

On Precision and Recall of Multi-Attribute Data Extraction from Semistructured Sources

Guizhen Yang
Department of Computer Science and Engineering
University at Buffalo
Buffalo, NY 14260-2000, USA
gzyang@cse.buffalo.edu

Saikat Mukherjee I. V. Ramakrishnan
Department of Computer Science
Stony Brook University
Stony Brook, NY 11794-4400, USA
{saikat,ram}@cs.stonybrook.edu

Abstract

Machine learning techniques for data extraction from semistructured sources exhibit different precision and recall characteristics. However to date the formal relationship between learning algorithms and their impact on these two metrics remains unexplored. This paper proposes a formalization of precision and recall of extraction and investigates the complexity-theoretic aspects of learning algorithms for multi-attribute data extraction based on this formalism. We show that there is a tradeoff between precision/recall of extraction and computational efficiency and present experimental results to demonstrate the practical utility of these concepts in designing scalable data extraction algorithms for improving recall without compromising on precision.

1 Introduction

Numerous Web data sources, as illustrated in Figure 1, present organized information about entities and their attributes. For instance, each veterinarian service provider in Figure 1 corresponds to an entity whose attributes include the name of the service provider (e.g., “ABC Animal Hospital”), the resident veterinarian who provides the service (e.g., “John, DVM”), the address and phone number of the service provider. Usually Web pages containing this kind of information exhibit a *consistent* presentation style.

A common approach for extracting data from Web sources is to build wrappers [1, 10, 20, 17, 13, 4]. Among these assorted approaches learning-based extraction techniques [13, 16, 4, 5] are becoming important since they exhibit a high degree of automation and scalability. Extraction based on learning techniques is a two step process. In the first step, called *labeling*, examples of relevant data to be extracted from the source are supplied. This labeling step can be either manual or completely automated. In the latter case



Figure 1. A List of Services Web Page

an *ontology*, encoding knowledge of the data domain, applies an attribute *identifier function* (e.g., pattern matching, keyword-based search) to locate an attribute’s occurrences in the data source. In the second step the labeled examples are used to automatically learn an extraction expression for pulling out all the relevant data in the source.

1.1 Precision and Recall in Learning-Based Data Extraction

Learning methods synthesize extraction expressions by a process of generalization from the labeled examples and so the set of attribute occurrences that they pull out will be a superset of the labeled examples. Hence these methods are said to increase *recall*¹. But a problem here is that the extraction expression may be too general and pull out incorrect attribute occurrences also, thereby losing *precision*².

For illustration let us consider the following example: Suppose “ABC Animal Hospital” and “XYZ Animal Hospital” are supplied as two examples of the *HospitalName*

¹Recall = $\frac{ce}{ce+te} \times 100\%$, where *ce* is the number of entities extracted correctly and *te* is the number of true entities not extracted.

²Precision = $\frac{ce}{ce+fe} \times 100\%$, where *ce* is the number of entities extracted correctly and *fe* is the number of false entities extracted.

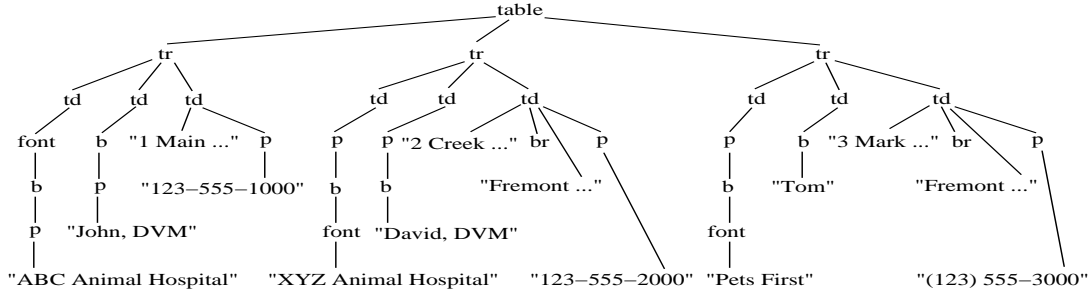


Figure 2. DOM Fragment of Figure 1

attribute. In the DOM tree (see Figure 2) corresponding to the Web page in Figure 1, the paths leading to the leaf nodes containing these text strings are $\alpha \cdot \text{table} \cdot \text{tr} \cdot \text{td} \cdot \text{font} \cdot \text{b} \cdot \text{p}$ and $\alpha \cdot \text{table} \cdot \text{tr} \cdot \text{td} \cdot \text{p} \cdot \text{b} \cdot \text{font}$, respectively, where α represents the path string from the root of the DOM tree to the *table* tag. We can learn an extraction expression, specifically the regular expression $E_1 = \alpha \cdot \text{table} \cdot \text{tr} \cdot \text{td} \cdot \text{font}^* \cdot \text{p}^* \cdot \text{b} \cdot \text{p}^* \cdot \text{font}^*$, from these two paths. If we execute E_1 as a path query on the DOM tree, it will return the text string “Pets First”. However, notice that the language³ of E_1 also includes the path string, $\alpha \cdot \text{table} \cdot \text{tr} \cdot \text{td} \cdot \text{p} \cdot \text{b}$, which terminates on the leaf node “David, DVM”. But this is an occurrence of a different attribute, *DoctorName*, in the schema. By extracting such false positives an extraction expression’s increase in recall can be compromised by a reduction in precision.

Observe that different extraction expressions can be learned from the same set of examples. For instance, we can learn another expression, $E_2 = \alpha \cdot \text{table} \cdot \text{tr} \cdot \text{td} \cdot \text{p}^* \cdot \text{b}^* \cdot \text{font} \cdot \text{b}^* \cdot \text{p}^*$, from $\alpha \cdot \text{table} \cdot \text{tr} \cdot \text{td} \cdot \text{font} \cdot \text{b} \cdot \text{p}$ and $\alpha \cdot \text{table} \cdot \text{tr} \cdot \text{td} \cdot \text{p} \cdot \text{b} \cdot \text{font}$. The path string $\alpha \cdot \text{table} \cdot \text{tr} \cdot \text{td} \cdot \text{p} \cdot \text{b}$ is excluded from E_2 ’s language. In fact none of the path strings leading to the *DoctorName* attribute data will be recognized by E_2 . So E_2 retains more precision than E_1 .

The above examples raises several interesting algorithmic questions: *Can we design learning algorithms without losing precision? What is the computational difficulty of such algorithms? Are there tradeoffs between precision/recall and computational efficiency?*

1.2 Our Approach

To the best of our knowledge a study of the complexity-theoretic aspects of precision and recall of extraction and their impact on the efficiency of learning extraction expressions has not been explored in the literature. In this paper we undertake such a study within the context of multi-attribute data extraction. We introduce a novel formalization of precision and recall of extraction that will serve

³The language of a regular expression is the set of strings recognized by this expression.

as the formal model for analyzing an algorithm’s precision/recall characteristics.

In our model, extraction expressions are regular expressions built using the concatenation (“.”) and the Kleene closure (“*”) operators only. We call them *Path Abstraction Expressions* (PAEs). To handle multi-attribute entities, we associate one PAE with each attribute. When a PAE is evaluated on the DOM tree of a Web page, the leaf nodes whose path strings match this PAE will be extracted.

To learn a PAE for an attribute we generalize from both its positive and negative examples. The labeling step provides positive examples for each attribute. The set of negative examples for an attribute comprises the positive examples of all the other attributes. Based on the extent to which false positives can be excluded from a PAE’s language, we ascribe different *quality* measures to the PAEs learned (see Section 2). Better quality PAEs eliminate more false positives and will therefore possess higher precision.

Our theoretical analysis shows, however, that learning good quality PAEs can be computationally intractable. Consequently, PAEs learned by practical heuristics often give rise to ambiguity when they are used to extract multi-attribute data. To this end, we model ambiguity resolution as an algorithmic problem over bipartite graphs and develop efficient algorithms for learning PAEs and resolving ambiguity. By combining domain knowledge (possibly encoded in an ontology) with these algorithms we can improve recall without much loss in precision. The experimental results, obtained from running our algorithms over 194 different Web pages listing veterinarian service providers and product descriptions, show that the overall recall achieved ranges from 58% to 100% with almost no loss in precision.

The rest of the paper is organized as follows. Section 2 formalizes different notions of PAEs with different quality measures. Complexity results on learning these classes of PAEs are presented in Section 3. Heuristics for learning PAEs and the algorithm for resolving ambiguity are also presented in Section 3. Experimental results appears in Section 4 and related work in Section 5. Finally, Section 6 concludes this paper with future work.

2 Problem Formalization

A *path abstraction expression* (abbr. PAE) is like a regular expression but with two restrictions: (1) it is free of the union operator (“|”); and (2) the Kleene closure operator (“*”) can only apply to single symbols. The set of strings recognized by a PAE, E , is denoted as $\mathcal{L}(E)$.

By disallowing the union operator (“|”) in the syntax of PAEs, we can force *generalization* in the learning algorithms. Otherwise, one can just compose a regular expression by concatenating all the input strings using the union operator. Such techniques do not really capture any regularity in the paths within a DOM tree.

Although we require that the Kleene closure operator (“*”) be applied to single symbols only, this does not really impose any extra technical difficulty. We make this simplification just for the Web domain, since in reality it is very rare that a consecutive sequence of tags would repeat itself in the root-to-leaf paths of a DOM tree.

Definition 1 (Nonredundancy) Let S be a set of strings and E a PAE such that $\mathcal{L}(E) \supseteq S$. We say that E is *nonredundant* w.r.t. S , iff we cannot perform either of the following operations on E to obtain a new PAE E' such that $\mathcal{L}(E') \supseteq S$: (1) Remove a symbol together with its Kleene closure operator; (2) Remove from a symbol the Kleene closure operator.

When $\mathcal{L}(E) \supseteq S$, we will normally say that E *covers* or *generalizes* S . However, if E overgeneralizes then it will produce more false positives. Note that if we can perform either of the two operations in Definition 1 on E and obtain a new E' , then $\mathcal{L}(E') \subset \mathcal{L}(E)$. Thus E' should produce less false positives in general. Therefore we require that one should learn a nonredundant PAE from examples.

Notice that nonredundant PAEs do not say anything about negative examples. To deal with negative examples we introduce the following definition.

Definition 2 (Consistency) Let E be a PAE, POS and NEG two sets of strings. We say that E is *consistent* w.r.t. $\langle POS, NEG \rangle$, iff $\mathcal{L}(E) \supseteq POS$ and $\mathcal{L}(E) \cap NEG = \emptyset$.

In the above Definition 2, the strings in POS serve as positive examples while those in NEG serve as negative examples. Intuitively, a consistent PAE generalizes all positive examples but excludes all negative examples. Therefore, extraction using consistent PAEs will have higher precision than nonredundant PAEs.

In practice we often need to extract multiple attributes of an entity. Our goal is to learn a list of PAEs, one for *each* attribute. Note that for any given attribute, the positive examples from other attributes will serve as negative examples for this attribute. If for *each* attribute, we can

learn a consistent PAE that covers the positive examples of this attribute but excludes all the positive examples of other attributes combined, then we say that this list of PAEs is unambiguous w.r.t. the given list of sets of examples.

Definition 3 (Unambiguity) Given a list of sets of strings, (S_1, \dots, S_n) , and a list of PAEs, (E_1, \dots, E_n) , we say that (E_1, \dots, E_n) is *unambiguous* w.r.t. (S_1, \dots, S_n) , iff E_i is consistent w.r.t. $\langle S_i, \bigcup_{j \neq i} S_j \rangle$ for $1 \leq i \leq n$.

However, even when a list of PAEs is unambiguous w.r.t. examples, the languages recognized by these PAEs may still overlap. Therefore, a more desirable, better quality is that these languages are pairwise disjoint. If so, then we say the list of PAEs is *inherently unambiguous*. Clearly, inherently unambiguous PAEs are able to retain more precision than those that are only unambiguous w.r.t. the given examples.

Definition 4 (Inherent Unambiguity) Let (S_1, \dots, S_n) be a list of sets of strings and (E_1, \dots, E_n) a list of PAEs. We say that (E_1, \dots, E_n) is *inherently unambiguous* w.r.t. (S_1, \dots, S_n) , iff $\mathcal{L}(E_i) \supseteq S_i$ for $1 \leq i \leq n$ and $\mathcal{L}(E_i) \cap \mathcal{L}(E_j) = \emptyset$ for $1 \leq i \leq n$, $1 \leq j \leq n$, and $i \neq j$.

We formally state our learning problems as follows.

Problem 1 (Consistent PAE) Given two sets of strings POS and NEG , is there a PAE that is consistent w.r.t. $\langle POS, NEG \rangle$?

Problem 2 (Unambiguous PAEs) Given a list of sets of strings, (S_1, \dots, S_n) , is there a list of PAEs, (E_1, \dots, E_n) , that is unambiguous w.r.t. (S_1, \dots, S_n) ?

Problem 3 (Inherently Unambiguous PAEs) Given a list of sets of strings, (S_1, \dots, S_n) , is there a list of PAEs, (E_1, \dots, E_n) , that is inherently unambiguous w.r.t. (S_1, \dots, S_n) ?

3 Computational Aspects of Learning PAEs

3.1 Learning Nonredundant PAEs

Algorithm *LearnPAE* takes as input a set of strings (S^+) and returns as output a nonredundant PAE (E) which generalizes this set. In Line 2, variable E is initialized with the first positive example. In Line 3, the shortest common supersequence (SCS)⁴ of the string stored in E and the next positive example is computed and then assigned to E . When the loop in Line 3 terminates, E stores a common supersequence for all the strings in S^+ . In Line 4, the string stored in E is generalized to a PAE that covers S^+ by adding * on all the symbols in E .

⁴Supersequences of a string can be obtained by inserting zero or more symbols into this string.

Algorithm LearnPAE(S^+)

input

S^+ : a nonempty set of strings

output

E : a nonredundant PAE which covers S^+

begin

1. Let α_i ($1 \leq i \leq n$) be a string in S^+ .
 2. $E = \alpha_1$
 3. **for** $2 \leq i \leq n$ **do** $E = SCS(E, \alpha_i)$ **endfor**
 4. Put a * on all the symbols of E .
 5. $E = \text{MakeNonredundant}(E, S^+)$
 6. **return** E
- end**

Algorithm *MakeNonredundant* takes as input a PAE, E , and a set, S^+ , of positive examples that E covers. Upon termination it makes E nonredundant w.r.t. S^+ . First, for every symbol with a * in E , it is checked whether dropping the symbol together with * results in a new PAE that still covers S^+ . If so, then the symbol together with * is dropped from E (Lines 4–7). If not, then it is checked whether dropping only * from this symbol produces a new PAE that covers S^+ . If it does, then the * is dropped from the symbol (Lines 9–10).

Algorithm MakeNonredundant(E, S^+)

input

E : a PAE which covers S^+

S^+ : a nonempty set of strings

output

Q : a nonredundant PAE which covers S^+

begin

1. n = the number of symbols in E excluding *
 2. Let x_i ($1 \leq i \leq n$) be the i -th symbol in E .
 3. **for** $1 \leq i \leq n$ **do**
 4. **if** a * is attached to x_i **then**
 5. $R = \text{drop } x_i \text{ together with } * \text{ from } E$
 6. **if** R covers S^+ **then**
 7. $E = R$
 8. **else**
 9. $R = \text{drop } * \text{ that is attached to } x_i \text{ from } E$
 10. **if** R covers S^+ **then** $E = R$ **endif**
 11. **endif**
 12. **endif**
 13. **endfor**
 14. $Q = E$
 15. **return** Q
- end**

We can show that the PAE, Q , returned by algorithm *MakeNonredundant* is nonredundant w.r.t. S^+ . Moreover, the complexity of algorithm *LearnPAE* is polynomial time.

3.2 Learning Consistent and Unambiguous PAEs

Recall that a consistent PAE covers all positive examples but excludes all negative examples. Since algorithm *LearnPAE* learns from positive examples only, the PAE that it produces may not be consistent. It turns out that the complexity of learning PAEs runs up very quickly once negative examples must be taken into account.

Theorem 1 *The consistent PAE problem is NP-complete.*

Therefore, one has to resort to heuristics for learning consistent PAEs. A simple heuristic is to first find a *distinguishing subsequence*⁵ of symbols which is present in all positive examples but not in any negative example. If such a distinguishing subsequence exists, then a consistent PAE can be easily constructed. Here we omit details of this heuristic due to want of space.

Our goal is to extract data values for multiple attributes associated with a concept. This requires learning a list of PAEs, one per attribute. The positive and negative examples used for learning a list of PAEs are obtained in the same way as for learning consistent PAEs. To extract data values from a source with high precision and recall, it is desirable that this list of PAEs be unambiguous w.r.t. examples. However, this different learning problem also turns out to be intractable in general.

Theorem 2 *The unambiguous PAEs problem is NP-complete.*

From a computational viewpoint learning a list of PAEs that is unambiguous w.r.t. examples amounts to requiring that each PAE in this list be consistent. Therefore in practice we can repeatedly applying heuristics for learning consistent PAEs to learn an unambiguous list of PAEs.

Finally, for an inherently unambiguous list of PAEs we require that their languages be pair-wise disjoint. Such a list of PAEs has the best precision and recall properties. Although the computational complexity of this problem remains open at this time, it can be shown that it is decidable.

Theorem 3 *The inherently unambiguous PAEs problem is decidable.*

3.3 Resolving Ambiguity

Because learning an unambiguous list of PAEs is computationally difficult, one has to resort to heuristics. Since heuristics may not guarantee that all the PAEs learned are consistent, ambiguity can occur when these PAEs are used for extracting multi-attribute data. We now describe an algorithm based on bipartite graphs that uses domain knowledge to resolve ambiguity as much as possible thereby improving recall while retaining high precision.

For simplicity we assume that record boundaries are already identified and so PAEs are applied to each entity block in a DOM tree (see Section 4.1 for more details). In addition, for expositional convenience we also assume that the attributes are single-valued. Extending the techniques in this section to multivalued attributes is straightforward.

We say that a PAE *matches* an attribute value whenever it accepts the path string terminating on the leaf node in a

⁵Subsequences of a string can be obtained by deleting zero or more symbols from this string.

DOM tree which is labeled with this value. The ambiguity resolution algorithm takes as input a list of PAEs (E) and a set of data values (D) in an entity block and returns a set of 1–1 associations between attributes and data values. Each data value consists of a text string and the path string in the DOM tree that leads to this text string.

Our ambiguity resolution algorithm works in two steps. First domain knowledge is used to resolve ambiguity. If a data value D_j has been identified by the ontology as the value for an attribute A_i , then the pair (A_i, D_j) is added to the set of associations for that record. The data value and the corresponding PAE are deleted from D and E (note that we assume all attributes are single-valued), respectively.

Algorithm BipartiteResolution(E, D)

```

input
   $E$  : a list of PAEs representing attributes
   $D$  : a set of strings representing data values in an entity block
output
   $A$  : a set of pairs in the form of (attribute,value)
begin
1.  $A = \emptyset$ 
2. Let  $E_i \in E$  ( $1 \leq i \leq n$ ) be the PAE for attribute  $A_i$ .
3. Let  $\alpha_j \in D$  ( $1 \leq j \leq m$ ) represent the data value  $D_j$ .
4. Create a bipartite graph  $G$  between  $E_i$ 's and  $\alpha_j$ 's.
5. do
6.    $M = \emptyset$ 
7.   for  $1 \leq i \leq n$  do
8.     if  $\text{degree}(E_i) = 1$  ( $\text{edge}(E_i, \alpha_k) \in G$  for some  $\alpha_k$ ) then
9.        $X = \{E_j \mid j \neq i, \text{edge}(E_j, \alpha_k) \in G, \text{degree}(E_j) = 1\}$ 
10.      if  $X = \emptyset$  then  $M = M \cup \{E_i\}$  endif
11.    endif
12.  endifor
13.  for each  $E_i \in M$  (there is only one  $\text{edge}(E_i, \alpha_k) \in G$ ) do
14.     $A = A \cup (A_i, D_k)$ 
15.    Remove all edges in  $G$  that are incident on  $\alpha_k$ .
16.  endifor
17. while  $M \neq \emptyset$ 
18. return  $A$ 
end

```

In the second step we derive more 1–1 associations between the remaining unresolved data values and PAEs using the algorithm *BipartiteResolution*. This algorithm first constructs a bipartite graph in which the two disjoint sets of vertices are E and D , respectively, and there is an edge between $E_i \in E$ and $\alpha_j \in D$ if E_i matches α_j (Line 4).

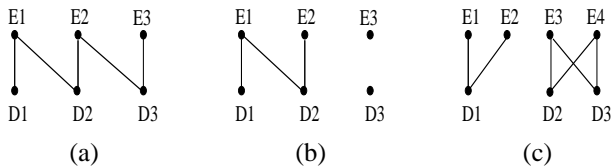


Figure 3. Bipartite Resolution

For instance, given the three records of the DOM tree in Figure 2 with the first two records as examples, suppose for *HospitalName*, *DoctorName*, and *PhoneNumber* the PAEs learned are $E_1 = \text{table} \cdot \text{tr} \cdot \text{td} \cdot p^* \cdot \text{font}^* \cdot b \cdot \text{font}^* \cdot p^*$,

$E_2 = \text{table} \cdot \text{tr} \cdot \text{td} \cdot b^* \cdot p \cdot b^*$, and $E_3 = \text{table} \cdot \text{tr} \cdot \text{td} \cdot p$, respectively. Let D_1 , D_2 , and D_3 represent the data values (including their path strings) “Pets First”, “Tom”, and “(123)555-3000” in the third record of the DOM tree, respectively. Then E_1 matches D_1 and D_2 , E_2 matches D_2 and D_3 , and E_3 matches D_3 only. The bipartite graph corresponding to these matchings is illustrated in Figure 3(a).

If a PAE E_i uniquely matches a data value α_k and no other PAE uniquely matches α_k , then we place a high confidence on this matching and make a 1–1 association between E_i and α_k (Lines 7–12). We also remove all edges that point to α_k (Line 15). For example, in Figure 3(a), since E_3 uniquely matches D_3 the attribute *PhoneNumber* is associated with D_3 and all edges leading into D_3 are deleted. The residual bipartite graph is shown in Figure 3(b).

The computation is repeated until it cannot derive more 1–1 associations. Continuing with the above example in Figure 3(b), E_2 now uniquely matches D_2 . Therefore, the attribute *DoctorName* can be associated with D_2 and all edges leading into D_2 removed. In the final residual graph there is only one edge between E_1 and D_1 . Hence the attribute *HospitalName* can be associated with D_1 .

However, algorithm *BipartiteResolution* may not always derive new associations. For example, given Figure 3(c), the algorithm terminates without any new association.

4 Experimental Results

Here we describe our experimental evaluation of our theories and techniques for extracting multi-attribute data. For simplicity we assume that all records are flat (*i.e.*, without nested structures) and an attribute occurrence does not span multiple leaf nodes in a DOM tree. In Section 6 we discuss relaxing these two assumptions.

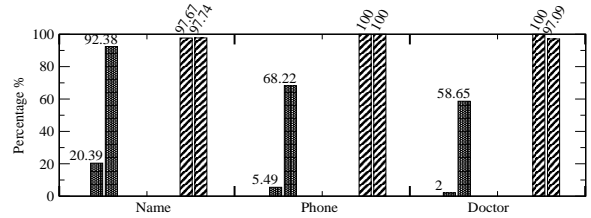
4.1 Experimental Setup

The following sequence of steps were carried out in our experiments: (1) generating data sets; (2) labeling examples; (3) learning PAEs from examples and using PAEs to extract data from Web pages; (4) manually verifying precision and recall metrics on extracted data.

Domains and Data Sets. We chose two different domains: veterinarian service providers and lighting products. For the former we focused on *referral* pages such as the one shown in Figure 1. We used a keyword-based Web search engine to retrieve a large collection of veterinarian service provider Web pages. From this collection we randomly selected 170 referral pages from different service provider Web sites. For the product domain we randomly collected 24 Web pages pertaining to lighting products. Each such page lists several product descriptions. These pages were downloaded from

Attribute Name	Actual Present	Ontology Identified	Nonredundant PAEs	Ambiguity Resolution
Hospital Name	2060	1667	420	1903
Phone Number	1841	963	101	1256
Doctor Name	1705	453	34	1000

(a)



(b)

Figure 4. Aggregated Extraction Results for Veterinarian Sites

Attribute Name	PAE #	Actual Present	Ontology Identified	Consistent PAEs	Recall %	Precision %
Hospital Name	47	338	287	294	86.98	97.35
Phone Number	18	111	47	100	90.09	100
Doctor Name	7	39	26	32	82.05	100

(a)

Attribute Name	Consistent PAEs #	Recall %	Precision %
Hospital Name	47	92.38	97.74
Phone Number	18	68.22	100
Doctor Name	7	58.65	100

(b)

Figure 5. Extraction Results for Consistent PAEs in the Veterinarian Domain

4 different Web sites: 2 from Kmart, 3 from OfficeMax, 13 from Staples, and 6 from Target.

The task of data extraction is considerably simplified by identifying the boundaries of individual entities in a Web page. For instance, in Figure 2, every subtree rooted at *tr* encapsulates a service provider entity. The problem of locating such entity blocks, referred to as *record-boundary discovery* in the literature, has been addressed in a number of previous works [1, 17, 20, 13, 4, 8]. Prior to applying our data extraction algorithms we identified the boundaries of entities in a Web page.

Labeling. We used two ontologies (using ontologies for data extraction is a known idea [7]) for labeling examples: one for veterinarian services and the other for lighting products. The attributes for veterinarian services comprised *HospitalName*, *PhoneNumber*, and *DoctorName*, while *Name* and *Price* were used for lighting products.

Two keywords, *hospital* and *clinic*, were used to identify examples of the *HospitalName* attribute while one keyword, *DVM*, was used for the *DoctorName* attribute. For the *PhoneNumber* attribute we used the regular expression (in Perl syntax), $[0-9]\{3\}-[0-9]\{3\}-[0-9]\{4\}$, as the identifier function. We used the keywords, *lamp*, *bulb*, and *tube*, for the *Name* attribute of lighting products and searched the symbol \$ to label examples of the *Price* attribute.

Extraction Process. Our experiments were conducted on 1GHz Pentium 4 machines each with 256MB memory. All algorithms were programmed in Java. All Web pages were parsed into DOM trees and the entity blocks identified. Note that using DOM trees for data extraction is a known idea (e.g., [19]). Next the identifier functions associated with the attributes in the ontology were used to generate training examples. Based on these examples nonredundant

PAEs were learned and then used to extract attribute data from each entity block. The two-step ambiguity resolution procedure described in Section 3 was applied to make 1–1 associations. This amounts to a strong bias towards high-precision rules [5]. Finally, precision and recall metrics were manually calculated from the extracted data.

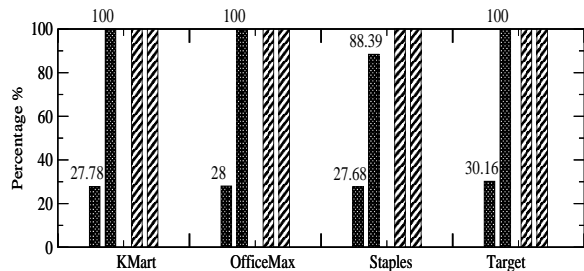
4.2 Analysis of Experimental Results

Nonredundant PAEs and Ambiguity Resolution. Figure 4 summarizes the recall and precision performance of extraction using nonredundant PAEs and the effect of ambiguity resolution. These results were aggregated over 170 veterinarian Web pages. In Figure 4(a), Column 2 lists the number of *actual* occurrences of each attribute in these pages. Column 3 shows the number of attribute values that were identified by the ontology. For example, search on the keywords, *hospital* and *clinic*, identified 1667 occurrences of the *HospitalName* attribute. Column 4 is the number of 1–1 associations between a nonredundant PAE and an attribute data value. For example, there were 420 such associations for the *HospitalName* attribute. Column 5 is the number of 1–1 associations that were made by the ambiguity resolution procedure. For instance, it resolved 1903 hospital names uniquely. Correctness of an association was manually verified over all the pages.

Figure 4(b) summarizes as bar charts the recall (shaded bars) and precision (checkered bars) performance of nonredundant PAEs, both before (illustrated by the bar on the left) and after (illustrated by the bar on the right) ambiguity resolution. Observe that there is a significant increase in recall with no loss in precision for all the three attributes after ambiguity resolution. This illustrates the effectiveness of our ambiguity resolution procedure.

Site	Attribute Name	Actual Present	Ontology Identified	Consistent PAEs
KMart	Name	18	5	18
	Price	18	18	18
OfficeMax	Name	25	7	25
	Price	25	25	25
Staples	Name	112	31	99
	Price	112	112	99
Target	Name	63	19	63
	Price	63	63	63

(a)



(b)

Figure 6. Extraction Results for Individual Product Sites

Consistent PAEs. Sometimes the PAEs generated by algorithm *LearnPAE* turned out to be consistent. We counted the number of consistent PAEs generated by algorithm *LearnPAE* and collected the recall and precision numbers only for those Web pages that generated such PAEs. In the table of Figure 5(a), Column 2 is the total number of Web pages where the nonredundant PAE for an attribute was consistent w.r.t. positive and negative examples. Columns 3 and 4 show the actual number of instances of that attribute in these pages and the number of instances identified by the ontology, respectively. Column 5 is the count of correct (manually ascertained) attribute data values extracted by these consistent PAEs. Columns 6 and 7 are the recall and precision numbers based on the 1–1 associations made *prior to* ambiguity resolution. In contrast observe the relatively low recall of nonredundant PAEs prior to ambiguity resolution (see the bars on the left in Figure 4(b)). This experimentally validates that consistent PAEs have superior recall and precision than nonredundant PAEs.

For yet another evidence of the superiority of consistent PAEs, observe in Figure 4(b) that after ambiguity resolution the *overall* recall of the *HospitalName* attribute is better than the *PhoneNumber* attribute which in turn is better than that of the *DoctorName* attribute. The reason can be readily explained by the number of consistent PAEs generated for the corresponding attributes, which have a high impact on the effectiveness of the bipartite graph resolution procedure. Observe in Figure 5(b) that this number is the highest for the *HospitalName* attribute and is the lowest for the *DoctorName* attribute.

Unambiguous PAEs. We ran algorithm *LearnPAE* to generate a *pair* of PAEs for extracting the *Name* and *Price* attributes from the lighting products pages of four different Web sites. These pages were all “well-structured” in the sense that the pair of PAEs produced by *LearnPAE* for each page turned out to be almost always unambiguous w.r.t. the examples identified by the ontology. The statistics for both attributes are shown in Figure 6(a), which can be interpreted the same way as those in the table of Figure 4(a). In Figure 6(b), we compare the recall (shaded) and preci-

sion (checkered) numbers of the identifier functions in the ontology (the left bar) and the unambiguous PAEs (the right bar). The statistics in Figure 6(b) are collected on the *Name* attribute only. Observe that precision is 100% while recall is almost close to 100% which experimentally demonstrates the superior quality of unambiguous PAEs.

5 Related Work

We model the notion of precision and recall that arises in wrapper building as a grammar inference problem. Therefore, to put our work in perspective we now review research in related areas, namely, grammar inference, sequence learning, and wrapper construction.

Grammar Inference. This well-known problem was first addressed in the seminal works of Gold and Angluin. Gold [9] showed that the problem of inferring a DFA of *minimum* size from positive examples is NP-complete. In [2] Angluin showed that the problem of learning a regular expression of *minimum* size from positive and negative examples is NP-complete. In our problem, however, we do not impose any constraint on the size of the PAEs learned. Our problems of learning consistent PAEs and unambiguous lists of PAEs (the problems considered in this paper) do not have equivalent counterparts in the classical works on grammar inference. Hence none of the known results is applicable.

Sequence Learning. There is a large body of work on learning subsequences and supersequences from a set of strings. The following problems are all NP-complete: (1) finding the shortest common supersequence (SCS) or the longest common subsequence (LCS) of an arbitrary number of strings over a binary alphabet [14, 18]; (2) finding a sequence which is a common subsequence/supersequence of a set of positive examples but not a subsequence/supersequence of any string in a set of negative examples [12, 15]. The semantics of PAEs differs substantially from string matching and hence their results are not applicable. Besides our problem of learning an unambiguous *list* of PAEs has no counterpart in these works.

Wrapper Construction. Research on wrapper construction for Web sources has made a transition from its early focus on manual [10, 19] and semi-automatic [1, 17] approaches to fully automated techniques based on machine learning [13, 16, 4, 5, 11, 6]. But the notion of ascribing a precision/recall metric to the learning of extraction expressions and its impact on algorithmic efficiency has not been explored in these works.

The issue of ambiguity resolution can also commonly arise in learning-based approaches to schema inference and data extraction from Web documents [6, 3] but has not been explored in the literature. Exponential-time and polynomial-time heuristics were proposed in [6, 3], respectively, to infer schemas from *unlabeled* Web documents for the purpose of automated data extraction. Recently we extended the results in this paper and showed that the problem of inferring good quality schemas from Web pages is NP-complete [21], even when the examples are labeled. Moreover, a collection of pages, generated from the *same* template, is required to learn a schema in [6, 3], but our learning techniques can apply to *individual* documents.

6 Conclusion and Future Work

Although we assumed that an attribute value resides in a single leaf node of a DOM tree, our results can be readily extended to those cases where an attribute value spans multiple leaf nodes — we will learn a list of PAEs per attribute. However, to deal with nested data types we will need to learn more expressive tree patterns. This will be an important and useful extension to our current work.

In our experimental setup, we made the simplifying assumption that record-boundary detection is able to eliminate “bad” examples. In practice the problem of “noise” in training examples cannot be ignored. We plan to address this issue and test our techniques on larger benchmark data in our future work.

Acknowledgments. This work was supported in part by NSF grants IIS-0072927, CCR-0205376, and CCR-0311512. The authors would like to thank the anonymous referees for their helpful comments and suggestions.

References

- [1] B. Adelberg. NoDoSe: A tool for semi-automatically extracting structured and semi-structured data from text documents. In *ACM International Conference on Management of Data (SIGMOD)*, 1998.
- [2] D. Angluin. On the complexity of minimum inference of regular sets. *Information and Control*, 39(3):337–350, 1978.
- [3] A. Arasu and H. Garcia-Molina. Extracting structured data from Web pages. In *ACM International Conference on Management of Data (SIGMOD)*, 2003.
- [4] N. Ashish and C. Knoblock. Wrapper generation for semi-structured Internet sources. *SIGMOD Record*, 26(4):8–15, 1997.
- [5] W. Cohen, M. Hurst, and L. Jensen. A flexible learning system for wrapping tables and lists in HTML documents. In *International World Wide Web Conference (WWW)*, 2002.
- [6] V. Crescenzi, G. Mecca, and P. Merialdo. RoadRunner: Towards automatic data extraction from large Web sites. In *International Conference on Very Large Data Bases (VLDB)*, 2001.
- [7] D. W. Embley, D. M. Campbell, R. D. Smith, and S. W. Liddle. Ontology-based extraction and structuring of information from data-rich unstructured documents. In *ACM International Conference on Information and Knowledge Management (CIKM)*, 1998.
- [8] D. W. Embley, Y. Jiang, and Y.-K. Ng. Record-boundary discovery in Web documents. In *ACM International Conference on Management of Data (SIGMOD)*, 1999.
- [9] E. M. Gold. Complexity of automaton identification from given data. *Information and Control*, 37(3):302–320, 1978.
- [10] J. Hammer, H. Garcia-Molina, S. Nestorov, R. Yerneni, M. M. Breunig, and V. Vassalos. Template-based wrappers in the TSIMMIS system. In *ACM International Conference on Management of Data (SIGMOD)*, 1997.
- [11] C.-N. Hsu and M.-T. Dung. Generating finite-state transducers for semi-structured data extraction from the Web. *Information Systems*, 23(8):521–538, 1998.
- [12] T. Jiang and M. Li. On the complexity of learning strings and sequences. *Theoretical Computer Science*, 119(2):363–371, 1993.
- [13] N. Kushmerick, D. S. Weld, and R. B. Doorenbos. Wrapper induction for information extraction. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 1997.
- [14] D. Maier. The complexity of some problems on subsequences and supersequences. *Journal of ACM*, 25(2):322–336, 1978.
- [15] M. Middendorf. On finding various minimal, maximal, and consistent sequences over a binary alphabet. *Theoretical Computer Science*, 145(1-2):317–327, 1995.
- [16] I. Muslea, S. Minton, and C. Knoblock. A hierarchical approach to wrapper induction. In *Third International Conference on Autonomous Agents (Agents’99)*, 1999.
- [17] M. Perkowitz, R. B. Doorenbos, O. Etzioni, and D. S. Weld. Learning to understand information on the Internet: An example-based approach. *Journal of Intelligent Information Systems*, 8(2):133–153, 1997.
- [18] K.-J. Räihä and E. Ukkonen. The shortest common supersequence problem over binary alphabet is NP-complete. *Theoretical Computer Science*, 16:187–198, 1981.
- [19] A. Sahuguet and F. Azavant. Web Ecology: Recycling HTML pages as XML documents using W4F. In *ACM SIGMOD Workshop on the Web and Databases (WebDB)*, 1999.
- [20] S. Soderland. Learning information extraction rules for semi-structured and free text. *Machine Learning*, 34(1-3), 1999.
- [21] G. Yang, I. V. Ramakrishnan, and M. Kifer. On the complexity of schema inference from Web pages in the presence of nullable data attributes. In *ACM International Conference on Information and Knowledge Management (CIKM)*, 2003.