Joint Optimization of Wrapper Generation and Template Detection

Shuyi Zheng [*]
Pennsylvania State University
University Park, PA 16802
shzheng@cse.psu.edu

Di Wu
The Chinese University of Hong Kong
Hong Kong, China
dwu@se.cuhk.edu.hk

Ruihua Song & Ji-Rong Wen
Microsoft Research Asia
Beijing 100080, China
{rsong,jrwen}@microsoft.com

ABSTRACT

Many websites have large collections of pages generated dynamically from an underlying structured source like a database. The data of a category are typically encoded into similar pages by a common script or template. In recent years, some value-added services, such as comparison shopping and vertical search in a specific domain, have motivated the research of extraction technologies with high accuracy. Almost all previous works assume that input pages of a wrapper induction system conform to a common template and they can be easily identified in terms of a common schema of URL. However, we observed that it is hard to distinguish different templates using dynamic URLs today. Moreover, since extraction accuracy heavily depends on how consistent input pages are, we argue that it is risky to determine whether pages share a common template solely based on URLs. Instead, we propose a new approach that utilizes similarity between pages to detect templates. Our approach separates pages with notable inner differences and then generates wrappers, respectively. Experimental results show that our proposed approach is feasible and effective for improving extraction accuracy.

Categories and Subject Descriptors

H.3.m [**Information Storage and Retrieval**]: Miscellaneous—*Data Extraction, Wrapper Generation, Web*

General Terms

Algorithms, Experimentation, Performance

Keywords

Wrapper, Template Detection, Information Extraction

[*]Work was done when the authors were visiting Microsoft Research Asia.

1. INTRODUCTION

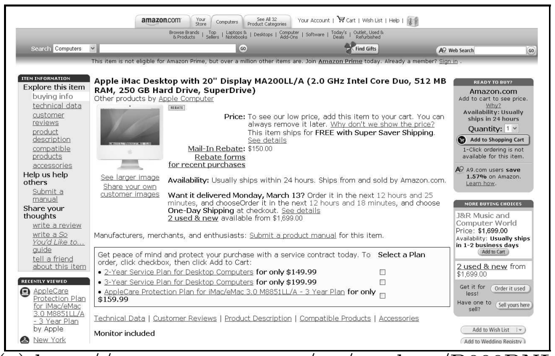
Websites like Amazon.com are data-intensive, and information on them comes from structured sources. Often the data are encoded into semi-structured HTML pages that employ templates for rendering. Some value-added services, such as comparison shopping, are emerging to query or manipulate the data, such as products and reviews, from several websites. To achieve high accuracy, the task of extracting structured information from Web pages is usually implemented by programs called wrappers. Manually writing wrappers for Web sources [9] is a tedious, time-consuming, and error-prone job, thus the study of automatic wrapper induction using machine learning techniques has been a subject of research in recent years [12, 11, 17, 16, 3, 6, 5, 18, 4, 21, 10]. This paper also focuses on wrapping Web sources in an automatic manner.

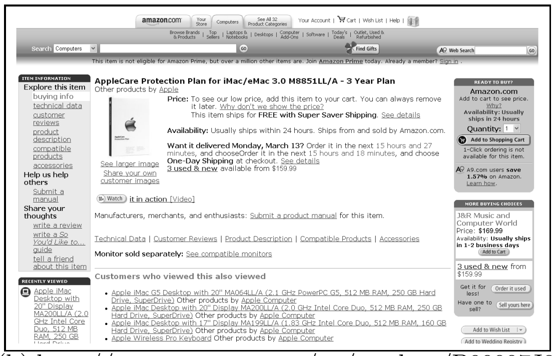
Although different wrapper induction systems employ various techniques and strategies to generate wrappers, they all separate template detection from learning wrappers. The detection groups training pages into several clusters or classes, based on cues like URLs. For example, [7] assumes that pages belonging to the same template are located at the same sub-directory of a website. Thus, pages are considered to share a common template if their URLs fit a common schema. Grouped pages are then fed into an induction module. The module generally assumes that a group of pages conforms to a common template, and generates a wrapper per class.

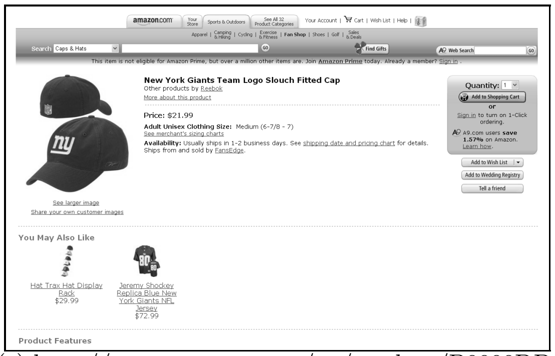
The separated template detection strategy has at least two limitations:

1. With the popularity of dynamic URLs, it is no longer as effective to detect templates by URLs as before, especially for some large-scale websites. Figure 1 lists four sample pages collected from Amazon.com. From their appearances, it is easy to tell that Figure 1(a) and Figure 1(b) share a common template, and Figure 1(c) and Figure 1(d) share another template. Comparing their URLs, we find there are no cues to allow correct grouping of the pages.

2. Even if URLs can group pages that share a template, such a method is sometimes far from optimal to generate only one wrapper for a complex template. For example, by looking closely at page (c) and (d) in Figure 1, we observe that page (d) is different from page (c) in some aspects. Page (d) has an additional attribute named Color, and it lists some also-viewed


Apple iMac Desktop with 20" Display MA200LL/A (2.0 GHz Intel Core Duo, 512 MB RAM, 250 GB Hard Drive, SuperDrive)
(a) http://www.amazon.com/gp/product/B000BNLGJA/


AppleCare Protection Plan for iMac/eMac 3.0 M8851LL/A - 3 Year Plan
(b) http://www.amazon.com/gp/product/B00007J8SC/



(c) http://www.amazon.com/gp/product/B0000DD95R/

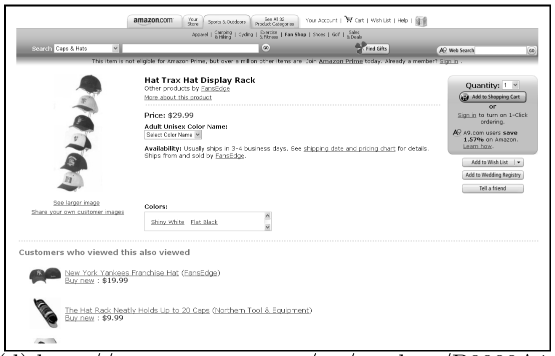

Hat Trax Hat Display Rack
Customers who viewed this also viewed
(d) http://www.amazon.com/gp/product/B0000A1AT9/

Figure 1: Sample pages from www.amazon.com

items in a column whereas page (c) lists some you-may-also-like items in a row. Building two wrappers for such a complex template may achieve better extraction accuracy.

To solve the problems above, we propose detecting template solely based on the similarity among page representations that are also used in wrapper generation. In our system, tree structures are used as representations for pages and wrappers. Based on a distance metric between a page and a wrapper, a clustering algorithm is employed to cluster similar enough pages into a class and induces a central wrapper for the class at the same time. Such a joint approach makes it possible to optimize the final performance of extraction by involving template detection in the training process.

Compared with prior works, our approach has two advantages:

1. Our approach is more stable because it does not rely on URLs or any other external features to detect templates. Instead, we attempt to detect templates based on inner structure similarity of pages, which is consistent with the principle of wrapper induction.

2. Given a set of pages, the number of wrappers is determined by how similar they are. This number can be optimized under the criterion of overall extraction accuracy of generated wrapper set.

The rest of this paper is organized as follows. We explain our main idea by clarifying two concepts on template in Section 2. Section 3 provides the basic representations. Section 4 formally states the addressed problem and briefly overviews our solution. In Section 5, we describe wrapper induction algorithm that is implemented in our system, and propose a new wrapper-oriented page clustering algorithm that joins template detection with wrapper generation together. Section 6 reports some experimental results, while Section 7 describes related works. Section 8 concludes the paper with directions for future work.

2. GROUND-TRUTH TEMPLATES AND SIMILARITY-BASED TEMPLATES

Before going to the details of our approach, in this section we will first describe the main idea of this paper.

What's a ground-truth template? In previous related work, the concept of template has been presented by various descriptive definitions. Most of them associate a template with a script that encodes the data of a category into a group of HTML pages, called a page class. For example, we can guess that both page (a) and page (b) in Figure 1 are generated dynamically by a script for the category of computer while page (c) and page (d) generated by another script for the category of cap. This kind of templates does exist and it is indispensable for wrapper induction systems to generate effective wrappers in reverse. We call these kind of templates *Ground-Truth Templates* because they denote the original relations among pages of a website. The corresponding page classes are called *Ground-truth Page Classes.*

However, the ultimate purpose of template detection is not to guess which pages are encoded by a ground-truth template, but to generate more effective wrappers that can correctly extract data. What we propose is to cluster pages into several groups based on how similar they are. In general, a page is less different from the pages in the same group than those pages in other groups. Each group is corresponded to a template that is called *Similarity-based Template* and the group itself is a *Similarity-based Page Class*.

For a particular page set P, since the ground-truth templates are invisible to us, it is difficult and not necessary to ensure that detected similarity-based templates are exactly same to ground-truth templates. For example, for pages like page (c) and page (d) in Figure 1, a ground-truth page class may be divided into two similarity-based page classes because the color attribute is optional so that some pages like page (d) have such attributes while others like page (c) do not. It is very likely that extracted results by using two similarity-based templates are good and even better than those by inducing one ground-truth template because the complexity of these templates becomes lower than that of one ground-truth template.

In addition, we found that the definition of similarity is highly related to alignment in the stage of wrapper induction. We propose to use consistent representations in both template detection and wrapper generation and jointly optimize these two stages to achieve better extraction performance. In our system, tree-structures are used as representations for pages and wrappers, although the representation is not restricted to tree-structure.

3. DATA REPRESENTATIONS

We describe the representations of a Web page and a wrapper in this section.

3.1 DOM (Document Object Model) Tree

DOM tree is the representation of a HTML page in our system. Each DOM node of a DOM tree represents an HTML tag pair (e.g., `<TABLE>` and `</TABLE>`). The nested structure of HTML tags corresponds to the parent-child relationship among DOM nodes. Thus, a DOM tree is formed naturally. More information about DOM specification can be found at [1].

In our experiments, DOM trees used for wrapper generation are manually labeled so that the generated wrappers can extract values and assign labels in one step. Labels are only assigned to leaf nodes of a DOM tree. DOM nodes with different labels are considered different no matter whether they have the same tag or not. In the rest of this paper, for a given DOM node σ, we use $\mathcal{T}(\sigma)$ and $\mathcal{L}(\sigma)$ to denote its tag and label.

3.2 Wrapper

In our system, a wrapper is also presented in tree structure that can be regarded as extended DOM trees with *Sign* for each node.

Definition 1. *(Node Sign) Given a wrapper node σ, its sign $\mathcal{S}(\sigma)$ indicates its matching rule in the alignment between its owner wrapper and a DOM tree. $\mathcal{S}(\sigma)$ can be 1 or an integer $N (N \geq 2)$ or one of the following wildcards: $?, +, *$.*

Rule 1. *Given a wrapper node σ,*

- *$\mathcal{S}(\sigma) = 1$ means σ can only match one DOM node.*
- *$\mathcal{S}(\sigma) = N (N \geq 2)$ means σ can match consecutive N DOM nodes.*
- *$\mathcal{S}(\sigma) = ?$ means σ can match one DOM node or no DOM node at all.*
- *$\mathcal{S}(\sigma) = +$ means σ can match consecutive N DOM nodes $(N \geq 1)$.*
- *$\mathcal{S}(\sigma) = *$ means σ can match consecutive N DOM nodes $(N \geq 1)$ or no DOM node at all.*

Such wrapper node signs are similar to the wildcards used in other works like [6, 18].

Another difference between a wrapper and a DOM trees is that a wrapper may have a kind of special nodes that act like pairs of parentheses, called *Parentheses Nodes*. These nodes have no corresponding tags and must be inner nodes with at least one child. For the sake of convenience, we call other DOM nodes or wrapper nodes as *Tag Nodes*.

4. PROBLEM DEFINITION AND SYSTEM OVERVIEW

In this section we formally define the extraction problem and briefly overview our solution.

4.1 Problem Definition

We define the problem as follows:

Given a set of labeled DOM trees D parsed from pages of a particular website, a group of wrappers $(w_1, w_1, ..., w_n)$ should be learned from D. And the target is to maximize the overall extraction accuracy $\mathcal{P}$ when generated wrappers are tested on another DOM-tree set $D^{'}$ that comes from the same website.

In this paper, we use manually labeled training data to explain and verify our ideas. Although the idea of joint optimization of wrapper induction and template detection is not constrained to labeled data, we do so for several reasons. First, our main focus is not on the algorithm of wrapper induction but on how to detect similarity-based templates and how the detection influences extraction performance. Labeled data can simplify the evaluation of extraction results. Second, using labeled data to generate wrappers is commonly used in some scenarios such as comparison shopping. As the accuracy of price is required to be close to 100 percent, automatic attributes labeling methods cannot meet the requirement. Furthermore, inducing wrappers based on labeling data is selectively used for only a few of head sites. For each site, as few as tens of pages are enough to train a robust wrapper set. Thus, the cost of labeling is acceptable.

4.2 System Overview

A flowchart of our system is shown in Figure 2.

To begin with, training pages are parsed into DOM trees before they are processed by our system. We will not discuss the HTML parsing technique since it is beyond the scope of this paper.

Second, the DOM trees will be fed to the wrapper-oriented page clustering module that combines template detection and wrapper generation into one step and outputs a set of wrappers. A by-product in the step is that the training DOM trees are also clustered into similarity-based page classes.

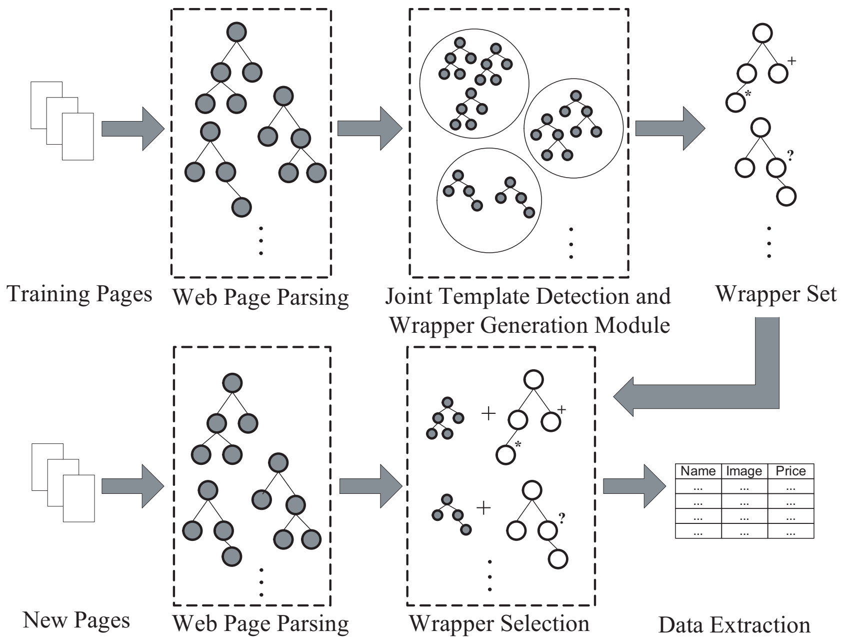


Figure 2: System overview

When a new Web page is introduced, it will be parsed into a DOM tree first. Then, our system can automatically select a wrapper from the generated wrapper set, which makes a best match with the DOM tree. At last, data is extracted and saved in a structured format like a relational database.

5. JOINT OPTIMIZATION OF WRAPPER GENERATION AND TEMPLATE DETECTION

In this section, we present the idea of joint optimization of wrapper generation and template detection in detail. We first introduce the wrapper generation algorithm that is implemented in our system. We then describe how template detection is combined with wrapper generation by a proposed algorithm called wrapper-oriented page clustering.

5.1 Wrapper Generation

In Section 5.1.1, we describe how to convert a DOM tree to a wrapper tree. This is the first step for a page before it is evolved in wrapper generation in our system. In Section 5.1.2 and Section 5.1.3, we implement a cost-driven algorithm to perform wrapper induction. This algorithm synthesizes several state-of-the-art techniques [6, 4, 18], e.g., regular expression inference.

5.1.1 *Convert a DOM tree to a Wrapper*

Given a source DOM tree T_d, supposing that the converted wrapper is T_w, we use $T_d \rightarrow T_w$ to indicate this conversion.

In $T_d \rightarrow T_w$, we need to perform a repeat pattern combination algorithm to make T_w more compact than T_d. This combination algorithm is similar to the work in [15]. If $N(N \geq 2)$ identical consecutive sub-trees are detected in T_d, they will be merged as one sub-tree rooted at a tag node σ in T_w, where $\mathcal{S}(\sigma) = N$. If $N(N \geq 2)$ identical consecutive sub-forests are detected in T_d, they will be merged as one sub-forest rooted under a parentheses node p in T_w, where $\mathcal{S}(p) = N$. Node labels are considered in the algorithm. Figure 3 illustrates this procedure, where letters indicate nodes' tags and subscripts indicate nodes' labels.

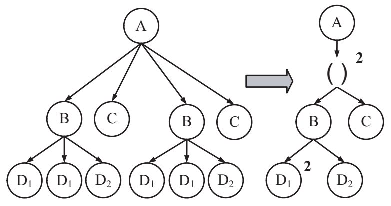


Figure 3: Repeat pattern combination

5.1.2 *Cost-Driven Tree Alignment*

Tree alignment is a frequently used algorithm in our system. There are two types of tree-alignment algorithms: one is for aligning two different wrappers, called WW-alignment, and the other is for aligning a wrapper and a DOM tree, called WD-alignment. We employ cost-driven dynamic programming for both algorithms.

In the tree-alignment algorithm, DOM nodes and wrapper nodes are the basic units for matching. Mismatched nodes will cause cost in the alignment. To calculate the cost, we need to assign weight to each node before aligning.

Definition 2. *(DOM-Node Weight) Given a DOM node σ, its weight $\mathcal{W}(\sigma)$ equals the number of nodes in the sub-tree rooted at σ, including itself.*

Definition 3. *(Wrapper-Node Weight) Given a wrapper node σ, its weight $\mathcal{W}(\sigma)$ can be calculated as follows:*

- *If σ is a leaf tag node and $\mathcal{S}(\sigma) = 1$, then $\mathcal{W}(\sigma) = 1$.*
- *If σ is a inner tag node, and $\mathcal{S}(\sigma) = 1$, then $\mathcal{W}(\sigma) =$ 1+ sum of its child nodes' weight.*

- *If σ is a parentheses node and $\mathcal{S}(\sigma) = 1$, then $\mathcal{W}(\sigma) =$ sum of its child nodes' weight.*

- *If $\mathcal{S}(\sigma) = ?$ or $\mathcal{S}(\sigma) = *$, then $\mathcal{W}(\sigma) = 0$.*

- *If $\mathcal{S}(\sigma) = +$, then $\mathcal{W}(\sigma) = \mathcal{W}(\sigma')$, where σ' is the same to σ except for $\mathcal{S}(\sigma') = 1$.*

- *If $\mathcal{S}(\sigma) = N$, then $\mathcal{W}(\sigma) = N * \mathcal{W}(\sigma')$, where σ' is the same to σ except for $\mathcal{S}(\sigma') = 1$.*

The reason we set a wrapper node σ's weight as 0 if $\mathcal{S}(\sigma) = ?$ or $\mathcal{S}(\sigma) = *$ is that this kind of wrapper node is allowed to be mismatched without causing any cost.

In this sub-section, we only describe WW-alignment and leave WD-alignment to Section 5.2.2.

Given two wrappers T_{w_1} and T_{w_2}, the basic procedure of WW-alignment is to align two forests: $\mathcal{A}(F_{w_1}, F_{w_2})$. It is performed in a top-down order layer by layer. Only nodes at the same layer of T_{w_1} and T_{w_2} can be aligned with each other.

RULE 2. *Given two wrapper nodes σ_{w_1} and σ_{w_2}, we say σ_{w_1} matches σ_{w_2}, iff all the following rules are satisfied,*

1. *σ_{w_1} and σ_{w_2} are either both inner nodes or both leaf nodes*

2. *$\mathcal{T}(\sigma_{w_1}) = \mathcal{T}(\sigma_{w_2})$*

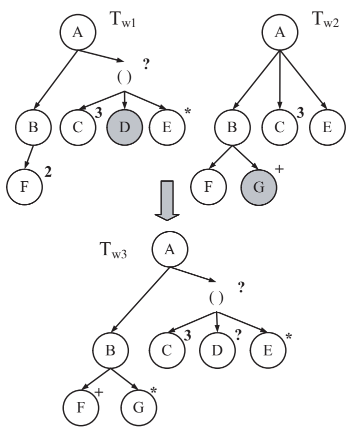
3. *If σ_{w_2} and σ_{w_2} are both leaf nodes, $\mathcal{L}(\sigma_{w_1}) = \mathcal{L}(\sigma_{w_2})$*

At each layer, $\mathcal{A}(F_{w_1}, F_{w_2})$ performs a sequence alignment between the array of F_{w_1}'s root nodes and that of F_{w_2}'s. Dynamic programming is adopted here to minimize the cost. All mismatched root nodes in F_{w_1} and F_{w_2} contribute their weight as cost to $\mathcal{A}(F_{w_1}, F_{w_2})$. For a pair of matched nodes σ_{w_1} and σ_{w_2} that are inner nodes, $\mathcal{A}(childF_{w_1}, childF_{w_2})$ will be invoked recursively, where $childF_{w_1}$ and $childF_{w_2}$ are the sub-forests consisting of sub-trees rooted at the child nodes of σ_{w_1} and σ_{w_2}. The cost caused by $\mathcal{A}(childF_{w_1}, childF_{w_2})$ will be counted in the cost calculation of $\mathcal{A}(F_{w_1}, F_{w_2})$. Because our WW-alignment algorithm works in such a top-down recursive way, it attempts to align nodes in two wrappers only if their parent nodes are aligned with each other. Such a mechanism saves some unnecessary alignment.

Figure 4 shows an example of WW-alignment. In this example, label difference is ignored for statement convenience. The alignment algorithm works as follows. First, $\mathcal{A}(\mathtt{A}, \mathtt{A})$ recursively invokes $\mathcal{A}\big(\mathtt{B}(\mathtt{C}^3\mathtt{DE}^*)^?, \mathtt{BC}^3\mathtt{E}\big)$. Then, according to the matching rule of wildcards ? (Rule 1), $\mathcal{A}\big(\mathtt{B}(\mathtt{C}^3\mathtt{DE}^*)^?, \mathtt{BC}^3\mathtt{E}\big)$ seeks a better solution between $\mathcal{A}(\mathtt{BC}^3\mathtt{DE}^*, \mathtt{BC}^3\mathtt{E})$ and $\mathcal{A}(\mathtt{B}, \mathtt{BC}^3\mathtt{E})$. Obviously, the former one costs less by now. Then, $\mathcal{A}(\mathtt{F}^2, \mathtt{FG}^+)$ is invoked recursively by both $\mathcal{A}(\mathtt{BC}^3\mathtt{DE}^*, \mathtt{BC}^3\mathtt{E})$ and $\mathcal{A}(\mathtt{B}, \mathtt{BC}^3\mathtt{E})$ to calculate the cost of these two solutions. In this example, WW-alignment algorithm can find an optimal result as Figure 4 shown with the cost of 2.

5.1.3 Wrapper Induction

After WW-alignment obtains an optimal result between two wrappers, a new wrapper can be constructed according



(Gray nodes are mismatched nodes.)

Figure 4: Wrapper induction

to the sign-inference function $\mathcal{I}$:

$\mathcal{I}(1, \mathtt{NULL})$	$=$	$?$	$\mathcal{I}(?, N)$	$=$	$*$
$\mathcal{I}(?, \mathtt{NULL})$	$=$	$?$	$\mathcal{I}(?, +)$	$=$	$*$
$\mathcal{I}(n, \mathtt{NULL})$	$=$	$*$	$\mathcal{I}(1, *)$	$=$	$*$
$\mathcal{I}(+, \mathtt{NULL})$	$=$	$*$	$\mathcal{I}(N, *)$	$=$	$*$
$\mathcal{I}(*, \mathtt{NULL})$	$=$	$*$	$\mathcal{I}(?, *)$	$=$	$*$
$\mathcal{I}(1, 1)$	$=$	1	$\mathcal{I}(+, *)$	$=$	$*$
$\mathcal{I}(N, N)$	$=$	N	$\mathcal{I}(1, N)$	$=$	$+$
$\mathcal{I}(+, +)$	$=$	$+$	$\mathcal{I}(N, +)$	$=$	$+$
$\mathcal{I}(?, ?)$	$=$	$?$	$\mathcal{I}(1, +)$	$=$	$+$
$\mathcal{I}(*, *)$	$=$	$*$	$\mathcal{I}(N_1, N_2)$	$=$	$+$
$\mathcal{I}(1, ?)$	$=$	$?$			

where NULL represents a mismatch of a wrapper node. For example, $\mathcal{I}(1, \mathtt{NULL})$ is applied for D node because D has the sign 1 in T_{w_1} while it is mismatched in T_{w_2}.

Given two source wrappers T_{w_1} and T_{w_2}, supposing that the generated wrapper is T_{w_3}, we use $T_{w_1} + T_{w_2} \rightarrow T_{w_3}$ to denote this induction procedure. Figure 4 also illustrates how to construct a new wrapper based on the alignment result.

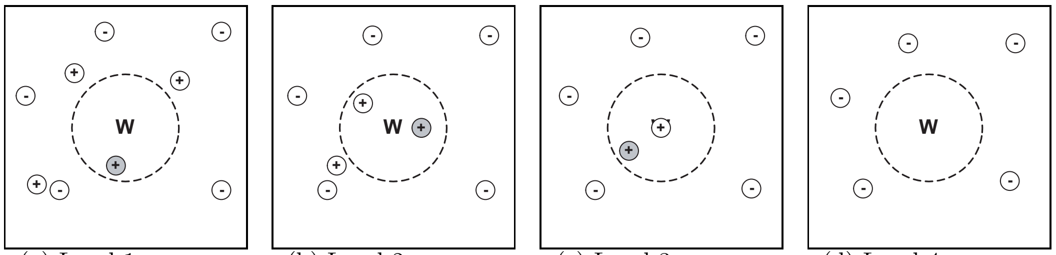
5.2 Combine Wrapper Generation with Template Detection

In Section 5.2.1, we describe the clustering algorithm that combines template detection and wrapper generation to achieve joint optimization. For clustering, a distance metric is defined in Section 5.2.2.

5.2.1 Wrapper-Oriented Page Clustering Algorithm

Wrapper-oriented page clustering (WPC) is the most novel part of our system. Given a set of DOM trees D, our WPC algorithm clusters DOM trees in D and generates a wrapper for each cluster. Actually, templates are detected one by one in WPC. After a template's clustering process is completed, all clustered DOM trees of this template will be removed and then clustering for another template will start. The cycles will stop when no DOM tree is left.

Here, we use Figure 5 to illustrate the clustering process of one template. In Figure 5, "W" represents a wrapper for this template, and positive points represent DOM trees that belong to the same template as the centered wrapper. Each



(a) Level-1 wrapper (b) Level-2 wrapper (c) Level-3 wrapper (d) Level-4 wrapper

Figure 5: Wrapper-oriented page clustering for one template

gray point represents a chosen DOM tree that will be used to refine the centered wrapper.

In the WPC algorithm, we use *Wrapper Level* to indicate a wrapper's complexity and generality. It is defined as the number of training DOM trees used to learn this wrapper.

First, a level-1 wrapper T_w is converted from a randomly chosen DOM tree and taken as the center for this template (Figure 5(a)). DOM trees whose distance to T_w is less than a given threshold ϵ (dashed circle in Figure 5) are considered belonging to the same template of T_w and will be used to refine T_w. Refining T_w with a DOM tree T_d includes three steps: converting T_d to a level-1 wrapper $T_w^{'}$; generating a new wrapper from T_w and $T_w^{'}$; and replacing T_w with the new wrapper.

After T_w is refined, it is upgraded by one level and becomes more general. Actually, for any DOM tree T_d, the recalculated distance between T_w and T_d is expected to decrease. For the DOM trees that match the wrapper perfectly (e.g., central DOM tree in Figure 5(c)), we will not use them to refine the centered wrapper because they will not bring any changes to it. For those DOM trees whose distance to the centered wrapper is less than threshold ϵ, T_d will be employed to refine T_w (Figure 5(a) and Figure 5(b)).

Finally, the WPC algorithm stops for one template when no DOM tree is within the given threshold (Figure 5(d)).

The full algorithm of WPC is listed in Figure 6.

Our proposed WPC algorithm has only one parameter, i.e., the distance threshold. Fortunately, there is a wide threshold to assure high performance. Please refer to experimental results shown in Section 6.

5.2.2 Distance Metric

In our system, instead of measuring the similarity between two DOM trees directly, we derive a *Wrapper-DOM Distance* (WD-Distance) to measure the distance between a wrapper and a DOM tree. This distance is used in both the wrapper-oriented page clustering module and the wrapper selection module.

WD-Distance is calculated based on the WD-alignment's cost. WD-alignment algorithm is similar to WW-alignment. Thus, we will not describe it in detail but only present the difference between them. In WW-alignment, nodes are aligned in a one-to-one manner; while in WD-alignment a wrapper node whose sign is +, * or N can be aligned with multiple DOM nodes (Figure 7).

For WD-alignment between a wrapper T_w and a DOM tree T_d, we use $\mathcal{C}_w(T_w, T_d)$ to denote the total cost caused by mismatched wrapper nodes and use $\mathcal{C}_d(T_w, T_d)$ to denote the total cost caused by mismatched DOM nodes. When

```
Algorithm: WPC($D$: DOM tree set, $\epsilon$: threshold)
1.  begin
2.      $\mathbb{R}$ := page cluster set;
3.      $\mathbb{W}$ := wrapper set;
4.      while $D$ is not empty
5.          create a new page cluster $C$;
6.          select a DOM tree $T_{d_1}$ from $D$ randomly;
7.          $T_{d_1} \rightarrow T_{w_1}$;
8.          move $T_{d_1}$ from $D$ to $C$;
9.          for each $T_d$ in $D$
10.             if $\Psi(T_{w_1}, T_d) = 0$
11.                 move $T_d$ from $D$ to $C$;
12.             endif
13.         endfor
14.         while $\exists T_{d_2} \in D : \Psi(T_{w_1}, T_{d_2}) < \epsilon$
15.             $T_{d_2} \rightarrow T_{w_2}$;
16.             $T_{w_1} + T_{w_2} \rightarrow T_{w_3}$;
17.             $T_{w_1} := T_{w_3}$;
18.             move $T_{d_2}$ from $D$ to $C$;
19.             for each $T_d$ in $D$
20.                 if $\Psi(T_{w_1}, T_d) = 0$
21.                     move $T_d$ from $D$ to $C$;
22.                 endif
23.             endfor
24.         endwhile
25.         add $C$ to $\mathbb{R}$, add $T_{w_1}$ to $\mathbb{W}$;
26.     endwhile
27.     return $\mathbb{R}$ and $\mathbb{W}$;
28. end
```

Figure 6: Wrapper-oriented page clustering algorithm

calculating the WD-Distance, it is necessary to normalize the cost $\mathcal{C}_w(T_w, T_d)$ and $\mathcal{C}_d(T_w, T_d)$ by the weight of a whole tree because the larger the number of nodes in the tree, the greater is the likelihood of the tree having nodes that are mismatched. Thus, Wrapper-DOM Distance is defined as follows:

DEFINITION 4. *(Wrapper-DOM Distance) Given a wrapper T_w and a DOM tree T_d, the wrapper-DOM distance*

$$\Psi(T_w, T_d) = \Big(\frac{\mathcal{C}_w(T_w, T_d)}{\mathcal{W}(T_w)} + \frac{\mathcal{C}_d(T_w, T_d)}{\mathcal{W}(T_d)}\Big)\Big/2$$

Here, weight of a DOM tree and weight of a wrapper are defined as below:

DEFINITION 5. *(DOM-Tree Weight) Given a DOM tree T_d whose root node is τ, then $\mathcal{W}(T_d) = \mathcal{W}(\tau)$*

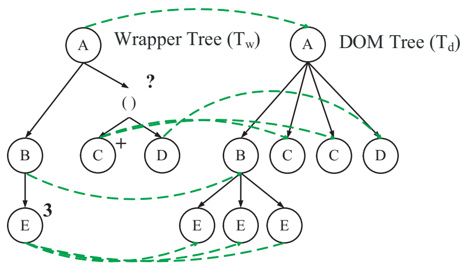


Figure 7: Alignment of a wrapper with a DOM tree

Definition 6. *(Wrapper Weight) Given a wrapper T_w whose root node is σ, then $\mathcal{W}(T_w) = \mathcal{W}(\sigma)$*

According to Definition 4, WD-distance is the arithmetic mean of the normalized cost caused by the wrapper side and that caused by the DOM-tree side. Thus, values are normalized in the range between 0 and 1. $\Psi(T_w, T_d) = 0$ means T_w perfectly matches T_d without any cost, and $\Psi(T_w, T_d) = 1$ means none of the nodes in T_w and T_d match in the alignment.

6. EXPERIMENTS

We test the performance of our approach through experiments.

6.1 Experiment Set-up

We use a dataset of 1,700 product pages from Amazon.com and a dataset of mixed 1,000 pages from 10 shopping websites. We call the former dataset Amazon and the latter M10 hereinafter. The reason we choose these datasets is that they are real-life large-scale Web sites and their Web pages are relatively more complex than datasets used in earlier works. Although we can also gain very good results on simple datasets, i.e., all pages are similar of each site, using complex real-life datasets is a better choice to show the effectiveness of our joint approach.

In each page, product records and their three attributes, namely product name, product image, and product price, are manually labeled. For each website, twofold cross validation is conducted. We use precision, recall, and F1 as measures in evaluation of data extraction results. These measures are calculated based on whether a labeled node is correctly extracted.

All experiments were run on a PC, with a 3.06 GHz Pentium 4 processor and 3.87 GB RAM.

6.2 Experimental Results

Experiment I: Effectiveness Test

On the Amazon data, our joint optimization approach achieves as high F1 as 94.88% by setting the threshold as 0.3 (Figure 8) with 44 wrappers generated. For comparison, we implement the separated template detection strategy based on URLs. For specific, training pages are divided into several templates by their URLs. Then, each template generates a wrapper using the same wrapper induction technique as that used in our WPC algorithm. The experimental result shows that these wrappers can only achieve 78% accuracy in terms of F1. Therefore, our approach outperforms the traditional method by about 17 points.

To further evaluate the performance of the WPC algorithm, we run the experiment on M10 data. Table 1 shows the evaluation results for each site. As we see, the average F1 is as high as 97.2%. For seven sites out of 10, the proposed WPC algorithm achieves F1 higher than 98%. The lowest F1 is got on pages from ftd.com. By case study, we find that the number of templates are unbalanced between the training set and the test set. When some templates are unseen in the stage of training, the generated wrappers reject those pages and extract nothing in testing. But for those seen templates, the wrappers can handle them well. That is why we get high precision and low recall.

Table 1: Results on 10 shopping websites

Website	Wra. #	Pre.	Rec.	F1
ashford.com	1	1.0000	1.0000	1.0000
circuitcity.com	2	1.0000	1.0000	1.0000
costco.com	23	0.9667	0.9153	0.9403
diamond.com	1	0.9875	0.9975	0.9925
ebags.com	2	0.9976	1.0000	0.9988
ftd.com	2	0.9833	0.7528	0.8527
officedepot.com	1	0.9850	1.0000	0.9924
overstock.com	6	0.9224	0.9979	0.9587
pricegrabber.com	1	0.9970	0.9773	0.9870
sears.com	3	0.9960	1.0000	0.9980
Average	4.2	0.9835	0.9640	0.9720

We notice that pages in site Costco.com are clustered into 23 similarity-based templates. It is surprising because in the viewpoint of a human, training pages of this site share only one template. Then, we use all training pages of this site to generate one wrapper. The wrapper can only achieve 87% accuracy in terms of F1, which is lower than the 23 wrappers generated by our approach.

Experiment II: WPC with Different Thresholds

We evaluate the performance of WPC algorithm as the threshold changing on Amazon data. We run WPC 18 times under different thresholds: from 0 to 0.85 with a 0.05 interval. Figure 8 shows three curves on performance when the threshold increases from 0 to 0.85. There is no result with greater thresholds because pages chosen are too diverse to learn a wrapper.

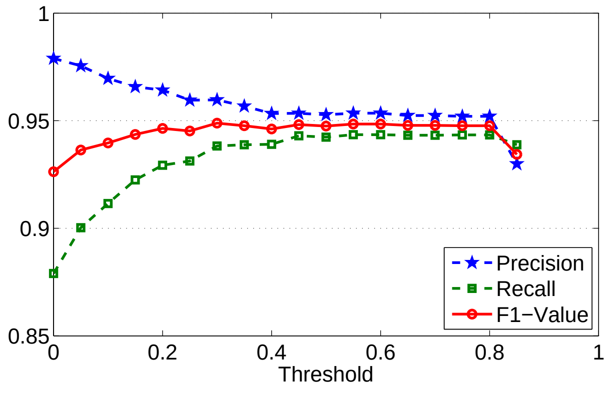


Figure 8: WPC performance under different thresholds

When the threshold is set as 0, only exactly matched pages can be absorbed by the wrapper. Thus, we got the highest precision while the recall is the lowest. It means that a generated wrapper is specific for the small number of pages although the wrapper is precise in its scope.

As we increase the threshold, the precision drops down and the recall goes up. In terms of F1, the peak value of 94.88% is achieved by setting the threshold to 0.3. After that, F1 stays above 94% and becomes stable until the threshold is set to 0.85. The stable range, from 0.3 to 0.8 indicates that it is not hard to set an appropriate fixed threshold in the approach.

We also list comparison of the number of wrappers generated with different thresholds in Figure 9. The number of wrappers or similarity-based templates decreases quickly from 832 to 44 as the threshold increases from 0 to 0.3. Then the wrapper number decreases slowly. There is an obvious drop if the threshold is set to 0.85. All training pages are clustered into just four. Such wrappers can cover more pages by sacrificing the effectiveness of extraction.

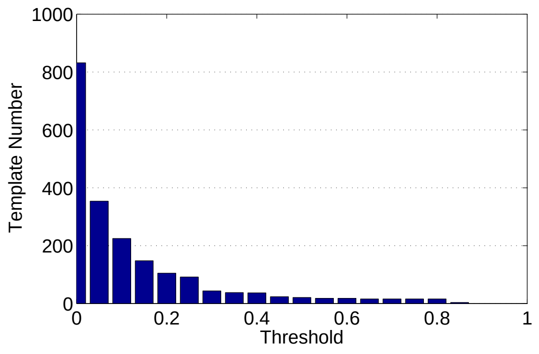


Figure 9: Template detection under different thresholds

The impact of different thresholds is also presented by the runtime of our algorithm in the wrapper generation process. When the threshold is set to 0, it takes 13,197 seconds (3.67 hours) to generate 832 wrappers. Then it drops to 1,424 (0.40 hours) seconds when the threshold increase to 0.15 and keeps stable around 2,000 seconds (0.56 hours) until the threshold reaches 0.8. After that, the runtime increases dramatically to 24,666 seconds (6.85 hours) when the threshold is set to 0.85. The runtime gets too much to tolerate when the threshold is greater. So we treat the situations as if they fail to learn a wrapper.

Experiment III: Stability Test

Since our algorithm chooses the initial DOM tree for clustering in a random way, we evaluate how the initial choice impacts performance of our approach in the experiment. We conducted WPC algorithm five times with the threshold set to 0.2 on Amazon dataset. Table 2 lists the number of template, extraction precision and recall of each run.

As Table 2 shows, in terms of F1, the mean is 94% while the standard variance is smaller than 4E-5. It indicates that our proposed approach is quite stable with the random strategy to select an initial DOM tree.

Table 2: Stability test result

	Template #	Precision	Recall	F1
1	99	0.9683	0.9249	0.9461
2	116	0.9460	0.9254	0.9356
3	111	0.9606	0.9238	0.9418
4	113	0.9510	0.9138	0.9320
5	111	0.9664	0.9236	0.9445

Experiment IV: Labeling Cost

As stated earlier, our approach requires manually labeling for wrapper generation. This experiment was designed to show how many training pages are required for learning wrappers to achieve an accuracy higher than 95% in terms of F1. Table 3 shows the results for all sites in M10. Actually, most websites only need a handful of labeled pages to meet the demand of accuracy. That proves that the cost of manually labeling in our approach is acceptable.

Table 3: Labeling test result

Website	Page #	Website	Page #
ashford.com	12	circuitcity.com	19
costco.com	31	diamond.com	12
ebags.com	19	ftd.com	N/A
officedepot.com	19	overstock.com	16
pricegrabber.com	7	sears.com	27

7. RELATED WORK

Our work is in the area of *Web Information Extraction*. It is highly related to previous works on wrapper induction. Several automatic or semi-automatic wrapper learning methods have been proposed. For example, WIEN [12] is the earliest method as we know on automatic wrapper induction. Other representative ones are SoftMeley [11], Stalker [17], RoadRunner [6], EXALG [2], TTAG [4], works in [18] and ViNTs [21]. Here, we only discuss RoadRunner, TTAG, and works in [18] and refer the reader to two surveys [13, 8] and two tutorials [19, 14] for more studies related to information extraction and wrapper induction.

In previous work, page clustering for wrapper induction is considered a trivial task in most previous wrapper induction systems. Among them, only RoadRunner [6, 7] and [18] proposed automatic approaches to implement page clustering for wrapper generation. Other methods all manually collect training pages, template by template.

In [7], page clustering for RoadRunner system is discussed. They use four types of page features to calculate the similarity between two pages: (1) distance from the home page; (2) url similarity; (3) tag probability; (4) tag periodicity. Based on page similarity, they adopt a popular non-supervised clustering algorithm *MiniMax* to conduct the page clustering process. This process is isolated from the wrapper generation process, which is the primary difference compared with our WPC algorithm. Wrapper selection problem is also discussed in [7].

In [18], tree edit distance is used to measure the distance between two pages. They use traditional hierarchical clustering techniques [20] in which the distance measured is the output of a restricted top-down tree mapping algorithm

(RTDM). The RTDM algorithm does not distinguish node tag and it is designed only for finding the main contents in news pages. This restricts that method from being applied to general information extraction problems. Similar to our system, works in [18] can also derive a similarity between a wrapper (called extraction patterns) and a page when selecting a proper wrapper for extracting data from a new page. However, template detection is still isolated from the wrapper generation process.

We need to mention TTAG because wrappers in TTAG are also presented as a tree structure with wildcards. The authors also employ a top-down layer-by-layer alignment, but the alignment in any layer is isolated from that in other layers. As a result, child nodes can still be aligned even when their parent nodes do not match. That is a different strategy from ours. Moreover, like most other previous systems, template detection is not discussed in TTAG.

8. CONCLUSIONS AND FUTURE WORK

In this paper, we propose a novel wrapper induction system that expresses a different opinion regarding the relation between template detection and wrapper generation. Our system takes a miscellaneous training set as input and conducts template detection and wrapper generation in a single step. By the criterion of generated wrappers' extraction accuracy, our approach can achieve a joint optimization of template detection and wrapper generation. Experimental results on real-life shopping websites prove the feasibility and effectiveness of our approach. The preliminary comparison demonstrates that our approach significantly outperforms the separated template detection strategy.

Our wrapper induction algorithm only leverages the HTML tag-tree structure and does not involve any content. As future work, we will try to extend the approach to handle the templates that contain some content strings.

9. ACKNOWLEDGMENTS

We are grateful to Dwight Daniels for detailed edits on writing this paper. Comments from the three anonymous referees are invaluable for us to prepare the final version.

10. REFERENCES

[1] http://www.w3.org/dom/.

[2] A. Arasu and H. Garcia-Molina. Extracting structured data from web pages. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, pages 337 – 348, 2003.

[3] C.-H. Chang and S.-C. Lui. Iepad: information extraction based on pattern discovery. In *Proceedings of the 10th International Conference on World Wide Web*, pages 681–688, 2001.

[4] S.-L. Chuang and J. Y.-j. Hsu. Tree-structured template generation for web pages. In *Proceedings of IEEE/WIC/ACM International Conference on Web Intelligence*, pages 327 – 333, 2004.

[5] W. W. Cohen, M. Hurst, and L. S. Jensen. A flexible learning system for wrapping tables and lists in html documents. In *Proceedings of the 11th International Conference on World Wide Web*, pages 232 – 241, 2002.

[6] V. Crescenzi, G. Mecca, and P. Merialdo. Roadrunner: Towards automatic data extraction from large web sites. In *Proceedings of the 27th International Conference on Very Large Data Bases*, pages 109 – 118, 2001.

[7] V. Crescenzi, G. Mecca, and P. Merialdo. Wrapping-oriented classification of web pages. In *Proceedings of the 2002 ACM symposium on Applied computing*, pages 1108 – 1112, 2002.

[8] S. Flesca, G. Manco, E. Masciari, E. Rende, and A. Tagarelli. Web wrapper induction: a brief survey. *AI Communications*, 17:57 – 61, 2004.

[9] J. Hammer, H. Garcia-Molina, J. Cho, A. Crespo, and R. Aranha. Extracting semistructured information from the web. In *Proceedings of the Workshop on Management fo Semistructured Data*, 1997.

[10] A. Hogue and D. Karger. Thresher: automating the unwrapping of semantic content from the world wide web. In *Proceedings of 14th International Conference on World Wide Web*, pages 86 – 95, 2005.

[11] C.-N. Hsu and M.-T. Dung. Generating finite-state transducers for semi-structured data extraction from the web. *Information Systems, Special Issue on Semistructured Data*, 23(8):521–538, 1998.

[12] N. Kushmerick, D. S. Weld, and R. B. Doorenbos. Wrapper induction for information extraction. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 729–737, 1997.

[13] A. H. F. Laender, B. A. Ribeiro-Neto, A. S. da Silva, and J. S. Teixeira. A brief survey of web data extraction tools. *SIGMOD Record*, 31(2):84–93, 2002.

[14] B. Liu. Web content mining (tutorial). In *Proceedings of the 14th International Conference on World Wide Web*, 2005.

[15] B. Liu, R. Grossman, and Y. Zhai. Mining data records in web pages. In *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 601 – 606, 2003.

[16] L. Liu, C. Pu, and W. Han. Xwrap: an xml-enabled wrapper construction system for web information sources. In *Proceedings of the 16th International Conference on Data Engineering*, pages 611–621, 2000.

[17] I. Muslea, S. Minton, and C. Knoblock. A hierarchical approach to wrapper induction. In *Proceedings of the 3rd Annual Conference on Autonomous Agents*, pages 190 – 197, 1999.

[18] D. C. Reis, P. B. Golgher, A. S. Silva, and A. F. Laender. Automatic web news extraction using tree edit distance. In *Proceedings of the 13th International Conference on World Wide Web*, pages 502 – 511, 2004.

[19] S. Sarawagi. Automation in information extraction and data integration (tutorial). In *Proceedings of the 28th International Conference on Very Large Data Bases*, 2002.

[20] P. Willett. Recent trends in hierarchic document clustering: a critical review. *Information Processing and Management*, 24(5):577–597, 1988.

[21] H. Zhao, W. Meng, Z. Wu, V. Raghavan, and C. Yu. Fully automatic wrapper generation for search engines. In *Proceedings of the 14th International Conference on World Wide Web*, pages 66 – 75, 2005.