High-Precision Web Extraction Using Site Knowledge

Meghana Kshirsagar, Rajeev Rastogi, Sandeepkumar Satpal, Srinivasan H Sengamedu, Venu Satuluri

Yahoo! Labs, Bangalore, India

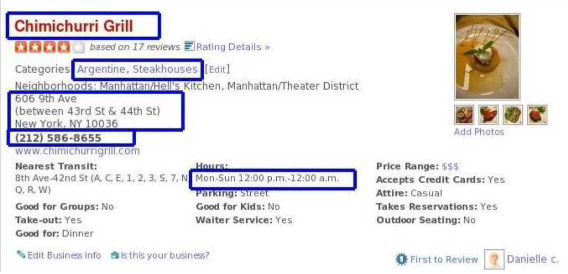
Abstract

In this paper, we study the problem of extracting *structured records* from semi-structured Web pages. Existing Web information extraction techniques like *wrapper induction* require a large amount of editorial effort for annotating pages. Other schemes based on *Conditional Random Fields* (CRFs) suffer from precision loss due to variable site structures and abundance of noise in Web pages. In this paper, we propose novel techniques that *exploit site knowledge for high-precision extraction with very few training examples.* We leverage site knowledge in an *unsupervised* manner in two ways. First, we use static repeating text in the pages of a Web site to segment long noisy Web page sequences into short segments with little noise. Then, after labeling the segments using a generic classification technique like CRFs, we exploit inter-page structure similarity along with attribute uniqueness constraints and proximity relationships to correct erroneous labels. Applied together, our unsupervised schemes exploiting site knowledge achieve 5 times higher accuracy compared to traditional CRF-based approaches for real-world restaurant pages.

1 Introduction

In this paper, we study the problem of extracting *structured records* from semi-structured Web pages belonging to thousands of Web sites. As an example, consider the page shown in Figure 1 for restaurant "Chimichurri Grill" from the aggregator Web site `www.yelp.com`. The page contains a wealth of information including details like the restaurant name, category, address, phone number, hours of operation, user reviews, etc. We are looking to extract this information from such *detail* pages, and store the extracted data for each page as attributes of a record as shown below.

Name	Category	Address	Phone	...
Chimichurri Grill	Argentine Steakhouses	606 9th Ave NY 10036	(212) 586-8655	...
21 Club	American	21 West 52^{nd} St NY 10019	(212) 582-7200	...



Chimichurri Grill

based on 17 reviews

Categories Argentine, Steakhouses [Edit]

Neighborhoods: Manhattan/Hell's Kitchen, Manhattan/Theater District

606 9th Ave
(between 43rd St & 44th St)
New York, NY 10036

(212) 586-8655

www.chimichurrigrill.com

Nearest Transit: 8th Ave-42nd St (A, C, E, 1, 2, 3, S, 7, N, Q, R, W)

Hours: Mon-Sun 12:00 p.m.-12:00 a.m.

Parking: Street

Price Range: $$$

Accepts Credit Cards: Yes

Attire: Casual

Good for Groups: No

Take-out: Yes

Good for: Dinner

Good for Kids: No

Waiter Service: Yes

Takes Reservations: Yes

Outdoor Seating: No

Figure 1: Example restaurant detail page.

A significant fraction of pages belong to Web sites that use automated scripts to dynamically populate pages from a back-end DBMS. These sites have thousands or even millions of Web pages with *fixed templates* and very similar structure. Examples of such sites include retailer Web sites like `www.amazon.com`, aggregator Web sites like `www.yelp.com`, news sites like `www.cnn.com`, video sites like `www.youtube.com`, and so on. Our experimental study on a crawled repository of 2 billion Web pages shows that over 30% of pages occur in clusters of size ≥ 100 with pages in each cluster sharing a common template. Thus, template-based pages constitute a sizeable portion of the Web, and we focus on extracting records from such pages in this paper.

1.1 Our Approach

As observed above, existing Web extraction methods either require large numbers of training examples or suffer from low precision. In this paper, we propose a new Web extraction approach that exploits site knowledge to achieve high precision while requiring only a few Web pages to be annotated by humans.

In our approach, we construct the training set by selecting sample pages from a few initial Web sites and having editors assign labels to attribute text in the sample pages. For each site, we learn wrappers from the samples and use them to label attribute values in the remaining Web pages of each of the few initially chosen Web sites. Thus, constructing the training set requires only a small number of sample pages belonging to a few Web sites to be annotated by humans, and so our approach incurs low overhead.

Next, we build CRF models from the training set, but augment our CRF-based extraction with novel pre- and post- processing steps that exploit site knowl-

International Conference on Management of Data
COMAD 2010, Nagpur, India, December 8–10, 2010

edge to boost prediction accuracy. Our pre-processing step identifies static repeating text across the pages of a Web site which we have found to be abundant in most Web pages. For instance, in Figure 1, the text strings "Categories:", "Nearest Transit:", "Price Range:", "Send to Friend", "Write a Review", etc. are all examples of static text. We exploit this static text in two ways. First, this static text is typically noise, and so it can be filtered out when extracting attribute values. Second, attribute values very rarely span across static text and so we can use the static text to segment Web pages. Thus, our pre-processing step creates short sequences with little noise from longer Web page sequences with lots of noise. This is then input to our CRF learning and labeling algorithms, thus boosting their accuracy.

For each new Web site, we use our CRF models to label the page segments from the site. Since the new Web site may be different from previous sites in terms of structure and content, many Web page elements may be assigned wrong attribute labels. We correct these erroneous labels in our post-processing step that exploits intra-page and inter-page constraints.

- *Intra-page constraints*. Frequently, attributes in a Web page satisfy uniqueness constraints and proximity relationships. For instance, in the restaurant Web pages in Figures 1, attributes like name, category, address, phone number, hours of operation, etc. have a single contiguous value, and further these attributes tend to appear in close proximity to each other. Thus, if there are multiple occurrences of an attribute label in a Web page, we can improve the accuracy by selecting the one that is closest to the other attributes.

- *Inter-page constraints*. As mentioned earlier, pages belonging to many of the popular Web sites are script-generated, and thus have fixed templates. As a result, we can exploit the structural similarity between pages of a Web site to adjust label values in similar locations (across the Web pages) to the majority label. The intuition here is that our CRF models will assign correct labels in a majority of cases, and thus applying a "wisdom of crowds" approach in such a scenario will help to correct the wrong labels (which in most likelihood will constitute a minority).

Note that existing machine learning techniques like CRFs cannot easily capture the above kinds of constraints (see Section 2 for a detailed discussion). Also, observe that the constraints typically hold at a coarser page granularity for detail pages. For list pages containing multiple data records, the same constraints hold at a finer record granularity. So for simplicity, we will only consider extraction from detail pages in this paper. And finally, we would like to point out that our proposed framework can be used in conjunction with other classification techniques as well in addition to CRFs (e.g., SVMs).

1.2 Our Contributions

Our main contributions can be summarized as follows.

(1) We present an end-to-end system design for high-precision Web information extraction using site knowledge. Our system requires only pages from a few initial Web sites to be annotated by humans. Extraction from subsequent sites is completely automated and does not require additional manual annotations; thus, our approach incurs low editorial overhead.

(2) We devise unsupervised pre-processing schemes to filter out noise and segment Web pages into shorter sequences using static repeating text across the pages of a Web site.

(3) We develop unsupervised methods for post-processing the segment labels assigned by any generic classifier (e.g., CRFs). Our scheme boosts accuracy by enforcing uniqueness constraints and exploiting proximity relationships among attributes to resolve multiple occurrences in a Web page. Unfortunately, the problem of selecting attribute labels that are closest to each other is NP-hard, and so we use a heuristic for attribute selection. Our scheme also exploits the structural similarity of pages to fix incorrect label values. To deal with structural variations among pages (e.g., due to missing attribute values), it employs the idea of edit distance to align labels across pages, and sets each label to the majority label for the location.

(4) We demonstrate the efficacy of our approach using CRFs as the underlying classifier. We conduct an extensive experimental study with real-life restaurant pages to compare the performance of our techniques with a baseline CRF method. Our results indicate that our pre-processing schemes improve accuracy by a factor of 4 compared to CRFs, and with the post-processing step included as well, we get a further accuracy gain of 40%.

The remainder of the paper is organized as follows. In Section 2, we survey related work on Web information extraction. We formally define our extraction problem in Section 3. We describe details of our extraction algorithm in Section 4. In Section 5, we present the results of our experimental study. Finally, we offer concluding remarks in Section 6.

2 Related Work

Existing Web information extraction techniques can be classified into two categories: (1) *Structure-based* techniques that rely only on the layout of pages and the HTML tag structure to extract data, and (2) *Content-based* techniques that also exploit the content of HTML elements in the page (in addition to the tag structure).

Early proposals for extracting data from Web pages were all structure-based. *Wrapper induction* [8, 10], one of the first approaches, relies on humans to annotate a few example pages from each Web site.

Wrapper-based solutions, however, cannot scale to tens of thousands of sites with different templates because manually labeling pages from so many sites can be labor intensive, time consuming, and expensive.

To alleviate the above problem with wrappers, [6, 2, 13] propose unsupervised techniques for extracting data from Web pages without manually labeled examples. [6, 2] automatically learn the common Web page template (containing optional and repeated element occurrences) for a site by analyzing multiple pages of the site. Zhang and Liu [13] present a two-step approach called DEPTA for automatically finding several similar structured data records in a single list Web page.

In contrast to the above approaches which primarily leverage site-specific structural cues, there is also a large body of literature on sequential models which exploit content cues in a site-independent manner. Zhu et al. [15] propose a content-based technique that relies on a graphical model called Hierarchical Conditional Random Field (HCRF) to assign (attribute) labels to (leaf) nodes of the HTML tree corresponding to a Web page. A drawback of CRFs with complex graph structures like hierarchies [15] or grids [14] is that they have many more parameters that need to be trained compared to linear-chain CRFs. The situation is further exacerbated by the fact that Web pages contain plenty of noise, and have diverse structures that can vary significantly from one site to the next. As a result, HCRFs can have low precision and our experimental results in Section 5 corroborate this. Furthermore, HCRFs do not incorporate constraints related to attribute uniqueness and page template similarity when assigning labels. Thus, combining our schemes with HCRFs can help to improve their extraction accuracy.

[4, 1, 3] use machine learning techniques like Hidden Markov Models (HMMs) and CRFs for automatically segmenting unformatted text such as addresses or bibliography items into structured records. [5] presents the Turbo Syncer framework for segmenting data records contained in multi-record Web pages which is very similar to the text segmentation problem. However, the schemes in these papers are developed for short similar structured text strings containing little or no noise. Consequently, they are unlikely to work well for much longer and noisy Web pages from a multitude of sites with diverse structure.

There is very little work in the research literature on label assignment in the presence of constraints. [11] proposes a novel inference procedure for CRF models based on integer linear programming (ILP) that allows certain constraints like uniqueness, existential, etc. to be enforced when computing the label sequence with the highest probability. While this is an interesting research direction, an ILP-based inference procedure has two drawbacks. First, it has exponential complexity and so may be impractical, from a computational perspective, for longer Web page sequences. Second, while ILPs can capture uniqueness constraints, it is unclear if they are expressive enough to capture intra-page proximity and inter-page layout similarity constraints that arise in a Web data extraction setting. Recently *Markov Logic Networks* (MLNs) [12] have been used for information extraction in the presence of constraints expressed as first-order formulas. However, while MLNs have greater expressive power, this comes at the expense of significantly higher computational overhead (due to the large number of groundings of the first-order formulas).

To the best of our knowledge, ours is the first work to extract records from noisy pages of large numbers of Web sites while requiring only a small sample of pages from a few initial sites to be annotated. We achieve high-precision extractions from new Web sites (without further site-specific annotations) through novel techniques that segment Web pages, and enforce attribute uniqueness, proximity, and site-level layout similarity constraints.

3 System Model

We consider the problem of extracting attribute values from Web sites belonging to a single vertical, e.g., restaurants. We will use $\mathcal{W}$ to denote the set of Web sites belonging to the vertical of interest, and we will denote the attributes for the vertical by $A_1, \ldots, A_m$. Each Web site $W \in \mathcal{W}$ consists of a set of *similar structured*[1] detail pages from each of which we extract a single record. It is important to note here that the pages in a Web site have similar but not identical structure. The structural variations between pages arise primarily due to missing attribute values. In addition to attribute values, Web pages contain plenty of *noise* which we denote using the special attribute A_0.

We model each Web page as a sequence of words obtained as a result of concatenating the text in the leaf nodes of the page's DOM tree (in the order in which they appear in the tree). When convenient, we will consider an alternate representation of a page as a sequence of leaf nodes from which the word sequence is derived. Each node has an associated XPath expression (without position information) that captures the location of the node in the page. Each node also has a unique identifier equal to the (text, XPath) pair for the node – this id is unique within the detail page containing the node[2].

In most pages, the text contained in a node is part of a single attribute value; thus, all words of a node have the same attribute label. We will denote the label for

[1] In reality, for certain large Web sites like www.amazon.com, there may be many different scripts that generate pages with different structures. In such a scenario, we treat each cluster of pages with similar structure as a separate Web site.

[2] If there are multiple nodes with identical ids, then we can ensure uniqueness by numbering them.

node n by $lbl(n)$. Furthermore, an attribute value may not be restricted to a single node, but rather may span multiple (consecutive) nodes. For instance, consider the address attribute. It can be formatted differently across Web sites – either as one monolithic node that includes street name, city, zip, etc. or a different one in which street name, city, zip, etc. are split across different nodes.

Example 3.1 Consider the following (simplified) HTML code fragment for a portion of the page in Figure 1.

<body>
<p> Yelp </p>
<h1> Chimichurri Grill </h1>
<p> <strong> Categories: </strong> Argentine, Steakhouses </p>
<div> 606 9th Ave <br> NY 10036 <br> </div>
<span> (212) 586-8655 </span>
</body>

The page is a sequence of 7 (DOM tree leaf) nodes with text, XPaths, and labels as shown below.

Node	Text	XPath	Label
n_1	Yelp	/body/p	Noise
n_2	Chimichurri Grill	/body/h1	Name
n_3	Categories:	/body/p/strong	Noise
n_4	Argentine, Steakhouses	/body/p	Category
n_5	606 9th Ave	/body/div	Address
n_6	NY 10036	/body/div	Address
n_7	(212) 586-8655	/body/span	Phone

Nodes n_1, n_3, and n_5 have ids (“Yelp”, /body/p), (“Categories:”, /body/ p/strong), and (“606 9th Ave”, /body/div), respectively. And finally, the page contains the following word sequence: “Yelp Chimichurri Grill Categories: Argentine, Steakhouses 606 9th Ave NY 10036 (212) 586-8655”. □

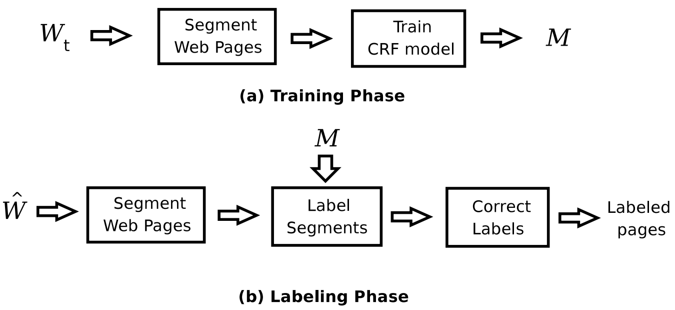
The input to our Web extraction system is a small subset of training Web sites $\mathcal{W}_t \subseteq \mathcal{W}$. Each node of a page belonging to a Web site in $\mathcal{W}_t$ has an associated attribute label. The node labels are obtained using wrappers learnt from a small sample of human annotated pages. These labeled page sequences belonging to sites in $\mathcal{W}_t$ serve as the input training data for our extraction algorithms. Our Web information extraction problem can be stated as follows.

Web information extraction problem: Given a set of labeled Web sites $\mathcal{W}_t \subseteq \mathcal{W}$, for a new Web site $\hat{W} \in \mathcal{W} - \mathcal{W}_t$, assign attribute labels $A_0, \dots, A_m$ to nodes of Web page sequences in $\hat{W}$. □

4 Information Extraction Algorithm

4.1 Overview

Our extraction system employs linear-chain CRFs [9] to label attribute occurrences in Web pages. A linear-chain CRF models the conditional probability distribution $P(\mathbf{l}|\mathbf{w})$, where $\mathbf{w} : \langle w_1 w_2 \cdots w_T \rangle$ is a sequence of words and $\mathbf{l} : \langle l_1 l_2 \cdots l_T \rangle$ is the corresponding label sequence. The conditional probability distribution is given by,



(a) Training Phase

(b) Labeling Phase

Figure 2: Key steps of training and labeling phases.

$$P(\mathbf{l}|\mathbf{w}) = \frac{1}{Z(\mathbf{w})} \exp(\sum_{t=1}^{T} \sum_{k=1}^{K} \lambda_k f_k(l_{t-1}, l_t, \mathbf{w}, t))$$

where $f_1, f_2, \dots, f_K$ are feature functions, λ_k is the weight parameter for feature f_k, and $Z(\mathbf{w})$ is the normalization factor. An example feature function is $f_k(l_{t-1}, l_t, \mathbf{w}, t) = 1$ if l_{t-1} = Address, l_t = Address and $w_t \in$ `5digitNumber`. It fires if the previous and current words are both labeled Address, and the current word is a 5-digit number (e.g., zip code). During training, the parameters λ_k of the CRF are set to maximize the conditional likelihood of the training set $\{(\mathbf{w}^i, \mathbf{l}^i)\}$. Then, for an arbitrary input sequence $\mathbf{w}$, inference of the label sequence $\mathbf{l}$ with the highest probability is efficiently carried out using the Viterbi algorithm. Further details of CRF training and inference algorithms can be found in [9].

Our extraction algorithm has two phases: the *training* phase and the *labeling* phase. The training phase uses the training data from sites in $\mathcal{W}_t$ to train a linear-chain CRF model, while the labeling phase assigns attribute labels to the pages of $\hat{W}$. Figure 2 depicts the key steps of both phases. As can be seen from the figure, both phases perform a pre-processing step where input page sequences are split into short sequences. In the training phase, the labeled page segments are used to train a CRF model. This model is then employed in the labeling phase to assign attribute labels to individual nodes in the page segments of $\hat{W}$. Since many of these assigned labels may be incorrect, we perform a final post-processing step in the labeling phase where we fix some of the incorrect labels.

In the following subsections, we describe the two phases of our extraction system in more detail. But first, we describe the segmentation step that is common to both phases.

4.2 Segmenting Web Pages

Web pages belonging to a site typically contain a fair amount of text that repeats across the pages of the site. We leverage this static text to segment Web pages

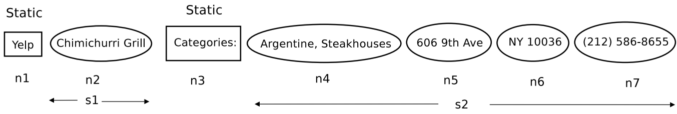


Figure 3: Example page segments.

from a site W by performing the following two steps.
1. *Identifying static nodes.* We say that a node n in a page p (of W) is static if a significant fraction α of pages in W contain nodes with the same id (that is, text and XPath) as n. We can detect static nodes by storing all the nodes in a hash table indexed by their ids. Let N be a set of nodes with the same id in a hash bucket. We mark the nodes in N as static if they occur in at least α fraction of pages in W. Empirically, we have found that a good setting for α is 0.8.
2. *Segmenting pages.* Consider a page p in W. We partition p into segments using static nodes, and treat each node sequence between any two consecutive static nodes as a separate segment. More formally, suppose that $n_j, n_{j+1}, \ldots, n_q$ is a subsequence of p such that n_j and n_q are static nodes, and $n_{j+1}, \ldots, n_{q-1}$ are not. Then $n_{j+1}, \ldots, n_{q-1}$ is a segment with id equal to node n_j's id. Thus, each segment s has an id that is equal to the id of the static node preceding s in p.

Observe that there may be multiple segments with the same id across the pages of a Web site. However, there is at most one segment with a fixed id e per page. Furthermore, attributes that satisfy the uniqueness constraint (e.g., restaurant name, address) lie entirely within a single segment although they may span one or more consecutive nodes within the segment. On the other hand, attributes with multiple occurrences like user reviews (or noise) may span multiple segments. Finally, due to page structure similarity, each attribute occurs in segments with the same id(s) across the pages.

In the page in Example 3.1, nodes n_1 and n_3 with text "Yelp" and "Categories:" are static nodes. These partition the page into two segments $s_1 = n_2$ and $s_2 = n_4 \cdot n_5 \cdot n_6 \cdot n_7$ with ids ("Yelp", */body/p*) and ("Categories:", */body/p/strong*), respectively. This is pictorially depicted in Figure 3. Observe that the word sequences in s_1 and s_2 are "Chimichurri Grill" and "Argentine, Steakhouses 606 9th Ave NY 10036 (212) 586-8655", respectively.

Training CRFs and labeling using segments instead of full page sequences leads to higher accuracy. This is because segmentation filters out static nodes that are essentially noise. It also ensures that attribute occurrence patterns of the training Web sites is not reflected in the CRF model. This leads to more accurate labeling because the structure of the new Web site $\hat{W}$ can be very different from the training set. And finally, labeling over segments ensures that errors in assigning labels in one segment do not propagate to other segments. As we will see later, segments are also useful for fixing incorrect attribute labels since we enforce uniqueness, proximity and structural similarity constraints at a segment level.

Note that in addition to static nodes, we can also use static repeating text like "Price:", "Address:", "Phone:", etc. that occurs in nodes with identical XPaths to segment pages. While the static text identification algorithm in this section is similar to the one presented in [7], there are some basic differences between the two works. The primary goal in [7] is to detect static text at the coarsest-possible granularity (e.g., navigation subpages) so that it can be eliminated from further processing like indexing. In contrast, our use of static text to segment Web pages is novel, and requires us to also detect fine-grained static content like "Categories:" in Example 3.1.

4.3 Training CRF Models

Our goal in the training phase is to learn CRF model M. We segment the labeled pages in the Web sites $\mathcal{W}_t$, and use the labeled segments as our training data. Each segment is treated as a separate sequence of words $\mathbf{w}^i$, and each word is assigned its node's attribute label to yield the label sequence $\mathbf{l}^i$. In our experiments in Section 5, we found that using nodes (instead of segments) as training instances to learn the CRF Model M also yielded good results. In our CRF models, we mainly use features based on node content since these are robust across sites. Example features include the appearance of a word from the training set lexicon, the occurrence of *regex* patterns like `AllCapsWord` or `5digitNumber`, number of words in the node text, etc. We describe our CRF features in more detail in Section 5.

4.4 Labeling Nodes from Web Site $\hat{W}$

Our goal in the labeling phase is to assign attribute labels to nodes that occur in the pages of a new Web site $\hat{W}$. Algorithm 1 contains the description of our procedure for labeling the pages of $\hat{W}$. The algorithm starts with segmenting the pages and labeling words in the individual segments using the trained CRF model M. For the word sequence in each segment, the label sequence with the highest probability is computed using the Viterbi algorithm as described in [9]. (In our experiments in Section 5, we were able to obtain good results even by labeling words in the individual nodes.) Since all words within a segment node belong to the same attribute, we choose the majority label as the label for the node. This helps to fix some of the wrong word labels at a node level.

Now, although a majority of the nodes will be labeled correctly by M, there may still be a sizeable number of nodes with incorrect labels. For certain attributes like Address, we found the error rates to be as high as 75% in our experiments. Our approach is to

Algorithm 1 Label_Pages($\hat{W}, M$)

```
Input: Web site to be labeled $\hat{W}$, CRF Model $M$;
Output: Labeled page sequences from $\hat{W}$;

Segment page sequences in Web site $\hat{W}$ using static nodes;
Let $S$ denote the set of page segments;
Label static nodes in pages of $\hat{W}$ as Noise;
Use CRF model $M$ to assign attribute labels to words in each segment in $S$;
for each segment node $n$ do
  Set label to majority label for words in $n$;
end for
$S'$ = Select_Segment($S$);
for each segment id $e$ do
  Let $S_e$ denote the segments with id $e$ in $S'$;
  $S'_e$ = Correct_Labels($S_e$);
  for each segment $s \in S'_e$ do
    Let $p$ be the page in $\hat{W}$ that contains $s$;
    for each node $n \in s$ do
      Set the label for node $n$ in $p$ equal to the label for node $n$ is $s$;
    end for
  end for
end for
return $\hat{W}$;
```

exploit attribute uniqueness constraints, proximity relationships among attributes, and the structural similarity of pages to correct erroneous labels. For simplicity, we will assume here that all non-noise attributes have a uniqueness constraint.

Within a page, attribute values are contiguous, and thus do not span segments. As a result, each attribute occurs within a single segment. Further, since pages in $\hat{W}$ have similar structure, each attribute occurs in segments with the same id across the pages. Procedure Select_Segment identifies the single segment id for each attribute, and converts the occurrences of the attribute label outside the segment id to noise. Within segments with a specific id identified for each attribute, there may still be errors involving the attribute label. These are corrected by Procedure Correct_Labels using an edit distance based scheme that exploits page structure similarity while allowing for minor structural variations.

Example 4.1 Consider the pages p_1, p_2 and p_3 shown in Figure 4 from a single Web site. Each page has 3 static nodes with text "Yelp", "Categories:", and "Nearest Transit:", and 3 segments with ids e_1, e_2 and e_3. The nodes for each segment are enclosed within ellipses and are annotated with the labels assigned to them by CRF model M. (Note that the Category attribute is missing from the e_2 segment of page p_2.) From the figure, it follows that the correct values for attribute Address occur in segments with id e_2. However, two Address nodes in the e_2 segment of page p_1 are incorrectly labeled as Noise. Furthermore, the two nodes in the e_3 segment of pages p_2 and p_3 are incorrectly labeled as Address. Procedure Select_Segment identifies e_2 as the single segment id for Address and converts the two Address labels in the e_3 segments to Noise. Procedure Correct_Labels then corrects the two Noise labels in the e_2 segment of page p_1 to Address. □

Algorithm 2 Select_Segment(S)

```
Input: Set of segments $S$;
Output: Segments $S$ with a single segment id selected for each attribute;

for each segment id $e$ do
  $attr(e) = \{A : A$ occurs in more than $\beta \cdot sup(e)$ segments with id $e$ in $S\}$;
end for
for each segment id $e$ do
  $w_e = \sum_f dist(e, f) \cdot |attr(f)|$;
end for
for each (non-noise) attribute $A$ do
  $seg(A)$ = segment id $e$ with minimum weight $w_e$ whose $attr$ set contains $A$;
  for each segment $s$ in $S$ with id $e \neq seg(A)$ do
    Set attribute labels for all nodes in $s$ with label $A$ to Noise;
  end for
end for
return $S$;
```

In the following subsections, we describe the procedures Select_Segment and Correct_Labels in more detail.

4.4.1 Selecting Segments for Attributes

Procedure Select_Segment selects a single segment id for each (non-noise) attribute A, and stores it in $seg(A)$. It starts by computing for every segment id e, the attributes for which e is a candidate, and stores these attributes in $attr(e)$. Here, we exploit the fact that a majority of the labels assigned by CRF model M will be correct. Thus, for e to be a candidate for an attribute A, A must occur frequently enough in segments with id e. For a segment id e, let $sup(e)$ denote the number of segments with id e in S. Then, we include A in $attr(e)$ if A occurs in more than $\beta \cdot sup(e)$ segments with id e, where $\beta \approx 0.5$.

If an attribute A occurs in the $attr$ set of only one segment id e, then the segment id $seg(A)$ containing attribute A is unique and equal to e. However, if A occurs in the $attr$ set of more than one segment id, then there may be multiple candidate segment ids for A, and we want to select one from among them. In order to select the segment id $seg(A)$ for attribute A, we make use of the observation that attributes typically

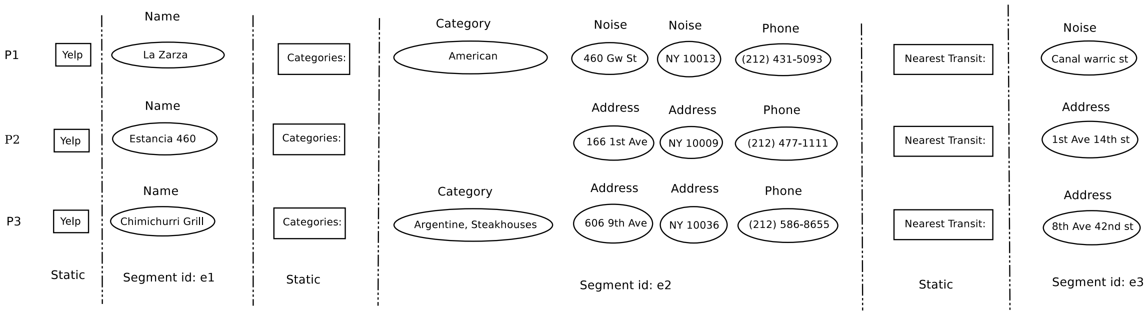


Figure 4: Example restaurant pages with labels.

appear in close proximity to each other within Web pages. For a pair of segment ids e, f, let $dist(e, f)$ denote the average distance between segment pairs with ids e and f over all pages. Here, we define the distance between a pair of segments as the number of intermediate segments between the segments. Alternately, we can also define the distance between a pair of segments to be the number of hops between the start nodes of the segments in the DOM tree of the page.

We can thus state the segment id selection problem as follows.

Segment id selection problem: Given a set of segment ids, a list of attributes $attr(e)$ for each segment id e, and a distance function $dist(e, f)$ between segment id pairs, select a single segment id $seg(A)$ for each attribute A such that (1) $A \in attr(seg(A))$, and (2) $\sum_{A,A'} dist(seg(A), seg(A'))$ is minimum. □

Above, the first condition ensures that $seg(A)$ is a candidate for attribute A while the second ensures that the segment ids for attributes appear close to each other. Unfortunately, selecting segment ids for attributes so that the total distance between all segment id pairs is minimized is an NP-hard problem. So instead, we use a heuristic to select segment ids for attributes.

Our heuristic assigns a weight w_e to each segment id e based on its distance to other segment ids that are candidates for attributes. Intuitively, segment ids that have larger $attr$ sets are more likely to be chosen as the segment id for an attribute. Thus, when computing w_e for a segment id e, we weigh the distance to each segment id by the number of attributes that it is a candidate for. We then select $seg(A)$ to be the segment id that is a candidate for attribute A and whose weight is the minimum. The intuition here is that when there are multiple competing segments that contain an attribute label, we give preference to the segment that is closest to other segments that contain attribute labels.

Finally, for each attribute A, once $seg(A)$ is assigned, we convert all labels A in segments with id not equal to $seg(A)$ to Noise. Thus, at the end of this step, only segments with id $seg(A)$ (at most one per page) contain nodes labeled A.

Example 4.2 Consider the pages p_1, p_2 and p_3 with node labels as shown in Figure 4. The $attr$ sets for segment ids e_1, e_2 and e_3 are as follows: $attr(e_1) = \{\text{Name}\}$, $attr(e_2) = \{\text{Category, Address, Phone}\}$, and $attr(e_3) = \{\text{Address}\}$. Thus, Address occurs in the $attr$ sets of both e_2 and e_3, and we want to select one of them. Note that $w_{e_2} = dist(e_2, e_1) \cdot |attr(e_1)| + dist(e_2, e_3) \cdot |attr(e_3)| = 1 \cdot 1 + 1 \cdot 1 = 2$. Similarly, since $dist(e_3, e_1) = 2$ and $|attr(e_2)| = 3$, we can show that $w_{e_3} = 5$. Thus, Procedure Select_Segment selects e_2 as the segment id for Address, and sets the Address labels in the e_3 segment of pages p_2 and p_3 to Noise. □

4.4.2 Correcting Attribute Labels in Segment

After selecting segments for each attribute, while we expect that the majority of segment nodes will be labeled correctly, some node labels may still be incorrect. In this subsection, we focus on correcting the labels for each attribute A in segments with id $seg(A)$. Our scheme exploits the fact that Web pages within a site have similar structure while being accommodative of small structural variations between pages due to missing attributes.

Now, a straightforward solution here is to number nodes from the start in each segment. Then, we can simply change the attribute label for all nodes in position i (of segments with identical ids) to the majority label in that position. The problem with this simple solution is that due to missing attributes, nodes in the same position i across the segments may contain values belonging to multiple different attributes. Hence, assigning the majority label to these values can cause nodes to be incorrectly labeled. For example, in Figure 4, due to the missing Category attribute in page p_2, our simple solution would incorrectly assign the label Category to the Address node with text "166 1st Ave". Similarly, grouping nodes with identical XPaths, and assigning the majority label to all nodes in a group may not work either. This is because different attributes may have identical XPaths, e.g., if they are elements of a list.

Intuition. A key observation we make here is that even though attributes may appear at variable node positions within a segment, since pages share a common template, the variations across segments with the same id will be minor, and primarily due to (1) missing or additional nodes in certain segments, and (2) incorrectly labeled nodes in some segments. Essentially, the edit distance between segments (restricted to node labels and XPaths) with the same id will in general be small. This allows us to develop an edit distance based algorithm for correcting the label assignments to nodes of segments in S_e with id e.

Let segment $s = n_1 \cdots n_u$ with attribute labels $l_1, \ldots, l_u$ and XPaths $x_1, \ldots, x_u$ for the nodes $n_1, \ldots, n_u$. We adjust the labels in s by computing a minimal sequence of edit operations (on nodes of s) required to ensure that *s matches* every other segment $s' \in S_e$, and then selecting the label for each node based on the majority operation. The edit operations on s that we consider are (1) $del(n_i)$ – delete node n_i from s, (2) $ins(n'_i, l'_i, x'_i)$ – insert a new node n'_i with label l'_i and XPath x'_i into s, and (3) $rep(n_i, l_i, l'_i)$ – replace the label l_i of node n_i in s with l'_i. Informally, we say that segments s and s' match if their label and XPath sequences match. Now, the *ins*/*del* operations align the corresponding node pairs in s and s' - these are essentially node pairs with identical XPaths and belonging to the same attribute. On the other hand, *rep* detects and resolves label conflicts between the corresponding node pairs. Note that we allow *rep* to only replace node labels but not XPaths because labels may have errors that need to be corrected while node XPaths are fixed.

For a segment s, let OS denote the set of minimum edit operation sequences for s to match every $s' \in S_e$ – here OS contains one operation sequence for each s'. Then, for a node n_i in s, if the majority operation in OS is $rep(n_i, l_i, l'_i)$, then this means that a majority of the nodes corresponding to n_i in the other segments in S_e have label l'_i. Since most of these node labels are correct, node n_i's label needs to be changed from l_i to l'_i. Similarly, if a majority of sequences in OS contain no operation involving n_i, then it implies that the labels of most corresponding nodes in other segments agree with n_i's label l_i, and so it must be correct and should be left as is. Note that operation $del(n_i)$ basically indicates that the corresponding node for n_i is absent in s', and hence the attribute for n_i is missing from s'. Furthermore, it is easy to see that there cannot be an *ins* operation in OS involving a node n_i in s. So we can safely ignore *del* and *ins* operations when computing the majority operation for a node n_i.

Min Edit Operations. Let $s = n_1 \cdots n_u$ (with labels $l_1, \ldots, l_u$ and XPaths $x_1, \ldots, x_u$) and $s' = n'_1 \cdots n'_v$ (with labels $l'_1, \ldots, l'_v$ and XPaths $x'_1, \ldots, x'_v$) be segments in S_e. Segments s and s' are said to match if $u = v$, and for all $1 \le i \le u$, $l_i = l'_i$ and $x_i = x'_i$. Now, suppose that $s = n_1 \cdot t$ and $s' = n'_1 \cdot t'$. Then, the minimum number of edit operations $min\_op\_num(s, s')$ required so that s matches s' can be computed recursively, and is the minimum of the following 3 quantities:

1. $min\_op\_num(t, t') + c(n_1, n'_1)$, where $c(n_1, n'_1)$ is equal to

- 0 if $l_1 = l'_1$ and $x_1 = x'_1$,
- 1 if $l_1 \neq l'_1$ and $x_1 = x'_1$, and
- ∞ if $x_1 \neq x'_1$.

2. $min\_op\_num(s, t') + 1$.
3. $min\_op\_num(t, s') + 1$.

The first choice above tries to match n_1 with n'_1 and t with t'. If l_1 and l'_1 are already equal and so are x_1 and x'_1, then no operations are needed to match n_1 and n'_1. However, if $l_1 \neq l'_1$, then a single operation is needed to replace l_1 with l'_1, and clearly, if $x_1 \neq x'_1$, then n_1 cannot be matched with n'_1. This is because n_1 and n'_1 cannot belong to the same attribute if their XPaths are different. The second choice above corresponds to inserting n'_1 with label l'_1 and XPath x'_1 into s, and the third choice corresponds to deleting n_1 from s.

Thus, the minimum sequence of edit operations $min\_op\_seq(s, s')$ needed to match s with s' can also be computed recursively (in parallel with $min\_op\_num(s, s')$ above), and essentially depends on which of the above three choices leads to the minimum value for $\min\_op\_num(s, s')$. If choice 1 has the minimum value, then $min\_op\_seq(s, s')$ is equal to $o \cdot min\_op\_seq(t, t')$ where operation o is null if $l_1 = l'_1$ and $x_1 = x'_1$, and $o = rep(n_1, l_1, l'_1)$ if $l_1 \neq l'_1$ and $x_1 = x'_1$. Else if choice 2 has the lowest value, then $min\_op\_seq(s, s') = ins(n'_1, l'_1, x'_1) \cdot min\_op\_seq(s, t')$, and finally if choice 3 has the smallest value, then $min\_op\_seq(s, s') = del(n_1) \cdot min\_op\_seq(t, s')$.

Procedure Description. For each segment s with id e, Procedure Correct_Labels first computes the minimum sequence of edit operations between s and every other segment $s' \in S_e$, and stores these in OS. For each node n (with current label $lbl(n)$) in s, it computes the new label based on the majority operation as follows. It first calculates $count(lbl(n))$, the number of operation sequences in OS that contain zero *del* or *rep* edit operations involving node n. This is essentially the number of operation sequences in which node n's label is left unchanged. For a label $l \neq lbl(n)$, $count(l)$ stores the number of operation sequences in which node n's label is replaced with label l. Then, the new label for node n in s is set to the attribute label l for which $count(l)$ is maximum. (Note that we break ties in favor of $lbl(n)$ whenever possible, and arbitrarily otherwise.) We also set the support $sup(n)$ of node n to be equal to this maximum value of $count(l)$.

Finally, for each attribute A whose label appears in segment s, from among maximal contiguous sequences of nodes with label A, we select the sequence containing the node n with maximum support $sup(n)$. We

Algorithm 3 Correct_Labels(S_e)

```
Input: Set of segments $S_e$ with id $e$;
Output: Segments $S_e$ with corrected labels;

for each segment $s \in S_e$ do
  $OS = \{min\_op\_seq(s, s') : s' \in S_e\}$;
  for each node $n$ in segment $s$ do
    $count(lbl(n)) = |\{os : os \in OS \wedge$ $os$ does not contain $del$ or $rep$ operation involving $n\}|$;
    for each attribute label $l \neq lbl(n)$ do
      $count(l) = |\{os : os \in OS \wedge$ $os$ contains edit operation $rep(n, lbl(n), l)\}|$;
    end for
    Set $lbl(n) = \arg\max_l count(l)$;
    Set $sup(n) = \max_l count(l)$;
  end for
  for each non-noise attribute $A$ that appears in $s$ do
    Select node $n$ in $s$ with maximum support $sup(n)$ from among nodes with label $A$;
    for each node $n' \neq n$ in $s$ with label $A$ do
      if $n$ and $n'$ are separated by a node with label different from $A$ then
        Set label for $n'$ to be equal to Noise;
      end if
    end for
  end for
end for
return $S_e$;
```

then update the labels of nodes with label A that lie outside this sequence to Noise. (Another option would be to output the maximal contiguous sequence of nodes that are all labeled A and whose average support is maximum.)

Example 4.3 Consider the 3 pages in Figure 4. Let s_1, s_2 and s_3 be the segments with id e_2 in the pages p_1, p_2 and p_3, respectively (for simplicity, we ignore the Phone attribute in all three segments).

- $s_1 = n_{11} \cdot n_{12} \cdot n_{13}$ with node text “American”, “460 Gw St”, and “NY 10013”;
- $s_2 = n_{21} \cdot n_{22}$ with node text “166 1st Ave”, and “NY 10009”;
- $s_3 = n_{31} \cdot n_{32} \cdot n_{33}$ with node text “Argentine, Steakhouses”, “606 9th Ave”, and “NY 10036”;

Now consider segment s_1. We have

- $min\_op\_seq(s_1, s_1) = \epsilon$, the empty sequence,
- $min\_op\_seq(s_1, s_2) = del(n_{11}) \cdot rep(n_{12}, \text{Noise}, \text{Address}) \cdot rep(n_{13}, \text{Noise}, \text{Address})$,
- $min\_op\_seq(s_1, s_3) = rep(n_{12}, \text{Noise}, \text{Address}) \cdot rep(n_{13}, \text{Noise}, \text{Address})$.

Thus, for node n_{11}, since count(Category) = 2 the label stays as Category, and for nodes n_{12} and n_{13}, the labels are modified to Address since count(Address) = 2 for these nodes. □

5 Experimental Evaluation

We compare the accuracy of our attribute extraction techniques that exploit site knowledge with baseline linear-chain and hierarchical CRF methods on real-life *restaurant* Web pages.

Datasets: We consider restaurant pages from the following 5 real-world Web sites: `www.citysearch.com`, `www.frommers.com`, `www.nymag.com`, `www.superpages.com`, and `www.yelp.com`. The dataset consists of a total of 455 pages. The number of pages from each of the above sites is 92, 71, 95, 100, and 97, respectively. In each page, we assign labels to the following 5 attributes: **Name** (N), **Address** (A), **Phone number** (P), **Hours of operation** (H), and **Description** (D). Attribute labels are obtained by first manually annotating one sample page from each site, and then using wrappers to label the remaining pages in each site. All words that do not belong to any of the 5 above-mentioned attributes are labeled as **Noise**. In our experiments, we use 50 pages from one site as test data, and all the pages from the remaining 4 sites as training data. This dataset is representative of the Web scale extraction task because of the following:

1. Restaurant Web sites are representative of the general class of script-generated sites belonging to Shopping, Blog, Video, and other verticals – these contain a lot of static text and attribute values which have strong “types” (price, date, count, etc.). The former helps in page segmentation and the latter helps in building good Node CRFs. Hence the techniques and experimental results described in the paper should be applicable across a broad range of verticals.
2. The order of attributes in the 5 sites is: NAPHD, NHAPD, NAPDH, NPAH, and NAPH. Thus, there is considerable variation in the attribute ordering across sites.
3. In a real-world setting, our proposed technique will be used to automatically annotate a small number (≈ 50) of pages of a new site. These annotated pages will then be used to generate a wrapper which can be used to extract efficiently from the remaining (of the order of thousands of) pages.

Extraction Methods: We compare the performance of our techniques with that of a baseline linear-chain CRF and a hierarchical CRF. To measure the incremental improvement in accuracy that we get due to each of our extraction steps, we also consider successive schemes derived by adding pre- and post-processing steps to the baseline.

• *Baseline (***CRF***):* Here, the linear-chain CRF model is built on the word sequence formed from all the leaf nodes in the DOM tree of the complete Web page.

- *Node CRF (***NODE***):* We train the linear-chain CRF on word sequences for individual nodes in page segments rather than the word sequence for the entire page. (Although we could also train on word sequences for individual segments, we found that training at a node granularity resulted in better performance.) We include all nodes belonging to non-noise attributes in the training set. In addition we randomly choose a fraction of nodes labeled Noise. We found that including all the noise nodes during training results in the CRF labeling most of the nodes in the test pages as Noise. In our experiments below, the fraction of noise nodes used for training is 10%. We did not include static nodes as part of the training or test data.
- *Node CRF+Segment Selection (***SS***):* In addition to training on word sequences for nodes, this scheme uses proximity constraints to identify the correct segment for each attribute.
- *Node CRF+Segment Selection+Edit Distance (***ED***):* This is our complete scheme described in Section 4 that also performs the final step in which edit distance is used to correct the labels on wrongly labeled nodes.

CRF Features: In our CRFs, we only use features based on the content of HTML elements in Web pages. We do not use structure or presentation information like font, color, etc. as CRF features since these are not robust across sites. We use binary features that fall into three categories:

- *Lexicon features:* Each word from the training set constitutes a feature. We build a lexicon over the words appearing in the training Web pages. If a word in a page is present in the lexicon, then the corresponding feature is set to 1.
- *Regex features:* Occurrences of certain patterns in the content is captured by regex features. Some examples of regex features are `isAllCapsWord` which fires if all letters in a word are capitalized, `3digitNumber` which indicates the presence of at least one 3-digit number, and `dashBetweenDigits` indicating the presence of a '-' in between numbers. The total number of regex features is 11[3].
- *Node-level features:* These features capture length information for a node, and overlap of the node text with the page title. Some examples include `propOfTitleCase` which indicates what fraction of the node text contains words that begin with a capital letter, and `overlapWithTitle` which indicates the extent of overlap of given text with the `<title>` tag of the Web page containing that text. Real-valued features are converted to multiple binary features through binning. The total number of these features is 7[4].

Note that the node-level features described above are the same for all the words in a node. We tried various combinations of the regularization parameter σ and normalization type (L1 or L2). We obtained the best performance with L1 normalization and σ=5. The CRF implementation we use is the CRFsuite (`http://www.chokkan.org/software/crfsuite/`).

Evaluation Metrics: We use the standard precision, recall and F1 measures to evaluate the methods. For each scheme, we average the above measures across 5 experiments - each experiment treats a single site as the test site, and uses the pages from the remaining 4 sites as training data.

Experimental Results: Table 1 depicts the precision, recall and F1 numbers for the various schemes. Observe that the baseline scheme **CRF** has the worst performance. The reason for this is that the attribute orders are different across sites, and the training pages contain lots of noise which biases the CRF to label most nodes as noise.

Below, we analyze the performance of each step of our extraction method.

- NODE: As can be seen from Table 1, the NODE scheme outperforms the baseline CRF scheme – this is because of shorter sequences and less noise in the training data. Furthermore, training at the granularity of a node ensures that the CRF does not learn the inter-attribute dependencies in the training data that do not hold in the test data. Observe that the recall of NODE is high for most of the attributes but the precision is moderate to low across attributes. This is because we only use content features, and node labeling is done without taking into account constraints like attribute uniqueness. For example, many restaurant pages contain multiple instances of addresses of which only one is the restaurant address. The NODE scheme labels all instances as addresses leading to reduced precision. Also, note that the precision is higher for single-node attributes like Name, Phone, and Hours compared to multi-node attributes like Address and Description because in the former case the training is on entire attributes as opposed to parts of attributes in the latter case.
- SS: Performing segment selection boosts the precision of all attributes. (On an average, each page is split into 40 segments.) The minimum and maximum increase in precision are 28% (for Name) and 150% (for Address). This demonstrates the effectiveness of uniqueness and proximity constraints to resolve multiple occurrences of an attribute in a page. Note that for constraint satisfaction to be effective the recall of the underlying CRF needs to be high since high recall ensures that the correct segment is selected. This also explains why SS performs poorly on Description since the recall of the CRF on Description is low (40%) and this leads to the wrong segment being selected.

[3] The complete list of regex features is as follows: `isAllCapsWord`, `hasTwoContinuousCaps`, `isDay`, `1-2digitNumber`, `3digitNumber`, `4digitNumber`, `5digitNumber`, `>5digitNumber`, `dashBetweenDigits`, `isAlpha`, and `isNumber`.

[4] The complete list is as follows: `noOfWords>20`, `noOfWords>50`, `noOfWords>100`, `propOfTitleCase<0.2`, `propOfTitleCase>0.8`, `overlapWithPageTitle`, and `prefixOverlapWithPageTitle`.

- ED: Our ED scheme has the best overall performance. It improves the recall of Hours by 11% by fixing incorrectly labeled Hour nodes. The reason for this is as follows: Hours is specified in different formats in different pages of the same site - some with AM/PM, some with only days, etc. When the CRF wrongly labels an hours node as NOISE in a page, ED corrects the label to Hours since Hours is the majority label of the corresponding nodes in other pages. Hence ED increases recall by exploiting cross-page regularity.

We next present the execution times of different components on a 2.33GHz Intel Xeon machine. The static node identification took 4.94 sec/site using 20 pages. Page segmentation time was 0.157 sec/page. Node CRF Training took 19.8 sec to learn a model on a dataset of 360 pages while Node CRF Labeling took 0.02 sec/page. The postprocessing steps of Segment Selection and Label Correction took 2.09 sec and 17.1 sec each on a test set of 50 pages. From the above, it can be seen that the attribute labeling (Segmentation + Labeling + Postprocessing) time for each new site is around 25 sec, which is quite fast. Also, as mentioned earlier, we automatically label only a few (≈ 50) pages of each new site and use wrappers learned from the labeled pages for large-scale extraction.

Comparison with Hierarchical CRF (HCRF): Our HCRF implementation is along the lines described in [15]. We use the implementation of the vision-based page segmentation (VIPS) technique from `http://www.cs.uiuc.edu/homes/dengcai2/VIPS/VIPS.html`. VIPS provides page layout features such as position, font, color and size, and constructs the *vision-tree* for a Web page. The cliques in the HCRF model graph are the vertices, edges and triangles of the vision tree. We have implemented HCRF on top of the GRMM toolkit `http://mallet.cs.umass.edu`; the inference algorithm used is junction trees.

We use only the element features as defined in [15] and not the block features since all our experiments are on detail pages (and not on list pages). The features used are visual, lexicon, and regex features. We have 5 visual features – row and column positions, area, font size, and weight. The lexical features are based on text and link lexicons. The regex features are: `containsSpecialCharacter`, `initCaps`, `containsTime`, `containsDay`, `n-digit-number` (for n=2, 3, 4, 5), and `isAllCapsWord`. Another additional feature used is `overlapWithPageTitle`. As before, real-valued features are converted to multiple binary features through binning. The total number of features in our implementation is 1826.

The HCRF implementation using GRMM requires a lot of memory (exceeding 3GB) and also takes a long time to train. Hence we use only 25 pages per site in the training sets in HCRF experiments. The test set is the same as that used in the rest of the experiments. The overall F1 score for HCRF is 0.262 while that of our proposed technique, when trained on the same training set of 25 pages per site, is 0.5274. This supports our argument that attribute ordering across sites is not consistent and full-page CRFs are not likely to perform well in cross-site labeling tasks.

	Label	**CRF**	**NODE**	**SS**	**ED**
Prec.	Name	0.39	0.78	1	1
	Phone	0.02	0.59	1	1
	Address	0.01	0.24	0.8	0.81
	Hours	0.22	0.67	1	1
	Desc	0.22	0.37	0.42	0.42
	Overall	**0.17**	**0.53**	**0.84**	**0.84**
Recall	Name	0.34	0.98	0.98	1
	Phone	0.2	0.99	0.98	0.99
	Address	0.16	0.88	0.82	0.83
	Hours	0.36	0.89	0.89	1
	Desc	0	0.67	0.25	0.25
	Overall	**0.21**	**0.88**	**0.78**	**0.81**
F1	Name	0.36	0.85	0.99	1
	Phone	0.04	0.69	0.99	0.99
	Address	0.02	0.33	0.8	0.81
	Hours	0.26	0.74	0.94	1
	Desc	0.02	0.43	0.32	0.32
	Overall	**0.14**	**0.61**	**0.81**	**0.82**

Table 1: Comparison of extraction methods.

6 Concluding Remarks

In this paper, we have proposed a new framework for high-precision extraction using site knowledge to improve the labelings of any underlying classifier. Site knowledge is used in an unsupervised manner in two different ways: (1) Analyzing multiple pages from a site to discover static content for segmenting pages from the site, and (2) Enforcing segment-level uniqueness, proximity, and structure similarity constraints on nodes labeled by the classifier. The primary requirement on node classifiers is that they should have high recall which is satisfied by our node CRFs. Thus the framework combines the strengths of wrapper-like techniques which leverage site-specific cues and site-independent machine-learning techniques like CRFs that exploit content features. We have demonstrated the effectiveness of our approach on real-life restaurant Web pages with average accuracy gains exceeding a factor of 5.

In our work presented in this paper, site knowledge is used in the pre- and post-processing steps. It will be interesting to develop a unified framework which integrates constraint satisfaction more tightly with the underlying classification technique. We intend to explore this as future work.

References

[1] E. Agichtein and V. Ganti. Mining reference tables for automatic text segmentation. In *ACM SIGKDD*, 2004.

[2] A. Arasu and H. Garcia-Molina. Extracting structured data from web pages. In *ACM SIGMOD*, 2003.

[3] K. Bellare and A. McCallum. Learning extractors from unlabeled text using relevant databases. In *AAAI Workshop*, 2007.

[4] V. Borkar, K. Deshmukh, and S. Sarawagi. Automatic segmentation of text into structured records. In *ACM SIGMOD*, 2001.

[5] S. Chuang, K. C. Chang, and C. Zhai. Context-aware wrapping: Synchronized data extraction. In *VLDB*, 2007.

[6] V. Crescenzi, G. Mecca, and P. Merialdo. Roadrunner: Towards automatic data extraction from large web sites. In *VLDB*, 2001.

[7] D. Gibson, K. Punera, and A. Tomkins. The volume and evolution of web page templates. In *WWW*, 2005.

[8] N. Kushmerick, D. S. Weld, and R. Doorenbos. Wrapper induction for information extraction. In *IJCAI*, 1997.

[9] J. Lafferty, A. McCallum, and F. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *ICML*, 2001.

[10] I. Muslea, S. Minton, and C. Knoblock. Hierarchical wrapper induction for semistructured information sources. *Autonomous Agents and Multi-Agent Systems*, 1(2), 2001.

[11] D. Roth and W. Yih. Integer linear programming inference for conditional random fields. In *ICML*, 2005.

[12] J.-M. Yang, R. Cai, Y. Wang, J. Zhu, L. Zhang, and W.-Y. Ma. Incorporating site-level knowledge to extract structured data from web forums. In *WWW*, 2009.

[13] Y. Zhai and B. Liu. Web data extraction based on partial tree assignment. In *WWW*, 2005.

[14] J. Zhu, Z. Nie, J. Wen, B. Zhang, and W. Ma. 2D conditional random fields for web information extraction. In *ICML*, 2005.

[15] J. Zhu, Z. Nie, J. Wen, B. Zhang, and W. Ma. Simultaneous record detection and attribute labeling in web data extraction. In *ACM SIGKDD*, 2006.