrating with other modules in Online Course Organization to label a larger amount of course pages automatically. In addition, we also plan to extend our solution to other specific domains, such as online news and product information, to extract useful metadata, which can directly benefit HP's business.

7. ACKNOWLEDGMENTS

This work is supported by HP Labs China under the project "Online Course Organization", as well as the National Natural Science Foundation of China under Grant No. 90412010.

8. REFERENCES

[1] Kushmerick, N., Weil, D., and Doorenbos, R. Wrapper induction for information extraction. In Proc. of Intl. Joint Conf. on Artificial Intelligence (IJCAI), pp729-735, 1997.

[2] Muslea, I., Minton, S., and Knoblock, C. A. Hierarchical wrapper induction for semistructured information sources. *Autonomous Agents and Multi-Agent Systems*. Vol. 4, No. 1-2, pp93-114, 2001.

[3] Freitag, D. Information extraction from html: Application of a general learning approach. In Proc. of the 15th Conf. on Artificial Intelligence (AAAI), pp517-523, 1998.

[4] Freitag, D., and Kushmerick, N. Boosted wrapper induction. In Proc. of the 17th National Conf. on Artificial Intelligence (AAAI), pp577-583, 2000.

[5] Freitag, D., and McCallum, A. Information extraction with HMM and shrinkage. In Proc. of the 15th National Conf. on Artificial Intelligence (AAAI), pp31-36, 1998.

[6] Yin, P., Zhang, M., Deng, Z., and Yang, D. Metadata extraction from bibliographies using Bigram HMM. In Proc. Of the 7th Intl. Conf. on Asian Digital Libraries (ICADL), pp310-319, 2004.

[7] Peng, F. and McCallum, A. Information extraction from research papers using conditional random fields. *Information Processing Management*. Vol. 42, No. 4, pp963-979, 2006.

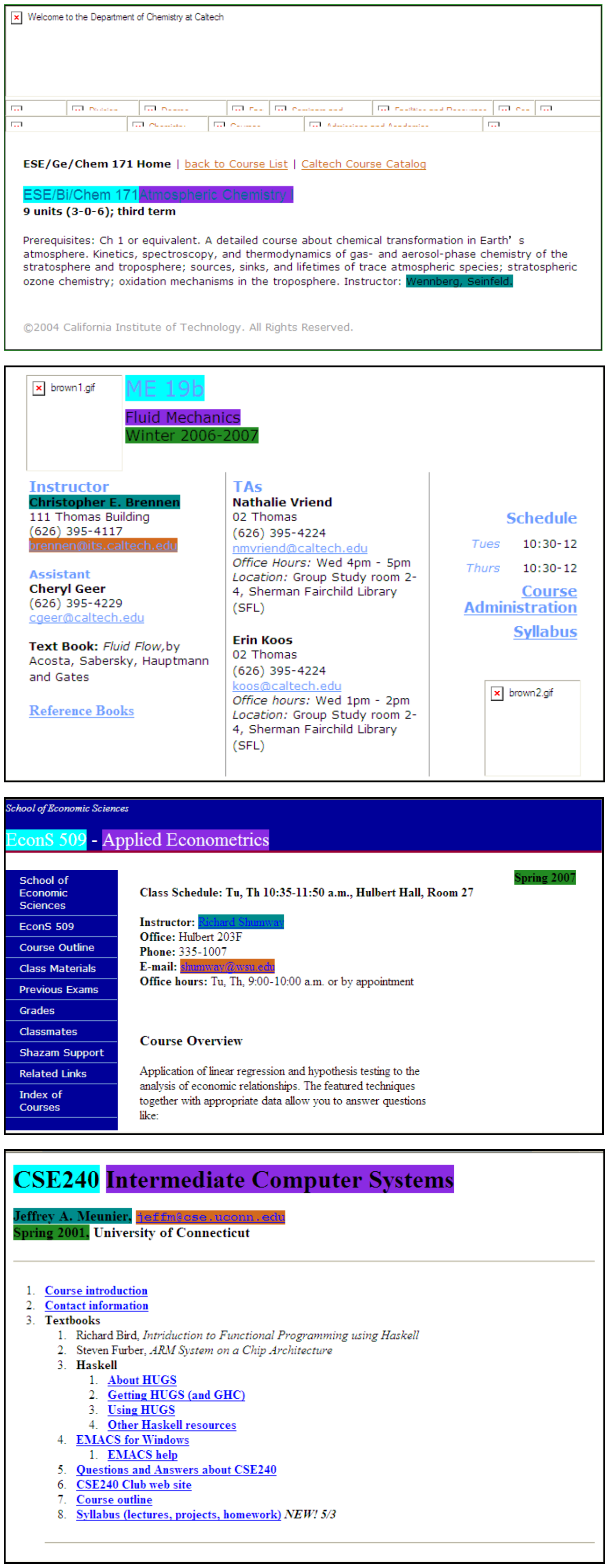
[8] Zhu, J., Nie, Z., Wen, J., Zhang, B., and Ma, W. 2D Conditional Random Fields for web information extraction. In Proc. of the 22nd Intl. Conf. on Machine Learning (ICML), pp1044-1051, 2005.

[9] Zhu, J., Nie, Z., Wen, J., Zhang, B., and Ma, W. Simultaneous record detection and attribute labeling in web data extraction. In Proc. of the 12th ACM Intl. Conf. on Knowledge Discovery and Data Mining (SIGKDD), pp494-503, 2006.

[10] Chen, J., Zhou, B., Shi, J., Zhang, H., and Wu, Q. Function-based object model towards website adaptation. In Proc. of the 10th Intl. Conf. on World Wide Web (WWW), pp587-596, 2001.

[11] Cai, D., Yu, S., Wen, J., and Ma W. VIPS: a vision-based page segmentation algorithm, *Microsoft Technical Report*, MSR-TR-2003-79, 2003.

[12] Xiang, P., Yang, X., and Shi, Y. Effective page segmentation combining pattern analysis and visual separators for browsing on small screens. In Proc. of the 2006 IEEE/WIC/ACM Intl. Conf. on Web Intelligence, pp831-840, 2006.



Welcome to the Department of Chemistry at Caltech

ESE/Ge/Chem 171 Home | back to Course List | Caltech Course Catalog

ESE/Bi/Chem 171Atmospheric Chemistry I
9 units (3-0-6); third term

Prerequisites: Ch 1 or equivalent. A detailed course about chemical transformation in Earth' s atmosphere. Kinetics, spectroscopy, and thermodynamics of gas- and aerosol-phase chemistry of the stratosphere and troposphere; sources, sinks, and lifetimes of trace atmospheric species; stratospheric ozone chemistry; oxidation mechanisms in the troposphere. Instructor: Wennberg, Seinfeld.

©2004 California Institute of Technology. All Rights Reserved.

brown1.gif

ME 19b
Fluid Mechanics
Winter 2006-2007

Instructor
Christopher E. Brennen
111 Thomas Building
(626) 395-4117
brennen@its.caltech.edu

Assistant
Cheryl Geer
(626) 395-4229
cgeer@caltech.edu

Text Book: *Fluid Flow*,by Acosta, Sabersky, Hauptmann and Gates

Reference Books

TAs
Nathalie Vriend
02 Thomas
(626) 395-4224
nmvriend@caltech.edu
Office Hours: Wed 4pm - 5pm
Location: Group Study room 2-4, Sherman Fairchild Library (SFL)

Erin Koos
02 Thomas
(626) 395-4224
koos@caltech.edu
Office hours: Wed 1pm - 2pm
Location: Group Study room 2-4, Sherman Fairchild Library (SFL)

Schedule
Tues 10:30-12
Thurs 10:30-12
Course Administration
Syllabus

brown2.gif

School of Economic Sciences

EconS 509 - Applied Econometrics

School of Economic Sciences
EconS 509
Course Outline
Class Materials
Previous Exams
Grades
Classmates
Shazam Support
Related Links
Index of Courses

Spring 2007

Class Schedule: Tu, Th 10:35-11:50 a.m., Hulbert Hall, Room 27

Instructor: Richard Shumway
Office: Hulbert 203F
Phone: 335-1007
E-mail: shumway@wsu.edu
Office hours: Tu, Th, 9:00-10:00 a.m. or by appointment

Course Overview

Application of linear regression and hypothesis testing to the analysis of economic relationships. The featured techniques together with appropriate data allow you to answer questions like:

CSE240 Intermediate Computer Systems

Jeffrey A. Meunier, jeffm@cse.uconn.edu
Spring 2001, University of Connecticut

1. Course introduction
2. Contact information
3. Textbooks
    1. Richard Bird, *Intriduction to Functional Programming using Haskell*
    2. Steven Furber, *ARM System on a Chip Architecture*
    3. Haskell
        1. About HUGS
        2. Getting HUGS (and GHC)
        3. Using HUGS
        4. Other Haskell resources
    4. EMACS for Windows
        1. EMACS help
    5. Questions and Answers about CSE240
    6. CSE240 Club web site
    7. Course outline
    8. Syllabus (lectures, projects, homework) *NEW! 5/3*

Figure 4. Some automatically labeled course pages.