A Fully Automated Object Extraction System for the World Wide Web

David Buttler, Ling Liu, Calton Pu
Georgia Institute of Technology
College of Computing
Atlanta, GA 30332, U.S.A.
{buttler, lingliu, calton}@cc.gatech.edu

Abstract.

This paper presents a fully automated object extraction system – Omini. A distinct feature of Omini is the suite of algorithms and the automatically learned information extraction rules for discovering and extracting objects from dynamic Web pages or static Web pages that contain multiple object instances. We evaluated the system using more than 2,000 Web pages over 40 sites. It achieves 100% precision (returns only correct objects) and excellent recall (between 93% and 98%, with very few significant objects left out). The object boundary identification algorithms are fast, about 0.1 second per page with a simple optimization.

1. Introduction

The amount of information on the web is growing at an astonishing speed. Search engines and browsers have become ubiquitous tools for accessing and finding information on the Web. Not surprisingly, the explosive growth of the Web has made information search and extraction a harder problem than ever.

Several research efforts have attempted to extract content structure from Web documents. They divide a web page into *objects* of interest, either by programmer (in a way analogous to writing wrappers) [3, 8] or with some degree of automation. For example, [3, 8] discover object boundaries manually. They first examine the documents and find the HTML tags that separate the objects of interest, and then write a program to separate the object regions. Other approaches [1, 2, 5, 6, 10, 12, 13] separate object regions with some degree of automation. These approaches rely primarily on the use of syntactic knowledge, such as specific HTML tags, to identify object boundaries. Embley et al [6] developed an heuristics-based automated object extraction method. Their ontology heuristic plays a critical role in achieving high accuracy, but it is costly to develop (about 2-man weeks for a given web site) [6]. In short, common techniques used for constructing wrapper programs often require embedding programmers' understanding of the specific presentation layout or specific contents of the Web pages. This turns out to be labor intensive and error-prone, especially for the web sites that are frequently changing their information presence on the Web. As a result, most of the information integration services do not scale. They have hard time to effectively incorporate additional or new content providers into their existing integration access framework.

This paper presents a fully automated object extraction system called Omini. Omini parses web pages into tree structures and performs object extraction in two stages. First, it uses a set of subtree extraction algorithms to locate the smallest subtree that contains all the objects of interest (e.g., by ignoring advertisements). Second, it employs a suite of object extraction algorithms to find the correct object separator tags that can effectively separate objects. Both stages are fully automated. The subtree extraction stage considerably reduces the number of possibilities considered in the object extraction stage. The main contribution of this paper is the automated learning of rules for minimal subtree extraction and object boundary identification. We tested and evaluated the Omini approach by human users and in a wrapper generation system (the XWRAPElite [14] developed at Georgia Tech). We conducted a series of experiments over 2000 Web pages from 50 popular web sites. The results were consistent and satisfying. Omini reached a recall ratio in the range of 93% to 98% and precision ratio of 100% in these experiments.

Before explaining the details of our approach, we would like to note that fully automated approach to information extraction from Web pages is just one of the many challenges in building a scalable and reliable information search and aggregation service for the Web. The rest of the paper proceeds as follows. We briefly review a set of preliminary concepts in Section . We introduce the Omini system architecture in Section . Then we describe the object separator extraction algorithms in Section . Section describes a combined algorithm that unifies the five independent object extraction algorithms. Section reports the experiments and demonstrates the effectiveness of our object extraction approach through an analysis of our experimental results. We conclude the paper with a summary and an outline of future work in Section .

2. Preliminarie

2.1. Well-Formed Web Document

The web documents considered in this paper are HTML or XML documents. A web document consists of text and tags. A *tag* in a web document is marked by a tag name and an optional list of tag attributes enclosed in a pair of opening and closing brackets "<" and ">". *Text* is a sequence of characters in between two tags. By HTML and XML specification standard, most of tags in a Web document appear in pairs. A tag whose name does not start with a forward slash (i.e., "/") is called a *start tag*; otherwise it is called an *end tag* and the name of an end tag is the name of its corresponding start tag proceeded by "/". A web document is said to be **well-formed** if it satisfies the following conditions:

- There are no opening or closing brackets, < and >, in the text of the document that are not tags. Instead, these characters, when used in the text of a document, are encoded as < and >.
- All tags must be paired; namely every start tag has a corresponding end tag.
- All attribute values in a tag must be quoted (e.g.).
- All tags which do not normally have end-tags (such as *<IMG>, <HR>*, and *
*) are immediately followed by a corresponding end tag. For example: *
* will be denoted by *
</BR>*.
- Pairs of tags must be nested inside one another without overlapping. For example, the document fragment "*<a>* ... **...** ... **" is not well formed. The correct nesting for this example fragment is "*<a>* ... **...** ... **".

2.2. Tree Representation of Web Documents

A well-formed web document can be modeled as a *tag tree*. All the internal nodes of a tag tree are tag nodes and all leaf nodes are content nodes (numbers, strings, or other data types such as encoded MIME types). A *tag node* denotes the part of the web document identified by a start tag and its corresponding end tag and all characters in-between. A tag node is labeled by the name of the start tag. A *leaf node* denotes the content data (text) between a start tag and its corresponding end tag or between an end tag and the next start tag in a web document. A leaf node is labeled by its content. An example tag node in an HTML document is *<Title>* Home Page *</Title>*, where *<Title>* is the name of the tag node and the text string Home Page is a leaf node.

There is a path from the root node to every other node in the tree. For a given node, the path expression from the root of the tree to the node can uniquely identify the node. Therefore, in subsequent sections we sometimes use such a path expression to refer to the node. Consider Figure 1. The path from the root node *HTML* to the *Title* node goes through the *Head* node. It can be expressed using the dot notation as HTML[1].Head[1].Title[1]. The numbers in the brackets after each node denotes the order of the child in the tag tree. Similarly, the path from *HTML* to *Body* is HTML[1].Body[2].

Let $\mathcal{T} = (V, E)$ be a tag tree of a web document D where $V = V_T \cup V_C$, V_T is a finite set of tag nodes (internal nodes) and V_C is a finite set of content nodes (leaf nodes); $E \subset (V \times V)$, representing the directed edges. We call a subtree anchored at node u, $u \in V$, a *minimal subtree* with the property P, if it is the smallest subtree that meets the following condition: there is no other subtree, say $subtree(w), w \in V$, which satisfies both the property P and the condition that u is an ancestor of w. In Figure 1 there are two subtrees that contain all of the *hr* nodes, the subtree anchored at *HTML* and the subtree anchored at *Body*. The subtree anchored at *Body*, as shown in Figure 2, is the minimal subtree that contains all of the *hr* nodes.

In addition to the notion of minimal subtree, the following concepts are used frequently in the subsequent sections to describe the Omini object extraction algorithms.

- $parent(u)$: The parent node of u is defined as $parent(u) = \{w | w \in V, (w, u) \in E\}$. The root node of a tree $\mathcal{T}$ is the only node which does not have a parent node.
- $children(u)$: The parent node of u refers to the set of child nodes of u. $children(u) = \{w | w \in V, (u, w) \in E\}$. This definition says that a node w is a child node of u if and only if there exists an edge $(u, w) \in E$.
- $fanout(u)$: For any node $u \in V$, we use $fanout(u)$ to denote the cardinality of the set of children of u. $fanout(u) = \|children(u)\|$ if $u \in V_T$ and $fanout(u) = 0$ if $u \in V_C$.
- $nodeSize(u)$: For any node $u \in V$, if $u \in V_C$, i.e., u is a leaf node, then $nodeSize(u)$ denotes the content size in bytes of node u. Otherwise, u is a tag node, i.e., $u \in V_T$ and $fanout(u) > 0$. We define $nodeSize(u)$ to be the sum of the node sizes of all the leaf nodes reachable from node u, i.e. $nodeSize(u) = \sum_{v_i \in children(u)} (nodeSize(v_i))$. For any node $u \in V$, we define the size of the subtree anchored at node u, denoted by $subtreeSize(u)$, to be the node size of u. I.e., $subtreeSize(u) = nodeSize(u)$.
- $tagCount(u)$: For any node $u \in V$, if $u \in V_C$ is a leaf node, then $tagCount(u) = 0$. Otherwise, $u \in V_T$ is a tag node and $tagCount(u) = 1 + \sum_{v_i \in children(u)} (tagCount(v_i))$. $tagCount(u)$ refers to the total number of tag nodes of which u is an ancestor.

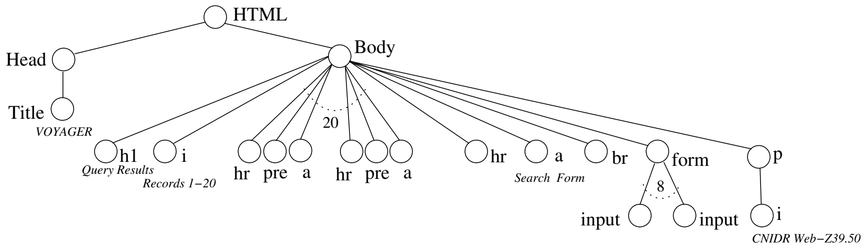

Figure 1: Tree Representation for Library Of Congress search results page.

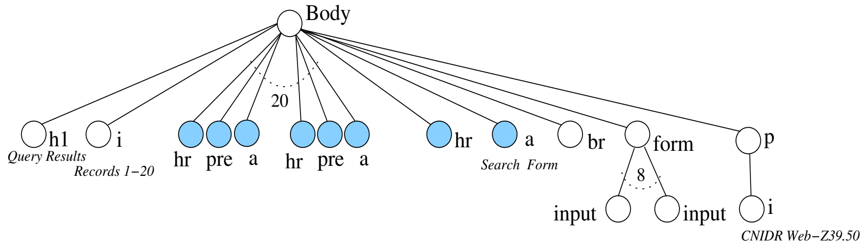

Figure 2: Minimal Subtree from Figure 1 with candidate object separator tags highlighted

3. System Architecture

Figure 3 shows the Omini system architecture. A user or an application may submit a URL to the Omini system to initiate the object extraction process. The results returned by Omini is a list of objects extracted from the given web page. The Omini object extraction process consists of three phases.

Phase 1: Preparing a web document for extraction
The phase one is dedicated to preparing the web document for extraction. It takes a URL from an end-user or an application, and performs the following three tasks: First, the Web page specified by the URL will be fetched from a remote site. Second, the fetched page will be cleaned using the syntactic normalization algorithm, which transforms the given web page into a well-formed web document. Third, the well-formed web document will be converted into a tag tree representation based on the nested structure of start and end tags.

Phase 2: Locating objects of interest in a Web page
The phase 2 process is divided into two consecutive steps: object-rich subtree extraction and object separator extraction. The former locates the minimal subtree that contains all the objects of interest in a page. The latter finds the object separator tags that can effectively separate objects.

Step 1: Object-rich subtree extraction
Web pages are designed for human browsing. In addition to the primary content regions, many web pages often contain other information such as advertisements, navigation links, and so on. Therefore, given a web document D, the first task in the object discovery phase is to identify which part of the document is the primary content region. Let T be the tag tree of the document D, then the task of locating the primary content region is reduced to the problem of locating the subtree of T which contains all the objects of interest. We call this task the *Object-rich subtree Discovery.* Obviously, there may be more than one subtree that contain all the objects of interest. The main goal of the object-rich subtree discovery is to locate the *minimal* subtree of T which contains all the objects of interest. In the first prototype of Omini, three individual subtree discovery rules and a method to combine them have been implemented. To choose the correct subtree we compare the fanout, the content size, or the tag count of all the subtrees in a given Web document. Consider the web document in Figure 4 and its tag tree in Figure 5. The objects that we are interested in extracting from this example Web document is obviously the search results, namely the twelve news items marked by the twelve tables at the right side of the tree. By applying the subtree heuristics, the minimal subtree that contains all the News objects of interest is the subtree anchored at the tag node $HTML[1].body[2].form[4]$.

Step 2: Object separator extraction
Once the primary content region is found, the next task is to decide how to separate data objects from each other, and from any other information in the web page. We refer to this task as the *Object separator discovery.* One goal of the object separator discovery is to develop a method that can fully automate the process of discovering the correct object separator, which will effectively separate objects in the primary content region and extract the objects of inter-

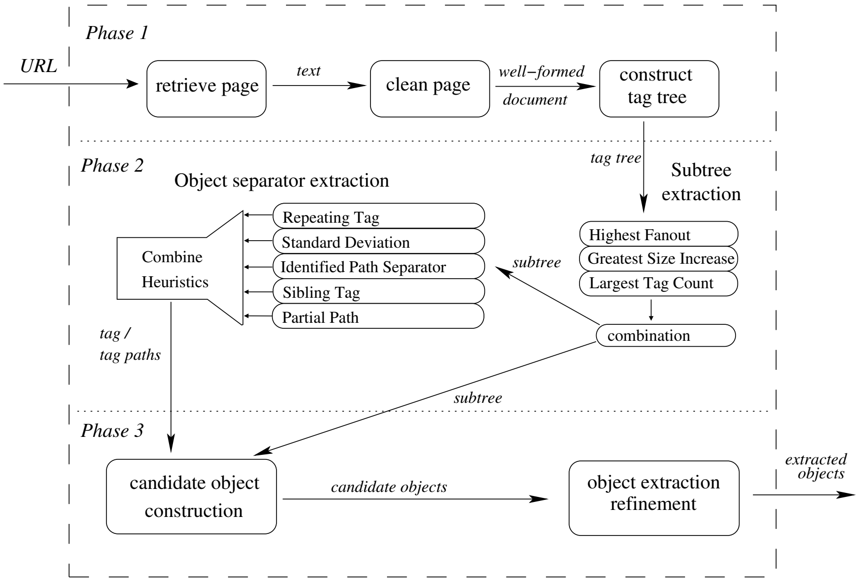

Figure 3: Omini System Architecture

est. To achieve this objective, we develop a set of individual algorithms, each of which can independently identify a ranked list of object separators, and provide a mechanism to combine these independent algorithms into a methodical approach for the object separator discovery.

Phase 3: Extracting objects of interest in a page
The phase 3 consists of two tasks: Candidate Object Construction and Object Extraction Refinement.

Candidate object construction is the process of extracting objects from the raw text data of the web document using the object separator tag identified in Phase 2. After the object separator tag is chosen the objects need to be extracted from the components of the chosen subtree. Sometimes the separator tag sits between objects, and other times it is the root of the object or a part of the object. Occasionally, an object may be broken down into two or more pieces by the chosen separator. When this happens, we need a mechanism to construct the whole object by merging the pieces together.

Object Extraction Refinement is the process of eliminating candidate objects that do not conform to the set of minimum criteria, which are derived by the object extraction process and satisfied by most of extracted objects. More concretely, in the process of constructing the objects, extraneous objects such as list headers or footers may be extracted occasionally. The object extraction refinement step will remove those objects that are structurally not of the same type as the majority of objects, such as an object that is missing a common set of tags, or that has too many unique tags. Also if the object is too small or too large it will be removed as well.

4. Object Separator Extraction

After the object-rich subtree extraction process, the problem of extracting the object separator tag in a web page is reduced to the problem of finding the correct object separator tag in the chosen minimal subtree. The problem can be addressed in two steps. First we need to decide which tags in the chosen minimal subtree should be considered as candidate object separator tags. Second, we need a method to identify the right object separator tag from the set of candidate tags, which will effectively separate all the objects.

There are several ways of choosing the object separator tags. One may consider every node in the chosen subtree as a candidate tag or just the child nodes of the chosen subtree as the candidate tags. Based on the semantics of the minimal object-rich subtree, it is sufficient to consider only the child nodes in the chosen subtree as the candidate separator tags.

In the first prototype of Omini, five separator tag identification heuristics are supported, covering a wide range of possible mechanisms for discovering object separators. Each of the five heuristics independently ranks the candidate tags. The standard deviation heuristic (SD) and the repeating pattern heuristic (RP) were first proposed in [6]. The SD heuristic ranks the candidate tags based on the standard deviation of sizes between two tags. The RP heuristic ranks the candidate tags based on the difference between the counts of a pair of tags and the counts of a single tag. The partial path heuristic (PP) and the sibling tag heuristic (SB) are introduced in Omini. The former is motivated by the observation that the multiple instances

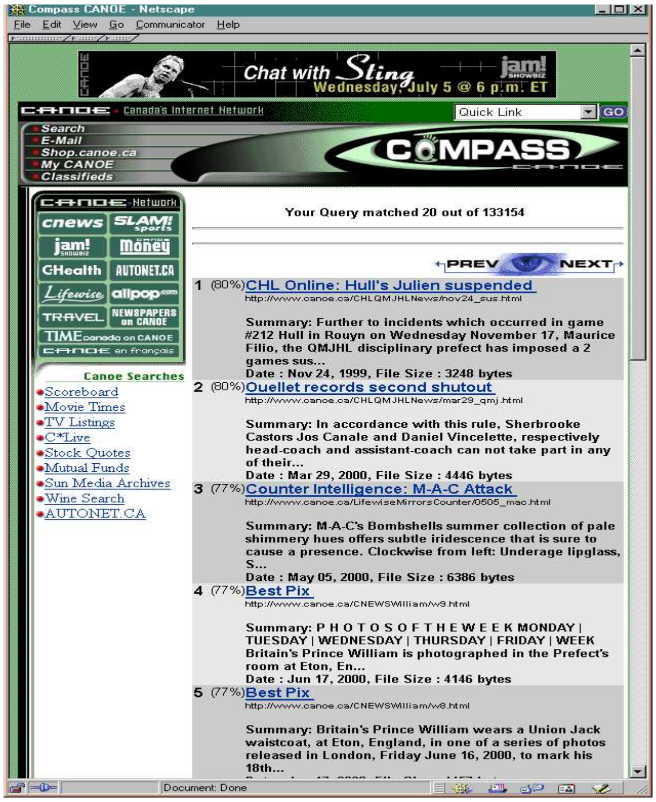

Figure 4: Canoe.com Search Result http://www.canoe.com – on July 2, 2000

of the same object type often have the same tag structure. The latter is based on the observation that the objects identified by highest count sibling pairs are more likely to be of the same object type than the highest count single tags. The identifiable path separator heuristic (IPS) is an extension of the IT (Identifiable Tag) heuristic proposed in [6]. Instead of using the same list of pre-determined and ranked candidate tags for every tag tree, a different list is used based on the subtree that is chosen.

In the subsequent sections we describe each of the five individual heuristics first and then we discuss the method to best combine the rankings of the five heuristics for selecting the correct object separator.

4.1. Standard Deviation Heuristic (SD)

The SD heuristic measures the standard deviation in the distance (in terms of the number of characters) between two consecutive occurrences of a candidate tag, and then ranks the list of candidate tags in ascending order by their standard deviation. It is motivated by the observation that the multiple instances of the same object type in a web document are typically about the same size.

Consider the tag tree for the Library of Congress web page shown in Figure 1. From the subtree extraction step, the subtree anchored at the node HTML[1].Body[2] is the chosen minimal subtree as shown in Figure 2. Among the set of child node tags, some tags have much higher counts than the others do. For example, the tag *hr* appears twenty-one times, the tag *a* appears twenty-one times, and the tag *pre* occurs twenty times. We refer to these tags as the highest count tags. The standard deviation in distance is calculated between two consecutive occurrences of hr tag, between two consecutive occurrences of pre tag, between two consecutive occurrences of a tag, and so on. Table 1 shows the top three candidate tags by applying the SD algorithm to the library of congress example in Figure 2. It ranks the candidate tags in ascending order by the standard deviation in distance, with the smallest standard deviation first.

Rank	Tag	Standard Deviation
1	hr	114
2	pre	117
3	a	122

Table 1: Standard Deviation for tags from the minimal subtree in Figure 2

4.2. Repeating Pattern Heuristic (RP)

The RP heuristic chooses the object separators by counting the number of occurrences of all pairs of candidate tags that have no text in between. It computes the absolute value of the difference between the count for a pair of tags and the count for each of the two paired tags alone and then ranks the candidate tags in ascending order by this absolute value. The intuition behind this heuristic is that a single tag may be used to mean many things, but a pattern of two or more tags is more likely to mean just one thing. When there is no such pairs of tags in the chosen subtree, the RP heuristic produces an empty list of candidate tags. It simply means that the RP heuristic has no answer about which of the candidate tags is the object separator tag.

Tag Pair	Pair Count	Difference
table, tr	13	0
img, br	2	0
map, table	1	0
form, table	1	0
br, img	1	1
br, table	1	1

Table 2: Repeating tag ranking for minimal subtree from Figure 5

Consider the subtree $HTML[1].Body[2].form[4]$ in Figure 5. Table 2 shows a ranked list of all tag pairs in de-

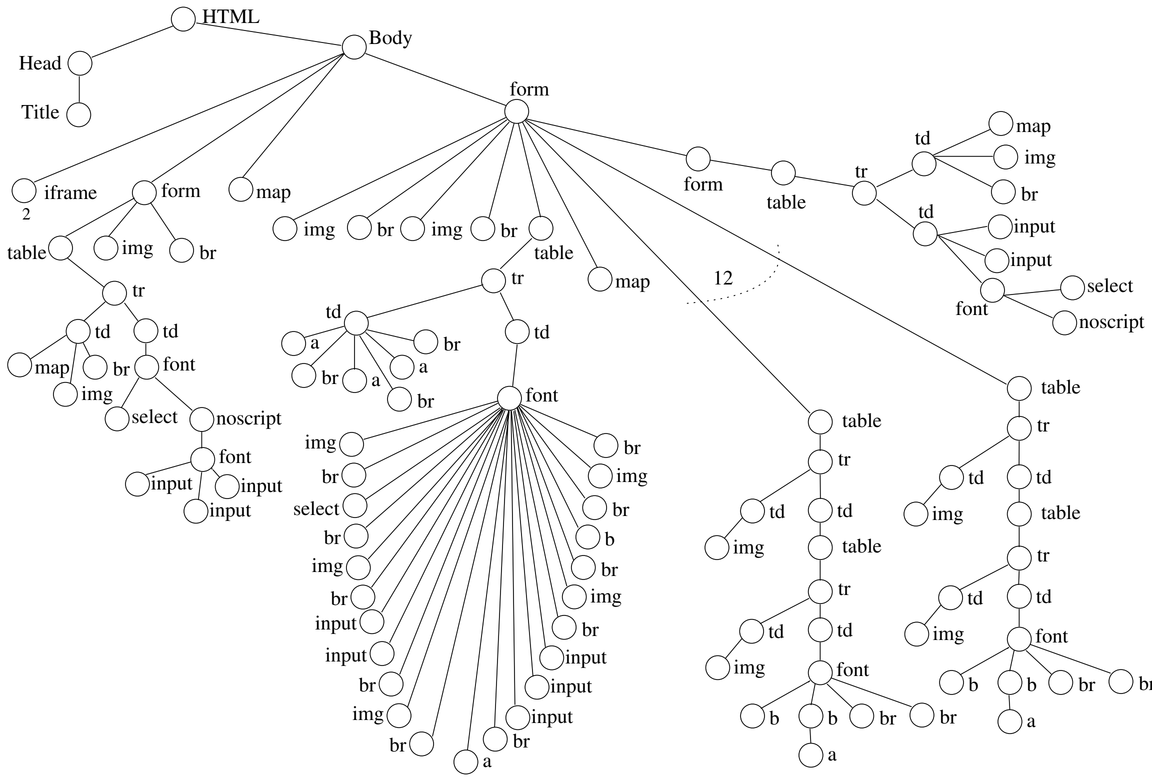

Figure 5: Tag Tree for Canoe.com, Figure 4

scending order by the pair counts and the difference between the pair count and the tag count.

4.3. Identifiable Path Separator Tag heuristic (IPS)

The IPS heuristic ranks the candidate tags of the chosen subtree according to the list of system-supplied IPS tags. The IPS tags are those tags that are identified by the system as the most commonly used object separator tags for different types of subtrees in Web documents. The idea behind this heuristic is the following. First we observe that Web documents, generated either by hand or by authoring tools or by server programs, often consist of multiple presentation layouts within a single page, each is defined by some specific type of HTML tags. For example, a web page may contain a table marked by table tag `table`, a list marked by the list tag `ul` or `ol`, and a paragraph marked by the tag `p`. Second, each such a presentation layout tends to use a regular structure. For example, a table tends to use the row tag `tr` and the column tag `td` to define rows and columns of the table; and a list tends to use the list item tag `li` to define the list structure. Therefore, for each presentation layout (i.e., a subtree type), there are a few tags that are used consistently for separating objects within the subtree.

Based on these observations and the set of Web documents we have tested, we create a list of object separator tags for each type of subtrees as shown in Table 3. The full list of object separators is composed of all the identified tags for each type of subtree listed in Table 3, with duplicates removed.

Subtree	Tag List
body	table,p,hr,ul,li,blockquote,div,pre,b,a
table	tr,b
form	table,p,dl
td	table,hr,dt,li,p,tr,font
dl	dt,dd
form	table,p,dl
ol	li
ul	li
blockquote	p

Table 3: A Table of Object Separator Tags

The next step is to determine the rankings of these commonly used object separator tags. Table 4 lists the distribution of all object separator tags we observed in our tests of 50 web sites with over 2000 web pages. Based on the ex-

Tag	% of time used as object separator
tr	34
table	18
p	10
li	8
hr	6
dt	6
ul	2
pre	2
font	2
dl	2
div	2
dd	2
blockquote	2
b	2
a	2

Table 4: Object separator probability

perimental results over the testing web sites, we produce the following ranking of the full list of IPS tags. We refer to such an ordered list as $IPSList$:
$\{tr, table, p, li, hr, dt, ul, pre, font, dl, div, dd, blockquote, b, a, span, td, br, h4, h3, h2, h1, strong, em, i\}$.
For tags of the same rank, the order is arbitrary.

4.4. Sibling Tag Heuristic (SB)

The SB heuristic counts pairs of tags that are immediate siblings in the minimal subtree, and ranks all pairs of tags in descending order by the number of occurrences of the pair. For those pairs of tags that have equal occurrence, the ranking follows the order of their appearances in the Web document. For example, in the fragment `<p><a>` ... `</a><b>` ... `</b><c>` ... `</c></p>`, the sibling pairs are (a,b) and (b,c) and each occurs only once. Table 5 ranks sibling pairs from the Library of Congress tag tree in Figure 1 and Canoe.com tag tree in Figure 5. The sibling tag heuristic is motivated by the observation that given a minimal subtree, the object separator tag is expected to appear as many times as there are objects.

4.5. Partial Path Heuristic (PP)

The PP heuristic lists all paths from a candidate node to any other node, which is reachable from this candidate node, in the chosen subtree, and counts the number of occurrences of each identified path. The list of candidate tags is ranked in descending order by the count of all the identified paths. If two paths have an equal count, then the longer path will rank higher than the shorter one because it indicates more structure. Interesting to note is that if there are no paths with a length more than one, such as in Figure 1, this heuristic reduces to simply choosing the tag with the highest count. The main idea behind this heuristic

	Canoe.com		Library of Congress	
Rank	pair	count	pair	count
1	table,table	11	hr,pre	20
2	img,br	2	pre,a	20
3	br,img	1	a,hr	20
4	br,table	1	h1,i	1
5	table,map	1	i,hr	1
6	map,table	1	hr,a	1
7	table,form	1	a, br	1
8			br,form	1
9			form,p	1

Table 5: Tag ranking for SB heuristic on Figure 5 and 1

is motivated by the observation that the multiple instances of the same object type often have the same tag structure.

Table 6 shows the PP rankings for the example web document in Figure 5 and the Library of Congress page in Figure 1.

	Canoe.com		Library of Congress	
Rank	tag	count	tag	count
1	table	26	hr	21
2	form	2	a	21
3	img	2	pre	20
4	br	2	form	8

Table 6: Tag ranking from partial path rankings for Figure 5 and Figure 1

5. Determining the Correct Object Separator Tag: The Combined Algorithm

We have discussed each individual heuristic and its algorithm to produce a ranked list of candidate tags. Each of the five individual algorithms works independently towards the same goal – finding the right object separator from the set of candidate tags. As a result, each heuristic chooses the highest ranked tag as the object "correct" separator. However, as we observed from the discussions in the previous sections, these five heuristics may not always agree on their highest ranked choice.

To understand the performance of these individual heuristics on different types of web pages, we conducted a series of experiments over 500 web pages from 15 different web sites (see [4] for further detail). An empirical probability distribution for the success rate of each individual heuristic is listed in Table 7. For each heuristic, we first calculated the number of times the heuristic chose a correct object separator tag at a particular rank. Then for each web site, we calculated the success rate of the given heuristic by normalizing the number of times the correct separator tag

Heuristic	Rank 1	2	3	4	5
SD	0.78	0.18	0.10	0.00	0.00
RP	0.73	0.13	0.00	0.00	0.00
IPS	0.40	0.46	0.13	0.07	0.00
PP	0.85	0.06	0.02	0.00	0.00
SB	0.63	0.17	0.12	0.06	0.03

Table 7: Probability rankings for object separator heuristics

is chosen by the number of pages tested over a particular web site. The success rate for each heuristic over the 15 web sites (shown in Table 7) was calculated by averaging the normalized numbers from each web site.

An obvious way to improve the accuracy of finding a correct object separator in a web document is to consider the best way of combining these independent heuristics. A well-known approach for combining evidences from two or more independent observations is to use the basic laws of probability [9]: Let $P(A)$ be the probability associated with the result of applying heuristic A over a web document, and $P(B)$ be the probability associated with the result of applying heuristic B over the same web document. The formula $P(A \cup B) = P(A) + P(B) - P(A \cap B)$ will produce the compound probability $P(A \cup B)$ for locating a correct object separator tag in this web document. For example, if the probability factors that a tag tr is an object separator in a web document are 78%, 63%, and 85%, then the compound probability for tag tr is 89% ($78\% + 63\% + 85\% - 78\% \times 63\% - 78\% \times 85\% - 63\% \times 85\% + 78\% \times 63\% \times 85\% = 99\%$).

To combine the five individual heuristics, there are 26 possible combinations in addition to the trivial case of none and the five individual heuristics.

Based on the set of testing web documents, the combination of all five heuristics performs the best.

Combo	Success	Combo	Success	Combo	Success
IB	0.61	RB	0.73	RI	0.75
RS	0.78	SI	0.78	SB	0.78
RIB	0.80	RSB	0.84	SIB	0.84
RPB	0.85	RP	0.85	SP	0.85
IPB	0.85	IP	0.85	PB	0.85
RSI	0.86	RIPB	0.86	RIP	0.86
RSP	0.88	SIPB	0.89	SIP	0.89
SPB	0.89	RSIP	0.92	RSIB	0.92
RSPB	0.92	RSIPB	0.98		

Table 8: Success rates for heuristic combinations on test data

6. Experimental Results

In this section we report our experiment setup and the experimental results for validation of our approach. We also report the experimental results conducted for a comparison of our approach with the BYU's approach [6].

To run our experiments, we downloaded and cached web pages from 50 different Web sites. To automatically retrieve the pages we first generated a random list of 100 words from the standard Unix dictionary. Then we fed each word into a search form at each of the 50 web sites. After retrieving the pages we discarded those pages which returned no results. For the static web pages which do not have a search interface, such as *agents.umbc.edu*, a manual approach is used to load the pages to the local storage. All experiments were carried out on the local version of the pages so as not to overload web sites and to be able to obtain consistent results over time.

The success rate of an algorithm is calculated in two steps. First, for each web site we calculate the percentage of the downloaded pages in which the highest ranked tag of the algorithm is the correct separator tag. These percentages are then averaged over the collection of web sites to determine the success rate for individual heuristics and their combinations.

We have discussed the combined algorithm for object extraction and our experimental approach to determining the best combination in Section . To further validate the effectiveness and quality of the Omini approach to object extraction, we ran the algorithms over 1,500 web pages from 25 web sites [4]. These 25 web sites cover a broad range of application domains and a good variety of web documents. For each of the 1,500 documents, we applied the five heuristics and the combination of the five **RSIPB**. Table 9 lists the experimental results. Even though four out of five individual heuristics did not reach a success rate higher than 90%, the combination of the five attained 94% success rate. The experimental results indicate that, while the heuristics are not extremely stable, it is beneficial to combine them.

Heuristic	Rank 1	2	3	4	5
SD	0.77	0.15	0.06	0.05	0.00
RP	0.77	0.10	0.07	0.02	0.00
IPS	0.88	0.08	0.07	0.00	0.00
PP	0.93	0.05	0.00	0.00	0.00
SB	0.71	0.13	0.06	0.07	0.04
RSIPB	0.94	0.05	0.02	0.01	0.00

Table 9: Probability rankings for object separator heuristics on experimental data

Success rate is not the only measure of quality for object separator identification algorithms. We also use recall and precision to judge how well our algorithms perform.

Recall is the percentage of positive instances of the target concept (in our case the object separator tag) that are correctly identified. Precision is the percentage of extractions made that are correct. Both of these numbers are defined in terms of false positives (FP), false negatives (FN), and true positives (TP). $Recall = \frac{TP}{TP+FN}$ and $Precision = \frac{TP}{TP+FP}$. A true positive is an instance where an object separator exists and it is correctly identified by the algorithms. A false negative is an instance where the object separator exists but is not found by the algorithms. A false positive is an instance where the object separator does not exist, but a tag is mistakenly identified as an object separator.

We evaluated the Omini system using 2,000 Web pages over 40 popular web sites. Table 10 shows the precision and recall measures obtained from a series of experiments on 500 web pages from 15 web sites (see [4] for further details), which we used for testing and training of our heuristics. Table 11 displays the precision and recall ratios for 1,500 web pages collected over 25 web sites [4]. The results were uniformly satisfying, attaining recall ratio in the range of 93% to 98% and precision ratios 100% on all the web documents we examined.

Heuristic	Success	Precision	Recall
SD	0.78	1.00	0.78
RP	0.73	0.84	0.73
IPS	0.71	0.82	0.71
PP	0.85	0.92	0.85
SB	0.62	0.89	0.62
RSIPB	0.98	1.00	0.98

Table 10: Probability rankings for object separator heuristics on test data

Heuristic	Success	Precision	Recall
SD	0.77	1.00	0.77
RP	0.77	0.97	0.77
IPS	0.88	0.94	0.88
PP	0.93	1.00	0.93
SB	0.71	0.97	0.71
RSIPB	0.94	1.00	0.94

Table 11: Probability rankings for object separator heuristics on experimental data

6.5. Experimental Comparison with the BYU's approach

The Omini object extraction approach was motivated by several research results reported in the literature [6, 7, 11]. Among these reported research results, the work done at BYU [6] inspires us the most. For example, we adopt two of the five data extraction heuristics from [6], namely the SD and RP heuristics, without any change. Our IPS heuristic is an evolution of their identifiable tag (IT) heuristic. IT chooses tags based on a predefined list of common object separators. We found this to be inflexible when a larger variety of web sites are considered. Therefore, instead of using the same list of candidate separators for all kinds of web sites, we use different lists depending on the type of the tag node at which the minimal subtree is anchored. This allows us, for example, to list the tag *tr* first for tables, *li* first for lists, *table* for *body* tags, and so on. Our IPS heuristic has much higher extensibility and scalability with respect to the fast evolution of the Web. We did not include the highest count (HC) heuristic, which ranks tags based the number of times they appear. Through several experiments we found that the success rate for the HC heuristic is not as desirable as one would expect. First, the HC this heuristic was not a part of any of the most successful heuristic combinations; Second, the combinations that include the HC heuristics were often less successful in choosing a correct object separator than the same combination without the HC heuristic. Finally, the PP heuristic is extension of the HC heuristic that behaves exactly the same as HC on certain classes of web sites (such as the Library of Congress page in Figure 1). We also rejected their proposed ontology match heuristic. This heuristic relies on knowing about the domain of a web site and having a detailed ontology for that domain. An example of such ontology is to use a specific word or phrase that is known to appear only once in every object of interest. As reported in [6], developing the ontology for a domain takes about 2 man-weeks of work. The goal of the Omini project is to develop a fully automated object extraction system. We believe that a fully automated approach is the best way to develop a scalable system as the Web grows. While the OM heuristic reportedly works well in [6], the required human intervention makes this heuristic unsuited to our approach.

For the sake of performance comparison, we have implemented all of the heuristics in [6] except for the ontology based heuristic. We conducted a set of experiments over a wide variety of web pages. There are some cases where the approach in [6] performs well. In those cases our heuristics perform at least as well, if not better. However, there are some cases where the four heuristics in [6] perform poorly; for example, on the sites listed in Table 12, their heuristics only achieved a success rate of only 59%. In these cases our heuristics still perform very well, with a 93% success rate.

Website	Date
www.bookpool.com	March 2000
www.ebay.com	March 2000
www.goto.com	March 2000
www.powells.com	March 2000
www.signpost.org	March 2000

Table 12: Example Web Sites

BYU		Omini	
Heuristic	**Success**	**Heuristic**	**Success**
RP	19	RP	19
SD	23	SD	23
IT	40	IPS	76
HC	40	SB	56
		PP	78
HTRS	59	RSIPB	93

Table 13: Success rates for different heuristics and combinations on the web sites in Table 12

7. Conclusion

We have presented the object separator extraction algorithms. These algorithms have been tested and evaluated using more than 2,000 documents from 40 Web sites (primarily electronic commerce sites). The experimental evaluation consists of three parts. First, the result of Omini analysis is visually inspected for the identification of false negatives (correct objects that were left out) and false positives (incorrect objects included in the result). Omini results show an accumulated precision of 100% (only correct objects are returned) and an accumulated recall between 93% and 98% (very few actual objects left out). Second, we replicated BYU's system [6] without the ontology heuristic (the human-dependent component). Omini results compare favorably to the BYU information extraction system. Finally, the Omini execution overhead is measured and the typical web page processing takes about 0.2 seconds. Furthermore, if the subtree and object separator discovery rules are cached, then the overhead is dominated by the fetching and parsing of web pages (about 0.1 seconds).

Fully automated object extraction is an important and necessary component in the construction of scalable and reliable next-generation information search and aggregation services on the Web. We plan to demonstrate the usefulness of Omini by combining it with a wrapper generation system (e.g., the XWRAP Elite [14]) to automate the wrapper generation and evolution process. Other potential future research directions include the automation of evaluation process and incorporation of feedback-based refinement of object extraction, as well as the integration with query optimization and semantic interoperability software systems.

Acknowledgement. The authors would like to thank the referees for their valuable comments. This work was partially supported by DARPA/ITO under the Information Technology Expeditions, Ubiquitous Computing, Quorum, and PCES programs.

References

[1] B. Adelberg. Nodose - a tool for semi-automatically extracting structured and semi-structured data from text documents. *ACM SIGMOD*, 1998.

[2] N. Ashish and C. A. Knoblock. Semi-automatic wrapper generation for internet information sources. In *Proceedings of Coopis Conference*, 1997.

[3] P. Atzeni and G. Mecca. Cut and paste. *Proceedings of 16th ACM SIGMOD Symposion on Principles of Database Systems*, 1997.

[4] D. Buttler, L. Liu, and C. Pu. Omini: An Object Mining and Extraction System for the Web. *Technical Report, Sept. 2000. Georgia Tech, College of Computing.*

[5] R. Doorenbos, O. Etzioni, and D. Weld. A scalable comparison-shopping agent for the world wide web. *Proceedings of Autonomous Agents*, pages 39–48, 1997.

[6] D. W. Embley, Y. Jiang, and Y.-K. Ng. Record-boundary discovery in web-documents. In *Proceedings of the 1999 ACM SIGMOD*, Philadelphia, Pennsylvania, USA, June 1999.

[7] O. Etzioni and D. Weld. A softbot-based interface to the internet. *Communications of the ACM*, 37(7):72–76, 1994.

[8] J. Hammer, H. Garcia-Molina, J. Cho, R. Aranha, and A. Crespo. Extracting semi-structured data from the web. *Proceedings of Workshop on Management of Semi-structured Data*, pages 18–25, 1997.

[9] J. J. Higgins and S. Keller-McNulty. *Concepts in Probability and Stochastic Modeling.* Duxbury, 1995.

[10] N. Kushmerick, D. Weil, and R. Doorenbos. Wrapper induction for information extraction. In *Proceedings of Int. Joint Conference on Artifical Intelligence (IJCAI)*, 1997.

[11] L. Liu, C. Pu, and W. Han. XWrap: An XML-enabled Wrapper Construction System for Web Information Sources. *Proceedings of the International Conference on Data Engineering*, 2000.

[12] A. Sahuguet and F. Azavant. WysiWyg Web Wrapper Factory (W4F). *Proceedings of WWW Conference*, 1999.

[13] S. Soderland. Learning to extract text-based information from the world wide web. *Proceedings of Knowledge Discovery and Data Mining*, 1997.

[14] G. Tech. XWRAP Elite Project. *http://www.cc.gatech.edu/projects/disl/XWRAPElite*, May 2000.