Discovery of concept entities from web sites using web unit mining

Yin, Ming; Goh, Dion Hoe-Lian; Lim, Ee Peng; Sun, Aixin

2005

Yin, M., Goh, H. L., Lim, E P., & Sun, A. (2005). Discovery of concept entities from web sites using web unit mining. International Journal of Web Information Systems, 20, 1-13.

https://hdl.handle.net/10356/93769

Discovery of Concept Entities from Web Sites using Web Unit Mining

Ming Yin
Division of Information Studies
School of Communication & Information
Nanyang Technological University, Singapore 639798
Email: yinm0001@ntu.edu.sg
Dion Hoe-Lian Goh
Division of Information Studies
School of Communication & Information
Nanyang Technological University, Singapore 639798
Email: ashlgoh@ntu.edu.sg
Ee-Peng Lim
Centre for Advanced Information Systems
School of Computer Engineering
Nanyang Technological University, Singapore 639798
Email:aseplim@ntu.edu.sg
Aixin Sun
School of Computer Science and Engineering
University of New South Wales
Sydney, NSW, Australia 2052
Email: aixinsun@cse.unsw.edu.au

Received: January XX 2005; revised: November XX 2005

***Abstract*— A web site usually contains a large number of concept entities, each consisting of one or more web pages connected by hyperlinks. In order to discover these concept entities for more expressive web site queries and other applications, the *web unit mining* problem has been proposed. Web unit mining aims to determine web pages that constitute a concept entity and classify concept entities into categories. Nevertheless, the performance of an existing web unit mining algorithm, iWUM, suffers as it may create more than one web unit (*incomplete* web units) from a single concept entity. This paper presents two methods to solve this problem. The first method introduces a more effective web fragment construction method so as reduce later classification errors. The second method incorporates site-specific knowledge to discover and handle *incomplete* web units. Experiments show that *incomplete* web units can be removed and overall accuracy has been significantly improved, especially on the precision and $F1$ measures.**

***Index Terms*— Web classification, Web information organization**

I. INTRODUCTION

A. Motivation

As the World Wide Web grows exponentially, search engines have become important tools to access web information. A large portion of web search activities aim to locate a set of concept entities relevant to the user query. Imagine a graduate student who wants to know about the field of Information Retrieval (IR). Expected concept entities include: an IR course, researchers specialized in IR research, and related conferences. However, in the hypertext environment, a concept entity is sometimes represented by a set of connected web pages instead of a single web page. For example, a concept entity of a professor includes not only his/her homepage but also other pages describing his/her research interests, teaching activities, or curriculum vitae. Concept entities widely exist in web sites such as company web sites hosting staff web pages, university web sites publishing courses, faculty and research information.

Recently, Sun and Lim [1] proposed the concepts of *web unit* and *web unit mining* problem. A web unit is a set of web pages that jointly provide information about a concept entity. For example, the set of web pages describing a professor mentioned previously constitute a *faculty* web unit.

Web unit mining consists of two sub-problems, namely *web unit construction* and *web unit classification*. In the former, web pages representing a single concept entity are identified so as to form a web unit. The latter involves assigning web units correct concept labels. Sun and Lim [1] proposed an algorithm called iterative Web Unit Mining (iWUM). iWUM carries out web unit construction and web unit classification in an iterative

manner. In this approach, web unit classification results affect web unit construction, and vice versa. Unfortunately, a web unit classification error may cause errors in the next step of web unit construction. One possible problem is that it might create more than one web unit (*incomplete* web unit) from a single concept entity. Each *incomplete* web unit contains incomplete information about a concept entity.

In order to address the *incomplete* web unit problem and to improve web unit mining accuracy, we propose two methods to enhance iWUM. In the first method, we aim to improve the web unit construction before the first classification step. In the second method, we aim to discover and handle *incomplete* web units produced by iWUM.

B. Contributions

We summarize our contribution in improving the iterative Web Unit Mining (iWUM) problem as follows:

- ***Enhanced web fragment generation***
  A new web fragment generation method is proposed to replace the one used by iWUM. It reduces the number of web fragments created so as to reduce possible errors in later classification steps. As a result, the chances of creating *incomplete* web units have been greatly reduced.
- ***Knowledge-based web unit optimization***
  Incomplete web units are distributed in a web site with some discernable patterns. A web unit optimization method is proposed to detect and remove those *incomplete* web units created by web unit mining.

We also find out that these two methods can be integrated together and deliver even better results.

C. Paper Outline

The rest of this paper is organized as follows. In Section II, we give a detailed discussion on the web unit mining problem and explain how *incomplete* web units are produced by iWUM. Related work in web unit mining is surveyed in Section III. In Section IV and V, we propose the enhanced web fragment construction method and the kWUM method. An integrated method is discussed in Section VI. Our experiments and results are presented and discussed in Section VII. Finally, we conclude this paper in Section VIII.

II. Background: Web Unit Mining

A web unit is a set of web pages that jointly provides information about a concept entity [1]. It consists of exactly one *key page* and zero or more *support pages*. Consider the *course* web unit shown in Figure 1. The first page is the course's (CS100) homepage and the others provide supplementary information of the course. The homepage is the entry point to all information about the course and thus the key page; the others are support pages. Similarly, for a *faculty* web unit, the key page is a faculty's homepage; support pages include those pages about his/her research interests, teaching activities, and so on.

If a set of web pages that jointly provides information about a concept entity is constructed into more than one web

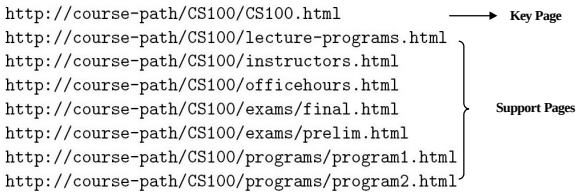

Fig. 1. An example web unit for course CS100

unit, each of them is regarded as an *incomplete* web unit. For example, the CS100 web unit contains a set of eight web pages, which are denoted as $p_1, p_2, \ldots, p_8$ (according to their listed order in Fiqure 1). If three web units are created from them, say $u_1 = \{p_1, p_2, p_3, p_4\}$, $u_2 = \{p_5, p_6\}$ and $u_3 = \{p_7, p_8\}$, they are *incomplete* web units. An *incomplete* web unit contains incomplete information about a concept entity. It does not comply with the definition of web unit, which requires a single concept instance to be represented by one web unit. Note that *incomplete* web units are outcomes of web unit mining and considered as errors.

The existence of *incomplete* web units does harmful to using web units to other applications. One important application is to allow search engines to index web information at the web unit level instead of the web page level. For example, the CS100 course entity can be indexed as a single unit (if the CS100 web unit is successful constructed). However, if three *incomplete* web units are created from the CS100 course entity, we need to index three units although all of them represent CS100. It is not efficient for search engines. Even worse, since each *incomplete* web unit contains incomplete information about CS100, it might cause search engines to fail to retrieve them for some user queries.

The web unit mining has two sub-problems, namely *web unit construction* and *web unit classification*. iWUM addresses these two sub-problems in an iterative manner. It first groups closely-related web pages (based on hyperlink connectivity) into small units (web fragments), which are then classified and merged with one another to form large web units. This classifying-merging procedure repeats until there is no change of category labels assigned to the web units. The rules of merging allow a single *labeled* web unit to merge with neighboring *unlabeled* web units, but not *labeled* web units. A *labeled* web unit refers to one that has been assigned a category label; otherwise it is called an *unlabeled* web unit. Note that those *unlabeled* web units are intermediate results of iWUM. They are expected to merge with neighboring *labeled* web units in the next web unit construction phase. When iWUM stops, those remaining *unlabeled* web units are discarded, only *labeled* web units are returned. In other words, web unit mining results are all *labeled* web units.

An example of how iWUM produces *incomplete* web units is given below, still considering the set of web pages constituting the CS100 web unit. Before conducting the first classification, iWUM groups them into a set of web fragments, say $f_1 = \{p_1, p_2, p_3\}, f_2 = \{p_4\}, f_3 = \{p_5, p_6\}, f_4 = \{p_7, p_8\}$. In order for iWUM to successfully merge f_1, f_2, f_3 and $f_4$into a single web unit, say u, the following two conditions are

required:

1) The web fragment containing the key page of u, f_1 in this case, should be classified to one pre-defined category.
2) Web fragments containing only support pages of u, f_2, f_3 and f_4 in this case, should not be classified to any pre-defined category but *unlabeled* instead.

If f_3, f_4 are classified to some pre-defined categories, they cannot be merged into u. Instead, three *incomplete* web units, $u_1 = \{p_1, p_2, p_3, p_4\}$, $u_2 = \{p_5, p_6\}$ and $u_3 = \{p_7, p_8\}$, might be created.

As a result, an error in web unit classification will lead to errors in web unit construction, resulting in *incomplete* web units. The more web pages of a web unit contains, the more web fragments are created during iWUM, and the higher the chance of classification error, therefore the more likely *incomplete* web units are created. Our initial experiment showed that among the web units mined by iWUM, 15% of them were *incomplete* web units.

III. Related Work

One sub-problem of web unit mining, *web unit classification*, is related to the web classification research, which aims to categorize web objects into pre-defined categories. Depending on what are defined as objects of interest, web classification can be carried out at different levels. Web page classification [2], [3], [4], [5], [6], [7], which attracts most research efforts, considers each web page as an object of interest. It utilizes both content features, e.g. title, body text, and other features, e.g. hyperlink structure, to conduct classification. Meanwhile web site classification [8], [9], [10], treats an entire web site as a single object of interest. Different from these previous work, web unit mining conducts classification at the web unit level.

In both web page and web site classification, the objects to be classified already exist: a web page could be determined by a URL and a web site could be determined by a domain name. However, this is not the case for web unit. Therefore another sub-problem of web unit mining, *web unit construction*, is to construct a set of web pages into a web unit. It is closely related to the hypertext research that explores ways to group a set or a subgraph of web pages into an information unit [11], [12], [13], [14]. Botafogo et al. [14] proposed to utilize graph theory to analyze hyperlink structure and identify *hypertext aggregates* from them. In [15], hypertext are divided into connected sub-graphs based on content-based similarities. [11] introduced the concept of *compound documents* and argued that information retrieval techniques are better suitable for working on *compound documents* than individual web pages. Although different definitions and approaches are used in these research, they are all based on the same underlying assumption: in hypertext environment, a complete unit of information is often presented by a set of web pages instead of a single web page. This assumption is also adopted by the web unit mining research. A notable difference between them and web unit mining is that information units are not associated with concept labels while web units are.

One major task of web unit construction is to identify the key page of a web unit. This is related to the home page finding problem [16], which is a special type of web search problem. There are studies on web features separating home pages from others. [17] divided URLs into four types, namely *root*, *sub-root*, *directory*, and *file*, and argued that web pages of the first three types are usually home pages. [18] mentioned that in-links anchor words and web page names are also useful in this task. Those techniques are helpful in identifying key pages of web units.

In our *enhanced web fragment generation* method, link structure has been extensively explored. Link analysis, which studies hyperlinks among web pages, is a very important research topic. [19] showed that link-based metrics, either simply in-degree or sophisticated HITS, PageRank, are effective in identifying high quality web pages, which could be used by search engines to improve their ranking strategies. Hyperlinks are created with many different purposes. [20] divided links into different levels such as page level, directory level, host level and domain level and argued that best performance is achieved when links of different levels are assigned different weights. In this research, hyperlinks are used to find closely connected web pages that possibly form a single information unit. We analyze hyperlinks within a web site (hierarchical structure) and divide them into different types.

IV. Enhanced Web Fragment Generation

A. Motivation

First, we discuss the influence of constructed web fragments to web unit construction. If a web unit u is initially divided into two web fragments, u can be successfully constructed when the two web fragments are correctly classified. However, if u is initially divided into four web fragments, it requires all the four web fragments to be correctly classified. It is obvious that the second situation is more prone to classification errors and *incomplete* web units are more likely to be produced.

In iWUM, the web fragment generation step computes a *connectivity index* score for each web folder. If the score is higher than a threshold, web pages in this web folder and its sub-folders are constructed as a web fragment. However, this method does not take into consideration different types hyperlink structures. It favors those sets of web pages that are fully-connected by hyperlinks. Furthermore, we also find that this method is unlikely to construct web fragments with web pages from multiple web folders. However, such multi-folder web fragments are desired in some situations.

One straightforward approach to solve the *incomplete* web unit problem is to reduce the number of web fragments generated from the set of web pages constituting a web unit. Imagine an ideal situation that we are able to generate one and only one web fragment for a web unit, the *incomplete* web unit problem never exists. However, this is too good to be true. Nevertheless, if we could generate web fragments closer to the corresponding web units, the chances of creating *incomplete* web units will be greatly reduced.

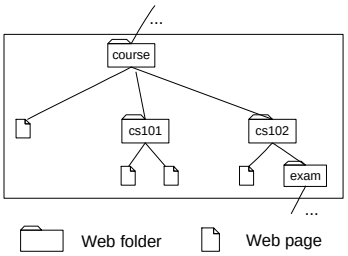

Fig. 2. URLs and Web Folders

B. Web Fragments

What is *web fragment*? According to [1], a web fragment is a set of closely-related web pages, which could be a potential web unit or a part of a web unit. Since web fragments are intermediate outcomes during iWUM, there is no clear definition about its size and scope. Just like a web unit, a web fragment also has one key page and zero or more support pages. The main difference among them is that each web unit has a concept label to indicate its category while a web fragment does not have. We believe that the concept of web fragment is similar to *compound document* [11], *logical domain* [12], *logical document* [13], and *aggregate* [14].

In order to construct web fragments from a collection of web pages, we mainly rely on two structural evidences: the hierarchical structure reflected by URLs (web directory) and link structure.

C. Web Directory

Given a collection of web pages from a web site, together with web folders extracted from them, a web directory is a tree structure with web folders as internal nodes and web pages as leaf nodes [1]. Note that a typical URL follows the syntax: *type://hostname [:port number] [/path][filename]*. Web folders can be extracted from the *hostname* and *path* components. As shown in Figure 2, two web folders are extracted from a *path* component */courses/cs101/*, namely *courses* and *cs101*, where *cs101* is a sub-folder of *courses*. Similarly, three web folders, *courses*, *cs101*, and *exam* are extracted from */courses/cs102/exam/*.

A web folder is usually created to contain a set of closely-related web pages. It is interesting to analyze how web pages in a web folder are connected with each other. We identify six connection patterns of a web folder. Note that here we are only interested in those web folders containing at least two web pages.

- ***Fully connected***
  There exist hyperlinks between any two web pages.
- ***Star***
  There exists one web page that has hyperlinks pointing to all other web pages.
- ***Linear***
  There exists a linear hyperlink path to connect all web pages.
- ***Hierarchical***
  There exists a hierarchical hyperlink structure to connect all web pages.
- ***Isolated***
  There does not exist any hyperlink between any two pages.
- ***Partially connected***
  There exist some hyperlinks to connect a few web pages, but cannot find a way to connect all web pages.

A web folder may satisfy more than one connection pattern. For example, *star* and *linear* patterns can be satisfied at the same time. To make it simple, when analyzing a web folder, we examine connection patterns in the order of *fully connected*, *star*, *linear*, *hierarchical*, *isolated*, and *partially connected* (from a more specific pattern to a more general one). If it satisfies *fully connected*, we stop; otherwise we move on to check *star*, and then *linear* and so on. As a result, for a web folder satisfying both *star* and *linear*, we simply regard it as *star*.

If a web folder contains only one web page or satisfies any of the four connection patterns, namely *fully connected*, *star*, *linear*, or *hierarchical*, we consider it as *well-connected*. In a *well-connected* web folder, there exists one entry page by which any other web page in the web folder (if existing) can be accessed without navigating through intermediate web pages outside the web folder. Such entry pages could be any node in *fully connected*, center node in *star*, first node in *linear*, or root node in *hierarchical*. Those entry pages are very likely to be the key pages of web fragments.

D. Link Structure

With the introduction of the web directory concept, we can divide hyperlinks into six different types. We analyze the relative locations of a link's source page and target page in the web directory. Let f_s and f_t denote the web folders containing a link's source page and target page respectively. Note that here we are only interested in hyperlinks within the web directory. Those hyperlinks involving web pages outside the web directory are ignored.

- ***From-parent links***
  A link that f_s is the parent folder of f_t.
- ***From-ancestor links***
  A link that f_s is the ancestor folder of f_t.
- ***Sibling links***
  A link whose source page and target page are in the same web folder.
- ***From-child links***
  A link that f_s is the child folder of f_t.
- ***From-descendant links***
  A link that f_s is the descendant folder of f_t.
- ***Across links***
  A link that f_s and f_t does not satisfy any of above conditions.

The first two types of links, *from-parent links* and *from-ancestor links* are mainly created to access web pages in lower levels of the web directory. *Sibling links* connect web pages in the same web folder. They account for the largest portion of hyperlinks within the web directory and are used to analyze the connection pattern of each web folder. *From-child links* and *from-descendant links* are mainly for the *back to*

home purpose. Unlike links of previous types that mainly help navigating web pages organized in a hierarchical structure, *across links* are more likely to indicate relationship between web pages. A typical *across link* is a link from a student's homepage to one of courses he/she takes. It is obvious that the student's homepage is not the common start point to access the course (except for the student). This link indicates there is a kind of relation between the student and the course (the student takes the course). An interesting observation is that the target page of an *across link* is often the entry page of a set of related web pages. When a student creates a link to a course, he/she tends to point to the homepage of the course instead of other web pages. This observation is also reported in [11].

Each type of link has different clues for identifying web fragments. For example, if a web page p is pointed by many *from-child links*, p together with web pages in its sub-folders are likely to form a web fragment. Web pages(s) in a *well-connected* web folder (connected by *sibling links*) are likely to form a web fragment. A web page pointed by many *across links* is likely to be the key page of a web fragment, but unlikely to be a support page of a web fragment. Furthermore, in a set of web pages $p_1, p_2, \ldots, p_n$, if there exists one and only one web page p_i that is pointed by many *across links*, $p_1, p_2, \ldots, p_n$ is likely to form a web fragment and p_i is the key page. Otherwise, if there does not exist such a p_i, $p_1, p_2, \ldots, p_n$ has no well recognized entry page and it might be a part of a web fragment (need to merge with others); if there exists more than one p_i, $p_1, p_2, \ldots, p_n$ is likely to contain more than one web fragment.

Algorithm 1 WebFragGen

```
Input:  web folder f
Output: web fragments wfl
1:  for each sub-folder f_i ∈ sub(f) do
2:    if f_i is unvisited then
3:      WebFragGen(f_i)
4:    end if
5:  end for
6:  flag := false
7:  mark f as visited
8:  if sub(f) == ∅ then
9:    if f is well-connected then
10:     construct a candidate fragment frag with pages in f
11:     flag = Is_Fragment(frag)
12:   end if
13: else
14:   if f is well-connected then
15:     construct a candidate fragment frag with pages in
        subT(f)
16:     flag = Is_Fragment(frag)
17:     if flag == false then
18:       construct a candidate fragment frag with pages in
          f
19:       flag = Is_Fragment(frag)
20:     end if
21:   end if
22: end if
23: if flag == true then
24:   add cf to wfl
25:   remove sub-tree of f, subT(f), from the web directory
26: end if
```

E. WebFragGen

After discussing the web directory and hyperlink structure, we characterize the web fragments we want to construct from three aspects as follows:

- ***Locality***
  A web fragment contains a set of web pages located near each other, in the same or neighboring web folder(s) in the web directory.
- ***Internal connectivity***
  Internal connectivity studies hyperlinks between web pages within a web fragment. A web fragment has a key page, through which all other pages in the fragment can be accessed, either directly or indirectly.
- ***External connectivity***
  External connectivity studies the hyperlinks that cross the boundary of a web fragment. We use the notation *external incoming link* to refer to a hyperlink from a web page outside of a web fragment to a web page inside the web fragment. If a web fragment has a large number external incoming links, most of those links should point to the key page, not to other web pages.

We propose a new web fragment construction algorithm, as shown in Algorithm 1. We create a web directory and run $WebFragGen(root)$, where $root$ is the root web folder of the web directory. Note that $sub(f)$ denotes the sub-folders of web folder f, $subT(f)$ denotes the subtree rooted at web folder f, including f, its sub-folders and descendent folders.

The web directory is traversed in a bottom-up manner. A web folder will be examined only after all its child web folders have been examined or it has no child web folder (Algorithm 1, Line 1-5). If a web folder f has no child web folder and is *well-connected*, a candidate web fragment $frag$ is created with all web pages in f. $frag$ is then checked by a function named $Is\_Fragment$, which determines whether $frag$ is a web fragment or not (Algorithm 1, Line 8-12). Once $Is\_Fragment$ returns *true* value, $frag$ is added to the web fragment list wfl and f is removed from the web directory (Algorithm 1, Line 23-26). If the currently-visited web folder f has some child web folders $sub(f)$, it indicates that web fragments are not found in $sub(f)$. We first create a candidate web fragment $frag$ with all pages in $subT(f)$ and test it. If it fails, we construct another candidate web fragment $frag$ with all pages in f and test it again (Algorithm 1, Line 14-21). Note that we gradually remove web folders(together with their sub-folders) from the web directory once web fragments are constructed.

When constructing a candidate web fragment, we also identify one candidate key page based on the connection pattern of the corresponding web folder. Here, center node

in *star*, first node in *linear*, or root node in *hierarchical* is the key page. For *fully-connected*, we cannot find one based on link structure information. Therefore a heuristic based on the name of the web page is utilized: a web page with the name such as "*default*", "*index*", "*home*" and "*welcome*" is the key page. For a candidate web fragment of pages from multiple web folders, we search the key page in the top web folder.

F. Web Fragment Classifiers

In the $WebFragGen$ algorithm, we propose a systematic way to construct candidate web fragments, identify possible key pages, and test them. However, *how to determine a candidate web fragment is a web fragment or not*, which is the function $Is\_Fragment$ supposed to handle, is not discussed yet. This is a difficult task. It might depend on many factors such as external connectivity, internal connectivity, the location in the web directory, number of pages, folders and so on. It is hard to design explicit rules or mathematical formula make a judgement. Therefore we adopt the machine learning approach: classifiers are trained by learning from a collection of sample web fragments and then they are used to determine wether a candidate web fragment is a web fragment or not. Note that they are binary classifiers, only 1 or 0 returned.

The number of pages in a web fragment varies. One could consist of only one web page, while another may consist of web pages from multiple web folders. We divide web fragments into three types.

- ***Single page web fragment***
  A web fragment consists of only one web page. Single page web fragments are usually constructed from those web folders containing only one web page.
- ***Single folder multi-page web fragment***
  A web fragment consists of more than one web pages, all of which are from the same web folder.
- ***Multi-folder web fragment***
  A web fragment consists of web pages from more than one web folder, usually a sub tree of web folders.

Different feature sets can be extracted from different types of web fragments. For example, a *single page web fragment* does not contain any support page. Therefore we can extract the number of external incoming links to support pages for the other two types of web fragments but not for *single-page web fragments*. Furthermore, we could even extract some features based on folder structure for *multi-folder web fragments*.

In order to improve classification accuracy, we create three classifiers, named *single page web fragment classifier*, *single folder multi-page web fragment classifier* and *multi-folder web fragment classifier*, to classify the three types of web fragments correspondingly. Although these three classifiers serve the same purpose–to determine whether a candidate web fragment is a web fragment or not, they are trained with different feature sets and by different training datasets.

G. Feature Sets

Table I lists the feature set for each type of web fragment classifier. The features are mainly based on hyperlink structure, which reflect the *internal connectivity* and *external connectivity* of a web fragment.

TABLE I
FEATURE SETS OF WEB FRAGMENT CLASSIFIERS

Classifier Type	Feature Set
Single page web fragment classifier	inlinks outlinks depth inLeafFolder
single folder multi-page web fragment classifier	external inlinks of the key page external inlinks of support pages internal inlinks of the key page depth # pages inLeafFolder
multi-folder web fragment classifier	external inlinks of the key page external inlinks of support pages internal inlinks of the key page depth # of pages # of folders depth of folder sub-tree max fan-out of folders

A *single page web fragment* consists of only one key page and zero support pages. The feature sets are rather simple:

- *Inlinks*: number of incoming links to the key page.
- *Outlinks*: number of outgoing links from the key page.
- *Depth*: depth of the web folder f (which contains the key page) in the web directory.
- *InLeafFolder*: 1 or 0 based on whether f is a leaf folder in the web directory (f has no child web folder).

A *single folder multi-page web fragment* consists of one key page and one or more support pages. The set of web pages can be treated as a whole and some features based on its internal and external connectivity can be extracted.

- *External inlinks of the key page*: number of incoming links from web pages outside of the web fragment to the key page.
- *External inlinks of support pages*: number of all incoming links from web pages outside of the web fragment to all support pages.
- *Internal inlinks of the key page*: number of incoming links from support pages to the key page
- *Depth*: same as that defined in *single-page web fragment*.
- *# Pages*: number of web pages in this web fragment.
- *InLeafFolder*: same as that defined in *single-page web fragment*.

The feature set of the *multi-folder web fragment classifier* contains all features used by the *single folder multi-page web fragments classifier* except the last one, *inLeafFolder*. It is obvious that the key page of a *multi-folder web fragment* is not in a leaf folder. As a result, we ignore it. Since a *multi-folder web fragment* contains a sub tree of web folders, a few additional features can be extracted as follows:

- *# of folders* : number of web folders in the web folder sub-tree.
- *Depth of folder sub-tree*: depth from the root of the sub-tree to the lowest web folder.
- *Max fan-out of folders*: the number of child web folders f_m has, where f_m is the web folder in the sub-tree that

has largest child web folders.

There are several features that count the number of links, such as *inlinks*, *outlinks*, *external inlinks* and so on. Note that we divide links into six types, *from-parent links*, *from-ancestor links*, *sibling links*, *from-child links*, *from-descendant links* and *across links*. We also argue that links of different types have different hints for identifying web fragments. Consider two *single-page web fragments*, $frag_1$ and $frag_2$, both of which have 5 incoming links. $frag_1$ has 1 *from-parent link* and 4 *from-child links* while $frag_2$ has 1 *from-parent link* and 4 *across links*. The 4 *from-child links* might indicate $frag_1$ needs to merge with some pages in its child folder to form a large web fragment; meanwhile the 4 *across links* might indicate that $frag_2$ is a good web fragment. Therefore if we count only the total number of inlinks, the difference between $frag_1$ and $frag_2$ is ignored. In order to keep this difference, which might be crucial for the classifier to make a correct decision, we count each type of links separately. As a result, we use six features instead of one. The inlinks of $frag_1$ and $frag_2$ are then represented as 1, 0, 0, 4, 0, 0 and 1, 0, 0, 0, 0, 4 respectively.

H. Normalization

These three classifiers are constructed base on SVM^{light}[21], which is an implementation of the Support Vector Machines (SVMs) — an accurate classifier widely used in various classification tasks. Since SVM^{light} requires that the value of each feature ranging from [0,1]. The normalization of some features is required.

- *Normalization of link count*
  For link count, we adopt a non-linear function $count_n = 1 - e^{-count/2}$, where $count_n$ is normalized score, and $count$ is the absolute link count value.
- *Normalization of* depth
  An average depth dep_{ave} is calculated from positive training samples. We measure the distance away from dep_{ave}. The function $dep_n = 1 - |dep - dep_{ave}|/dep_{max}$ is used, where dep_n is the normalized depth score, dep is the absolute depth value, and dep_{max} is the maximum absolute depth value.
- *Normalization of other attributes*
  For all other features, we adopt a simple linear normalization function $x_n = (x_n - x_{min})/(x_{max} - x_{min})$, where x_n is the normalized score, x is original feature value, x_{max} and x_{min} are maximum and minimum values of the feature.

I. Training Dataset Construction

The training datasets for these three classifier are extracted from our manually-constructed web units. In order to make constructed web fragments close to web units, we simply use those web units as positive training samples. Here the concept label associated with each web unit is ignored. Note that we can also divide web units into three types, *single page web unit*, *single folder multi-page web unit*, and *multi-folder web unit*, by the same rule we use to divide web fragments.

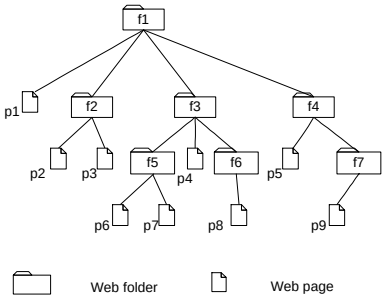

Fig. 3. A *multi-folder* web unit

Naturally, *single-page web units* are positive training samples for the *single page web fragment classifier*; so are *single folder multi-page web units*, *multi-folder web units* to the other two types of classifiers.

The negative training samples are not web units, but subsets of web units. They are extracted from *multi-folder web units*. Figure 3 shows a *multi-folder web unit* consisting of web pages $p_1, p_2, \ldots, p_9$. From this web unit, we extract seven subsets, which are used as negative training samples:

- *Negative single page web fragments*
  $\{p_4\}$, $\{p_5\}$, $\{p_8\}$, $\{p_9\}$.
- *Negative single folder multi-page web fragments*
  $\{p_2, p_3\}$ provided that f_2 well-connected,
  $\{p_6, p_7\}$ provided that f_5 well-connected.
- *Negative multi-folder web fragments*
  $\{p_4, p_6, p_7, p_8\}$ provided that f_3 well-connected,
  $\{p_5, p_9\}$ provided that f_4 well-connected.

Suppose that our $WebFragGen$ will construct $p_1, p_2, \ldots, p_9$ into a web fragment $frag$. During the construction process, web fragment classifiers will examine these seven subsets, testing whether they are web fragments by themselves. Only if all these tests return negative results, could $frag$ be eventually constructed. Therefore, these seven subsets are naturally negative training samples for our web fragment classifiers.

V. Knowledge-based Web Unit Mining (kWUM)

In this section, we propose a different approach to solve the *incomplete* web unit problem. We are interested in analyzing web units constructed by web unit mining. We aim to discover *incomplete* web units (if existing) and conduct web unit optimization to handle them. In this approach, site-specific knowledge about the constraints of web unit distribution in a web site are utilized.

A. Web Unit Distribution

We are interested in analyzing the distribution of web units within a web site. Therefore a web directory is constructed with web units and web folders extracted from them. Unlike the web directory mentioned in previous section that constituting web pages and web folder, the one constructed here consists of web units and web folders. An example web directory is shown in Figure 4.

- **Observation 1.** Web units are not evenly distributed in a web directory. Instead, most web units are distributed

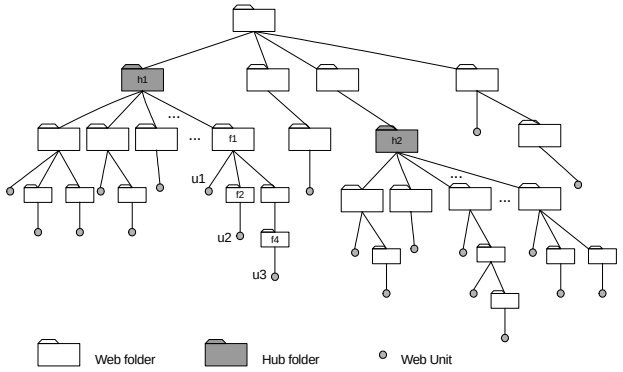

Fig. 4. Web directory and web units

under a few hub folders. A hub folder is one that has many sub-folders containing web units.

For example, a university web site will give each faculty, student, or course a folder to publish their web pages. Those folders are often located near to each other, likely under a common parent web folder, which is a *hub folder*. Sub-folders of a *hub folder* are the most likely places in a web directory to contain web units. In Figure 4, $h1$ and $h2$ are two example hub folders. Most web units in this web directory are located in the sub-folders of $h1$ or $h2$.

- **Observation 2.** *Incomplete* web units are likely located in lower-level web folders under sub-folders of a hub folder.

The key page of a web unit is usually at the highest-level web folder compared to support pages. Since *incomplete* web units consists of only support pages of a web unit, they tend to be located at lower-level web folders.

- **Observation 3.** *Incomplete* web units are either located in the same web folder or form *invalid* parent-child pairs.

A set of *incomplete* web units that split a single concept entity are located near each other. Some are even in the same web folder. Others are in different web folder and form *invalid* parent-child pairs. Two web units, e.g. u_i and u_j, can form a parent-child pair if the web folder containing u_j is a sub-folder or descendent folder of the web folder containing u_i. This parent-child web unit pair is denoted as $pc(u_i, u_j)$. In Figure 4, $pc(u_1, u_2)$ and $pc(u_1, u_3)$ are parent-child web unit pairs. Based on web units' category labels, we have different types of parent-child web unit pairs such as *faculty-course* pairs, *faculty-faculty* pairs, and so on. We then divide them into two groups: *valid* or *invalid*, based on the site-specific knowledge described in Section V-B.

B. Site-specific Knowledge

Given a web site and a set of concepts relevant to the web site, there are some constraints on how the concept entities are located in the web directory. We model these constraints as invalid parent-child concept pairs and call them *site-specific knowledge*. Each invalid parent-child concept pair suggests that it is highly unlikely to have an entity of the child concept be found in the sub-folder or descendent folder of the web folder containing an entity of the parent concept. The set of invalid parent-child concept pairs varies from web site to web site. Consider university web sites that contain three categories of web units, *faculty*, *student*, and *course*. *Faculty-faculty* and *faculty-student* concept pairs are invalid concept pairs as it is unlikely that a faculty or a student puts his/her own home page under another faculty's home page. For some specific university web sites, more invalid parent-child concept pairs may be specified. If a university web site W_1 has dedicated web space for course materials, it is unlikely that course web pages are stored under the web folder assigned to faculty. As a result, *faculty-course* concept pairs is invalid for W_1. We assume that this site-specific knowledge about invalid parent-child concept pairs of a web site is provided by a domain expert who is familiar with that web site.

C. kWUM Algorithm

The details of kWUM are given in Algorithm 2. kWUM requires three inputs, *a collection of web pages from a web site*, *a set of category labels*, and the *parent-child pair validity table* that represents the site-specific knowledge.

Algorithm 2 kWUM Algorithm

```
Input:  A collection of web pages from a web site,
        A set of category labels,
        Parent-child pair validity table
Output: Web units
1: generate an initial set of web units by applying iWUM, {u}
2: build a web directory with web units in {u}, root as the root folder
3: hfl =FindHubFolders(root)
4: for each hub folder hf ∈ hfl do
5:    for each sub-folder sh_i ∈ sub(hf) do
6:       merge multiple web units within the same web folders under subT(sh_i)
7:       RemoveInvalidPair(sh_i)
8:    end for
9: end for
```

We first utilize iWUM to create a preliminary set of web units (Algorithm 2, Line 1). We then discover and handle *incomplete* web units based on web structure and site-specific knowledge. A web directory is built with web units produced by iWUM (Algorithm 2, Line 2). Web folders whose sub trees contain many web units, or *hub folders*, are discovered by $FindHubFolders$ (Algorithm 2, Line 3). A hub folder is one that has many sub-folders possibly containing *incomplete* web units. We then discover and handle *incomplete* web units in each sub-folder of the hub folders (Algorithm 2, Line 4-9). For each sub-folder of hub folders, sh_i, we first merge web units within the same web folder under $subT(sh_i)$ (Algorithm 2, Line 6). The key page of each merged web unit is identified based on filenames and hyperlinks. Then we carry out the removal of invalid parent-child web unit pairs (Algorithm 2, Line 7).

As shown in Algorithm 3, hub folders are found by traversing the web directory in a breadth-first manner, where web folder $root$ is examined first, followed by its sub-folders and descendent folders. For each web folder f, $E(f)$ is calculated as defined in Equation 1 (Algorithm 3, Line 6). Note that $N(f)$ denotes the number of labeled web units under $subT(f)$. We set a minimum threshold E_{min} such that web folders satisfying $E(f) \geq E_{min}$ will be determined as hub folders. If f is a hub folder, it is inserted into a hub folder list, hfl (Algorithm 3,

Algorithm 3 FindHubFolders

```
Input:  web folder r
Output: list of hub folder
1: create an empty key folder list kfl
2: create an empty folder list fl
3: fl.push_back(root)
4: while |fl| > 0 do
5:    f=fl.pop_front()
6:    calculate E(f)
7:    if E(f) > E_min then
8:       kfl.push_back(f)
9:    else
10:      for each sub-folder cf_i ∈ sub(f) do
11:         fl.push_back(cf_i)
12:      end for
13:   end if
14: end while
15: return kfl
```

Line 8); otherwise, all sub-folders of f are inserted into a queue, fl, in order to find hub folders among them later (Algorithm 3, Line 10–12). Note that sub-folders of a hub folder will not be further examined. This is because we assume that no hub folders should be found under another hub folder. Finally, when fl is empty, all hub folders are stored in hfl.

$$E(f) = \begin{cases} -\sum_{cf_i \in sub(f)} \frac{N(cf_i)}{N(f)} * \log_2 \frac{N(cf_i)}{N(f)} \\ \qquad \text{if } f \text{ has sub-folders} \\ 0 \qquad \text{otherwise} \end{cases} \quad (1)$$

Algorithm 4 RemoveInvalidPair

```
Input:  Web folder sh_i
1: create an empty list fl
2: fl.push_back(sh_i)
3: while |fl| > 0 do
4:    f = fl.pop_front()
5:    get web unit u contained in f
6:    get the set of web units {cu} such that ∀cu_j ∈ {cu}, pc(u, cu_j)
7:    count number of valid and invalid parent-child pairs, num_v and num_i
8:    if num_v < num_i then
9:       if exists category cat for web unit u such that num_v >= num_i then
10:         assign u to category cat that maximizes num_v − num_i
11:      end if
12:   end if
13:   for each web unit cu_j ∈ {cu} do
14:      if pc(u, cu_j) is invalid pair then
15:         merge cu_j and its descendent web units with u
16:      else
17:         get web folder f_j that contains cu_j
18:         fl.push_back(f_j)
19:      end if
20:   end for
21: end while
```

The detailed removal procedure is shown in Algorithm 4. We take a top-down approach to examine each web folder f in the sub-tree of sh_i, $subT(sh_i)$. Let u be the web unit in f, all parent-child web unit pairs with u as the parent web unit are examined (Algorithm 4, Line 5–7). We conduct a majority voting between the number of valid pairs and invalid pairs, num_v and num_i. If $num_v \geq num_i$, the category label of u is assumed to be correct. Otherwise, we try to assign another category cat for u such that after re-computing the number of valid and invalid parent-child pairs, $num_v - num_i$ is maximized and non-negative. If such a cat is not found, u's category label is not changed (Algorithm 4, Line 8–12). For each child web unit of u, cu_j, if it forms an invalid parent-child pair with u, it is merged into u, u's key page unchanged (Algorithm 3, Line 15); otherwise the web folder containing cu_j, is inserted into a queue, fl (Algorithm 4, Line 17–18). In this way, all invalid parent-child pairs with u as the parent web unit are removed. We then pop up a web folder from fl and repeat the same processing. Finally, when fl is empty, all invalid pairs under $subT(sh_i)$ are removed.

VI. Integrated Approach

We have discussed two different methods to address the *incomplete* web unit problem. In Section IV, we propose a more effective web fragment construction method to reduce the number of web fragments, so as to reduce later classification errors. In Section V, we propose kWUM to discover and remove *invalid* parent-child web units pairs from web unit mining results, based on the observation that *incomplete* web units often form *invalid* parent-child pairs.

Note that these two methods are used in the different stages of the web unit mining process. The *enhanced web fragment generation method* is applied before the first classification is conducted, known as a *pre-classification* step. Meanwhile kWUM is applied after the last run of classification is finished, known as an *post-classification* step. The two methods can be used either individually or combined together to improve the web unit mining results. Therefore the integrated approach is to first use the *enhanced web fragment generation method* to improve web fragment generation step and then use kWUM to further improve web unit mining results. Since *enhanced web fragment generation method* and kWUM handle *incomplete* web units from different perspectives, we expect the integrated approach to deliver even better performance.

VII. Experiments

In this section, we conduct experiments on a dataset called UniKB to evaluate the *enhanced web fragment generation method*, kWUM, and the integrated method. Web units produced by incorporating these methods are compared with the original iWUM results. Note that experiment performance are measured by a variant of precision, recall, and F1 measure, as defined in [1] (described in Appendix A).

A. UniKB

Web unit mining was originally evaluated using the WebKB dataset[1]. WebKB is a widely used dataset for the web classification task, containing 4159 pages from four university web sites collected in 1997.

However, WebKB is a small and relatively old dataset. The small number of web pages in each web site cannot reflect present web sites of complex structures. Therefore, we create a new dataset, *UniKB*. In early April 2004, we downloaded web pages (HTML, HTM, SHTML) from two web sites: *www.cs.washington.edu* and *www.cs.utexas.edu* (Those two web sites are also used by WebKB). Web pages of the following types were then removed:

- *Web pages of restricted access*
  Web pages that contain error messages such as "HTTP 403" and "credentials required".

[1]http://www-2.cs.cmu.edu/ webkb/

TABLE II
UNIKB DATASET OVERVIEW

Concept / University	course		faculty		student		other
	u	*p*	*u*	*p*	*u*	*p*	*p*
Washington	989	16833	65	1307	201	2441	4941
Texas	243	4236	62	1994	173	2276	3599

- *Dynamic pages*
  Web pages whose URLs contain a "?" followed by a list of parameters.
- *PowerPoint slides*
  Web pages that are converted from *PowerPoint* slides, containing only images and no text content. They are unusable for web unit mining due to a lack of text.
- *Email archives*
  Web pages that contain only email archives.
- *Reference documentations*
  Web pages that contain stand-alone documentations such as *Java Documentation* and *Ant Manuals*.

After removal, a total of 25,522 and 11,105 web pages remained in Washington and Texas respectively. This collection of web pages is named as *UniKB*. Entities of three concepts, namely *course*, *faculty*, and *student* are manually identified in *UniKB* and web units are constructed accordingly. First, the homepage of each course, faculty, and student is identified. Then web units are constructed as follows: for a homepage like *[domain]/home/allen/index.htm*, web pages starting with *[domain]/home/allen/* form a web unit, say u_1, with the homepage as the key page. There is only one exception: if there exists another homepage, *[domain]/home/allen/cs101/index.htm*, web pages starting with *[domain]/home/ allen/cs101/* are no longer part of u_1, but form a new web unit, say u_2.

The statistics of manually-labeled web units in UniKB are shown in Table II, where u and p refer to the number of web units and web pages respectively; and $other$ refers to those web pages not belonging to any web unit. A web unit in UniKB has 16 web pages on average while a web unit in WebKB has only 4.

Note that both Washington and Texas contain a large number of *course* web units. This is because there might many different "versions" of a single course. For example, a course like *cs100* might have a *winter 2000* version, a *summer 99* version and so on. Different versions are independent of each other: each version has its own homepage and other pages describing course materials; each version has its own dedicated web space, e.g. *course/cs100/00wi* for the *winter 2000* version, *course/cs100/99su* for the *summer 99* version; there seldom exists hyperlinks connecting different versions. Based on the above situation, we treat each version as an individual web unit. This is the reason why there are 989 *course* in Washington.

After labeling web units, we analyzed their distribution in the web site structure and determined the validity of different types parent-child concept pairs. As shown in Table III, Washington has only one valid parent-child concept pair, *course-course*. This pair is valid because two course web units (different versions of a course) could form parent-child web unit pairs. Meanwhile Texas has two, *course-course* and *faculty-course*. Note that in Table III, 1 denotes a valid parent-child concept pair and 0 denotes an invalid one.

TABLE III
PARENT-CHILD VALIDITY TABLE

Washington

Parent unit	Child unit		
	course	faculty	student
course	1	0	0
faculty	0	0	0
student	0	0	0

Texas

Parent unit	Child unit		
	course	faculty	student
course	1	0	0
faculty	1	0	0
student	0	0	0

B. Web Fragment Generation

In order to evaluate the enhanced web fragment generation algorithm proposed in Section IV, we adopt two different methods to construct web fragments with web pages in UniKB. In method I, web fragments are constructed by the original iWUM web fragment generation step, which relies on the connectivity index and a few heuristics. This method could be regarded as a baseline method and be used to evaluate method II. Note that in the original iWUM approach, each input web page will be eventually associated with a web fragment. On the contrary, in $WebFragGen$, some web pages are not associated with any web fragment. In order to make a fair comparison, method II thus takes a mixed approach: first, web fragments are constructed by the $WebFragGen$ algorithm; second, the remaining web pages not belonging to any web fragment are fed to the original iWUM to construct web fragments again.

TABLE IV
WASHINGTON WEB FRAGMENTS

Method	WebFragGen		Connectivity Index		Web fragments
	f	*p*	*f*	*p*	*f*
I	-		15331	25522	15331
II	2223	23088	908	2434	3131

TABLE V
TEXAS WEB FRAGMENTS

Method	WebFragGen		Connectivity Index		Web fragments
	f	*p*	*f*	*p*	*f*
I	-		8656	11105	8656
II	838	10063	351	1042	1189

The web fragments constructed by these two methods are shown in Table IV (Washington) and Table V (Texas). As in Table IV, the total 25,522 web pages are constructed into 15,331 web fragments by method I. In method II, $WebFragGen$ constructs 2,223 web fragments with $23,088$ web pages. The remaining $2,434$ web pages are then constructed into 908 web fragments by the original iWUM. In both Washington and Texas, the original iWUM is less effective at constructing web fragments (less than 2 web pages per web fragment). As we expected, $WebFragGen$ greatly reduces the total number of web fragments constructed.

C. Results and Discussion

Our web unit mining experiments compared the web units mined by four different methods: 1)*iWUM*, 2)*kWUM*, 3)

EnhancedFrag, and 4) *Integrated*. The first two methods utilize web fragments constructed by method I while the last two methods utilize web fragments constructed by method II. Table VI shows the number of web units mined in each category by the four methods. The last column "labeled" displays the number of web units we manually labeled. Note that $< Cat > (W)$ and $< Cat > (T)$ denote the category Cat in Washington and Texas respectively. Web units mined by the last two methods are much less than those mined by the first two.

TABLE VI
MINED WEB UNITS

Category	iWUM	kWUM	EnhancedFrag	Integrated	**Labeled**
Course(W)	775	825	620	681	*989*
Faculty(W)	87	69	63	62	*65*
Student(W)	276	249	181	187	*201*
Course(T)	279	198	143	129	*243*
Faculty(T)	78	78	49	55	*62*
Student(T)	275	166	139	136	*173*
Total	1770	1585	1195	1250	*1733*

The precision (Pr), recall (Re), and $F1$ [2] value for each category of these four methods are shown in Table VII, in which the highest score of each measure for each category is displayed in bold font. The $F1$ values of the following 3 methods are compared with those of iWUM (the percentage improvement is shown in parenthesis).

Web units are extracted with reasonable accuracies using iWUM although the performance of different categories varies. In general, iWUM results in high recall but low precision. For all categories except $Course(W)$, Re scores are much better than Pr scores. The low precision of iWUM is mainly due to the existence of *incomplete* web units. Different from all others, *Course(W)* has a very high Pr (0.875), even better than the corresponding Re (0.686). We notice that Washington has dedicated web space for all course materials and the homepage of each course follows a similar style or template for its content. As a result, course web units are easily distinguished from others by iWUM, which determines concept labels mainly based on the key page content. On the contrary, in Texas, course materials are often embedded in faculties' web folders and there is no common style or content in their homepages. In summary, web site structure and key page content are the two major factors that affect the performance of iWUM. The iWUM results could be regarded as a baseline results to evaluate the performance of later three methods.

kWUM handles the *incomplete* web unit problem by removing *invalid* parent-child web unit pairs and thus improves the precision. For all categories except $Course(W)$, Pr scores from kWUM are significantly increased compared to those from iWUM. kWUM has mixed influence on recall. It significantly improves Re on $Faculty(T)$ (from 0.661 to 0.806), somewhat reduces Re on $Faculty(W)$ (from 0.800 to 0.785) and $Course(T)$ (from 0.777 to 0.669), and slightly improve Re for the other categories. The varied performance is due to that the web unit optimization method adopted by kWUM occasionally removes some good web units. Despite different influences in Pr and Re, the overall performance, which is measured by $F1$ value, is improved for the six categories.

In $EnhancedFrag$, a much smaller set of web fragments is constructed and replace those constructed by iWUM itself. In this approach, the *incomplete* web unit problem is handled by reducing the possible classification errors. Its influence on the web unit mining performances are two-fold and consistent for all the six categories: Pr scores are higher than those from iWUM or kWUM; Re scores are lower compared to iWUM or kWUM results. Considering the overall performance, for some categories, $F1$ values are higher than those from kWUM while for other categories, $F1$ values are even lower than those from iWUM. The improved precision scores are mainly because the number of *incomplete* web units have been greatly reduced through a more effective web fragment construction step. The reduced recall scores are mainly because $EnhancedFrag$ failed to identify some *low quality* web units as web fragments and considered them as part of other web fragments.

During the process of manually labeling web units, we notice that our labeled web units are of different *quality* or *authority*. Some web units contain more information than others. For example, a student/faculty web unit could have a lot of information about his/her study, research or publication while another one just briefly indicates he/she is a student/faculty. A course web unit of "current quarter" may contain more information than the same course of "winter 99", which is merely an archive and has become obsolete. It is not surprising that those high quality web units are more likely to be linked by many other web pages. Since our $EnhancedFrag$ utilizes linkage structure to identify web fragments, those high quality web units of many incoming links are likely to be constructed into web fragments. Meanwhile those less *well-known* web units are often ignored by $EnhancedFrag$. As a result, $EnhancedFrag$ tends to mine high quality web units successfully and ignore low quality ones. That is why its recall scores have been reduced.

In the *Integrated* method, kWUM is used to optimize the web units mined by $EnhancedFrag$. Just like what kWUM does to iWUM results, the *Integrated* method also has mixed influence on $EnhancedFrag$ results: it further improves the precision scores for all categories except $Course(W)$; meanwhile it improves the recall scores and the overall $F1$ scores for all categories except $Course(T)$.

Comparing the performance of these four methods for different categories, we notice that highest precision scores lie in the last two methods (5 in *Integrated* and 1 in $EnhancedFrag$); highest recall scores lies in first two methods (4 in kWUM and 2 in iWUM); highest $F1$ scores lies in second and fourth methods (4 in *Integrated* and 2 in kWUM).

Finally, we measure the overall performance of each method by *macro-average* and *micro-average*, denoted as $MacroAve$ and $MicroAve$. The *macro-average* weights equally all the six categories, regardless of how many web units belong to it. The *micro-average* weights equally all the web units. As shown in Table VII, for both *macro-average* and *micro-average*, the *Integrated* method delivers the best Pr and $F1$ scores and kWUM deliver the best Re scores.

[2] $F1 = 2 * Pr * Re/(Pr + Re)$

TABLE VII
COMPARISONS OF DIFFERENT WEB UNIT MINING METHODS

Concept	iWUM			kWUM			EnhancedFrag			Integrated		
	Pr	Re	$F1$	Pr	Re	$F1$	Pr	Re	$F1$	Pr	Re	$F1$
Course(W)	0.875	0.686	0.769	0.867	**0.723**	0.788 (2.6%)	**0.976**	0.611	0.751 (-2.3%)	0.971	0.667	**0.791 (2.9%)**
Faculty(W)	0.598	**0.800**	0.684	0.739	0.785	0.761 (11.3%)	0.778	0.754	0.766 (11.9%)	**0.790**	0.754	**0.772 (12.8%)**
Student(W)	0.438	0.599	0.506	0.514	**0.634**	0.568 (12.1%)	0.652	0.584	0.616 (21.7%)	**0.670**	0.624	**0.646 (27.6%)**
Course(T)	0.674	**0.777**	0.722	0.822	0.669	**0.738 (2.3%)**	0.890	0.529	0.665 (-7.9%)	**0.930**	0.496	0.647 (-10.4%)
Faculty(T)	0.526	0.661	0.586	0.641	**0.806**	**0.714 (22.0%)**	0.653	0.516	0.577 (-1.6%)	**0.709**	0.629	0.667 (13.8%)
Student(T)	0.447	0.711	0.549	0.747	**0.717**	0.732 (33.2%)	0.835	0.671	0.744 (35.4%)	**0.890**	0.699	**0.783 (42.6%)**
Macro ave	0.593	0.706	0.636	0.722	**0.722**	0.717 (12.7%)	0.798	0.611	0.686 (7.9%)	**0.827**	0.645	**0.718 (12.8%)**
Micro ave	0.680	0.694	0.687	0.777	**0.710**	0.742 (8.0%)	0.877	0.604	0.715 (4.2%)	**0.892**	0.643	**0.748 (8.8%)**

In a summary, both of the enhanced web fragment construction ($EnhancedFrag$) and kWUM can solve the *incomplete* web unit problem and thus greatly improve web unit mining precision. However, $EnhancedFrag$ usually reduces web unit mining recall as it may ignore some *low quality* web units while kWUM has mixed influence on recall. Furthermore, we explored the combination of these methods (*Integrated* method) and show that it delivers even better performance, achieving highest precisions for all categories and highest overall $F1$ for most categories.

VIII. CONCLUSION AND FUTURE WORK

Web unit mining aims to discover the set of web pages that represent a single concept entity from a web site. Mined web units are usually objects of interest for people who search in that web site, e.g. *courses*, *professors*, or *students* entities from a university web site. As a result, a major application of web units is to incorporate them into search engines so as to support better indexing and searching.

An existing web unit mining algorithm, iWUM, creates *incomplete* web units, with each covering only a subset of web pages of a single concept entity. The existence of *incomplete* web units does harmful to using web units to index web information. In order to address this problem, we have proposed two different methods, the *enhanced web fragment generation* method and the kWUM method. Experiments with the UniKB dataset show that a large portion of *incomplete* web units could be removed and web unit mining performance is thus improved, especially on the precision measures. The integration of these two methods delivers even better results. The removal of *incomplete* web units make mined web units more useful to web information organization and other applications.

Web unit mining is a new research field. There is still much room for further improvement. For example, as existing web unit mining algorithms heavily rely on web site structure to construct web units and key page content to classify web units, we need to conduct experiments with other web sites. More research efforts are required to improve web unit mining performance on those web sites not well-structured and on those web units with key pages of little content. In another direction of our future work, we focus on utilizing mined web units to enhance web information retrieval. We will explore ways for modeling web units and develop web unit-based ranking strategies.

REFERENCES

[1] A. Sun and E.-P. Lim, "Web unit mining: finding and classifying subgraphs of web pages," in *Proceedings of the twelfth international conference on Information and knowledge management*, 2003, pp. 108–115.

[2] S. Chakrabarti, B. Dom, and P. Indyk, "Enhanced hypertext categorization using hyperlinks," in *Proceedings of the 1998 ACM SIGMOD international conference on Management of Data*, 1998, pp. 307–318.

[3] M. Craven and S. Slattery, "Relational learning with statistical predicate invention: Better models for hypertext," *Journal of Machine Learning*, vol. 43, no. 1-2, pp. 97–119, 2001.

[4] J. Furnkranz, "Hyperlink ensembles: A case study in hypertext classification," *Journal of Information Fusion*, vol. 1, pp. 299–312, 2001.

[5] H.-J. Oh, S. H. Myaeng, and M.-H. Lee, "A practical hypertext catergorization method using links and incrementally available class information," in *Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval*, 2000, pp. 264–271.

[6] A. Sun, E.-P. Lim, and W.-K. Ng, "Web classification using support vector machine," in *Proceedings of the 4th international workshop on Web information and data management*, 2002, pp. 96–99.

[7] Y. Yang and X. Liu, "A re-examination of text categorization methods," in *Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 1999, pp. 42–49.

[8] M. Ester, H.-P. Kriegel, and M. Schubert, "Web site mining: a new way to spot competitors, customers and suppliers in the world wide web," in *Proceedings of the 8th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2002, pp. 249–258.

[9] Y. Tian, T. Huang, W. Gao, J. Cheng, and P. Kang, "Two-phase web site classification based on hidden markov tree models," in *Proceedings of IEEE/WIC Web Intelligence, Tianlun Dynasty Hotel, BEIJING, CHINA, October 13-17, 2003*, 2003.

[10] L. Terveen, W. Hill, and B. Amento, "Constructing, organizing, and visualizing collections of topically related web resources," *ACM Transactions on Computer-Human Interaction*, vol. 6, no. 1, pp. 67–94, 1999.

[11] N. Eiron and K. S. McCurley, "Untangling compound documents on the web." in *Hypertext*, 2003, pp. 85–94.

[12] W.-S. Li, O. Kolak, Q. Vu, and H. Takano, "Defining logical domains in a web site," in *Hypertext*, 2000, pp. 123–132.

[13] K. Tajima and K. Tanaka, "New techniques for the discovery of logical documents in web," in *DANTE*, 1999, pp. 125–132.

[14] R. A. Botafogo and B. Shneiderman, "Identifying aggregates in hypertext structures." in *Hypertext*, 1991, pp. 63–74.

[15] K. Tajima, Y. Mizuuchi, M. Kitagawa, and K. Tanaka, "Cut as a querying unit for www, netnews, e-mail." in *Hypertext*, 1998, pp. 235–244.

[16] D. Hawking and N. Craswell, "Overview of the trec-2001 web track." in *TREC*, 2001.

[17] W. Kraaij, T. Westerveld, and D. Hiemstra, "The importance of prior probabilities for entry page search." in *Proceedings of the 25th annual international conference on Research and developement in information retrieval*, 2002, pp. 27–34.

[18] N. Craswell, D. Hawking, and S. E. Robertson, "Effective site finding using link anchor information." in *Proceedings of the 24th annual international conference on Research and developement in information retrieval*, 2001, pp. 250–257.

[19] B. Amento, L. G. Terveen, and W. C. Hill, "Does "authority" mean quality? predicting expert quality ratings of web documents." in *Proceedings of the 23th annual international conference on Research and developement in information retrieval*, 2000, pp. 296–303.

[20] X.-M. Jiang, G.-R. Xue, W.-G. Song, H.-J. Zeng, Z. Chen, and W.-Y. Ma, "Exploiting pagerank at different block level." in *Proceedings of the 5th International Conference on Web Information Systems Engineering*, 2004, pp. 241–252.
[21] T. Joachims, *Making large-scale support vector machine learning practical*. MIT Press, 1999.

APPENDIX A - PERFORMANCE METRICS FOR WEB UNIT MINING

It is difficult to use the standard *precision* and *recall* measures to evaluate web unit mining performance. A web unit is set of web pages. A mined web unit could only partially match a labeled web units. Furthermore, the key page and support pages may have different importance.

TABLE VIII
CONTINGENCY TABLE FOR WEB UNIT u_i

Web unit evaluation		Perfect web unit u_i'		
		$u_i'.k$	$u_i'.s$	NU
Constructed web unit u_i	$u_i.k$	TK_i	SK_i	–
	$u_i.s$	KS_i	TS_i	FS_i
	NU	NK_i	NS_i	–

Given a web unit u_i constructed by a web unit mining method, we must first match it with an appropriate labeled web unit u_i', also known as the *perfect web unit*. We define u_i' to be the labeled web unit containing $u_i.k$ and u_i' has the same label as u_i; $u_i.k$ can be either the key page or a support page of u_i'. The contingency table for matching a web unit u_i with its perfect web unit u_i' is shown in Table VIII. Each table entry represents the overlapping web pages between the key/support pages of u_i and u_i'. For example $TK_i = \{u_i.k\} \cap \{u_i'.k\}$ and $TS_i = u_i.s \cap u_i'.s$. The entries in the last column and row account for pages that appear either in u_i or u_i', but not both. $FS_i = u_i.s - (u_i'.s \cup \{u_i'.k\})$. $NK_i = \{u_i'.k\} - \{u_i.k\} - u_i.s$. $NS_i = u_i'.s - \{u_i.k\} - u_i.s$. Note that $|TK_i| + |KS_i| + |NK_i| = 1$ and $|TK_i| + |SK_i| = 1$. If the perfect web unit for u_i does not exist, u_i is considered invalid and will be assigned zero precision and recall values. Otherwise, the *precision* and *recall* of a web unit, u_i, are defined as follows.

$$Pr_{u_i} = \frac{\alpha \cdot |TK_i| + (1-\alpha) \cdot |TS_i|}{\alpha + (1-\alpha) \cdot (|KS_i| + |TS_i| + |FS_i|)} \quad (2)$$

$$Re_{u_i} = \frac{\alpha \cdot |TK_i| + (1-\alpha) \cdot |TS_i|}{\alpha + (1-\alpha) \cdot (|SK_i| + |TS_i| + |NS_i|)} \quad (3)$$

To account for the importance of key pages, a *weight factor* α to represent the degree of importance is introduced. $|u|$ is the number of web pages in web unit u. If $\alpha = 1$, the importance of key page completely dominates over the support pages. By choosing a α value, we can assign appropriate importance to key and support pages in a web unit mining performance metric. More information about the performance metrics for web unit mining can be found in [1].