Synthesis of Forgiving Data Extractors

Adi Omari
Technion
omari@cs.technion.ac.il

Sharon Shoham
Tel Aviv University
sharon.shoham@gmail.com

Eran Yahav
Technion
yahave@cs.technion.ac.il

ABSTRACT

We address the problem of synthesizing a robust data-extractor from a family of websites that contain the same kind of information. This problem is common when trying to aggregate information from many web sites, for example, when extracting information for a price-comparison site.

Given a set of example annotated web pages from multiple sites in a family, our goal is to synthesize a robust data extractor that performs well on all sites in the family (not only on the provided example pages). The main challenge is the need to trade off precision for generality and robustness. Our key contribution is the introduction of *forgiving extractors* that dynamically adjust their precision to handle structural changes, without sacrificing precision on the training set. Our approach uses decision tree learning to create a generalized extractor and converts it into a forgiving extractor, in the form of an XPath query. The forgiving extractor captures a series of pruned decision trees with monotonically decreasing precision, and monotonically increasing recall, and dynamically adjusts precision to guarantee sufficient recall.

We have implemented our approach in a tool called TREEX and applied it to synthesize extractors for real-world large scale web sites. We evaluate the robustness and generality of the forgiving extractors by evaluating their precision and recall on: (i) different pages from sites in the training set (ii) pages from different versions of sites in the training set (iii) pages from different (unseen) sites. We compare the results of our synthesized extractor to those of classifier-based extractors, and pattern-based extractors, and show that TREEX significantly improves extraction accuracy.

1 Introduction

We address the problem of synthesizing a robust data extractor based on a set of annotated web pages. Web sites often change their formatting and structure, even when their semantic content remains the same. A robust extractor [6, 15, 17] can withstand modifications to the target site. A non-robust extractor would have to be adjusted (typically manually) every time the formatting of a site changes.

Our idea is to construct a robust extractor by training it on a family of sites that have content that is semantically similar. We conjecture that the ability of a single extractor to handle multiple different sites means that the extractor is likely robust, as it is able to overcome differences between sites, and thus possibly also differences due to future changes of the same site.

The notion of a *family of sites* is somewhat vague, and we assume that it is user-defined. The intent is that sites of the same family contain the same kind of information, even if they differ on the way it is presented. Sites of the same family could be, for example, a family of book-selling sites, hotel reservation sites, etc. We conjecture that despite the fact that sites in a family may differ, they have a similar set of underlying semantic concepts, that a generalized extractor will be able to capture. For example, for book-selling sites, the notions of *author*, *title*, and *price* are likely to be presented in some way on all sites in the family.

Goal Given a set of training pages (HTML documents) within a family, where each page is annotated by tagging data items of interest, our goal is to synthesize a *robust extractor* that maximizes accuracy over the training set.

When constructing the generalized extractor for a family, there is a natural tradeoff between accuracy and generality. Constructing a precise extractor may prevent it from being robust to future changes. Constructing a loose extractor makes it more robust, but would yield results of poor accuracy. Both of these options are undesirable. Instead, our key contribution is the construction of *forgiving extractors* that adjust their precision dynamically and do not commit to a specific generalization tradeoff upfront.

Existing Techniques Manually writing a data extractor is extremely challenging. This motivated techniques for automatic "wrapper induction", learning extraction queries from examples [14]. Automated techniques reduce the burden of writing extractors, but still require manual effort (e.g., providing tagged samples for each site).

There has been a lot of work on pattern based techniques, using alignment of XPaths (e.g., [25, 20, 19]). These techniques learn paths to the annotated data items and generalize them to generate an extraction query. When provided with items that have significantly different paths (e.g., originate from different sites) these techniques may result in an overly relaxed query expression, and significant loss of precision. As a result, pattern based techniques are often limited to a single web site, and are sensitive to formatting changes.

Model based techniques [10, 8, 26, 13], use features of DOM elements to learn a model that classifies DOM items to *data* and *noise*. These methods have several drawbacks. First, they lack a standard execution mechanism for extraction (in contrast to XPath queries that are available for pattern based techniques). Second, the classifiers trained by these techniques are often hard to understand and modify. Last, but not least, the generalization in these models is

WSDM 2017, February 06-10, 2017, Cambridge, United Kingdom

DOI: http://dx.doi.org/10.1145/3018661.3018740

performed *at training time* and thus presents the standard dilemma between precision and generality.

Our Approach Our approach is based on the assumption that, despite differences in presentation, *some* sites in the family do share *some* local syntactic structural similarity (e.g., a book title may appear under some heading class, or its parent may be of some particular class). Our claim (which we support empirically) is that training on a few sites from the family will cover most of the local structural patterns. Following this insight, our approach tackles the problem in two steps:

(1) Precise generalization for the training set: given a set of annotated pages from different sites, we use decision tree learning to synthesize a *common XPath query* that *precisely* extracts the data items from all sites. Intuitively, the decision tree is used as a generalization mechanism that picks the important features for precise extraction. The overall set of features consists of XPath predicates that capture local syntactic structural properties. The synthesized query increases robustness, while maintaining precision on the training set.

(2) Dynamic generalization: the XPath query constructed in the previous step is precise for pages in the training set, but may be overly restrictive for extraction from other pages. To generalize to other pages, we introduce the novel notion of *forgiving XPaths*—a query that *dynamically relaxes its requirements*, trading off precision to make sure that target items are extracted.

The query generated by our approach has the benefits of both pattern based techniques and model based techniques. On the one hand, it is a standard XPath query, which has a wide support from web-browsers, programming languages and DOM parsers. This also makes it human readable, easy to review and modify by a programmer. On the other hand, the query has the flexibility and generalization ability of the techniques based on learning models.

Main Contributions The contributions of this paper are:

- A novel framework for synthesis of robust data extractors for a family of sites from examples (annotated web pages).
- A new technique for generalization of XPath queries using decision trees and "forgiving XPaths", which adjust precision dynamically.
- An implementation of our technique in a tool called TREEX and an experimental evaluation of the robustness and generality of the forgiving extractors. We evaluate precision and recall on: (i) different pages from sites in the training set (ii) pages from different versions of sites in the training set (iii) pages from different (unseen) sites. Our evaluation shows that TREEX is able to synthesize robust extractors with high precision and recall based on a small number of examples. Further, comparison to existing pattern based and model based techniques shows that TREEX provides a significant improvement.

2 Overview

In this section, we provide an informal overview of our approach. We elaborate on the formal details in later sections.

2.1 Motivating example

Consider the problem of synthesizing an extractor of book information based on examples from three book-seller sites: abebooks.com (ABE), alibris.com (ALIBRIS) and barnesandnoble.com (B&N). To simplify presentation, we focus on an extractor for the *author* attribute.

Fig. 1 shows simplified fragments of HTML documents presenting book information from the three different sites. We annotated

```
Abe
<div id="bookInfo">
 <h1 id="book-title">Rachael Ray 30-Minute Meals 2</h1>
 <h2>
   <a userselected href="/servlet/Se.._-author">
     Ray, Rachael</a>
 </h2>...
</div>

B&N
<section id="prodSummary">
<h1 itemprop="name">History of Interior Design</h1>
 <span>
  by <a userselected href="/s/.._contributor..">
       Jeannie Ireland</a>
 </span>...
</section>

Alibris
<div class="product-title">
<h1 itemprop="name">La Baba del Caracol</h1>
 <h2>by
  <a userselected itemprop="author" itemscope href="..">
       C. Maillard</a>
 </h2>
 </div>
```

Figure 1: Fragments of webpages with the author attribute values for a book on three different book seller sites.

these documents by tagging the HTML nodes that contain instances of the *author* attribute with a special HTML-attribute, *userselected*.

Note that the three different sites have different structure, and that the attribute of interest (*author*) appears differently in each site. There are many ways to write an XPath query that extracts the author from each site. For example:

- `//div[@id="bookInfo"//a` for ABE
- `//section/span/a` for B&N
- `//a[@itemprop="author"]` for ALIBRIS

Alternatively, other example queries could be:

- `//a[@*[contains(.,"author")]]` for ABE and ALIBRIS
- `//*[contains(text(),"by")/a` for B&N (also ALIBRIS).

Some of these queries are more robust than others, and some would generalize better across site versions. For example, the query `//*[contains(text(),"by")/a` for extracting the author name in B&N is more general than the query `//section/span/a`, which relies on a specific nesting structure of `section`, `span`, and `anchor`. Because there are many possible queries that extract the same information, it may be possible to use the different sites to pick queries that would be more robust. However, it is not clear how to do this.

For example, using an alignment based XPath generation technique on our three training pages would result in an XPath like `//*/*/a`, which is too general, and would lead to significant loss of precision (as shown in Section 5). This is because alignment based techniques (and pattern based techniques in general) assume that documents in the training set have a shared template. They therefore fail to handle significant structural differences between documents in the training set, as occur when the training set contains documents from multiple sites. Pattern-based generalization techniques also produce queries that are sensitive to changes, and are therefore not robust.

2.2 Our Approach

Our extractor synthesis process attempts to automatically pick the queries that would be more robust and general. The synthesis starts by a precise query generalization for the training set, which is obtained by learning a decision tree for identification of the tagged

nodes. This is followed by a dynamic generalization, which is obtained by creating XPath queries which we call *forgiving*.

Precise Generalization for the Training Set Synthesis starts by using the annotated documents to learn a decision tree which correctly identifies all the tagged HTML nodes. To do so, our method first extracts features that describe the tagged HTML nodes.

Extracting features: Feature extraction is done by traversing the paths from tagged nodes to the root and collecting features from nodes on the path and from their surrounding context. The extracted features are all valid XPath predicates (explained in Section 3.1). These features are used as tests in the inner nodes of the decision tree, and are the building blocks of our XPath queries.

Consider the *author* node in the ALIBRIS website (Fig. 1). Our feature extraction step starts from the `a` node having the *users-elected* attribute, and traverses the path to the HTML root node though the nodes `h2` and `div`, while collecting features both from the nodes on the path and from their surrounding context. Among the extracted features are the following:

- `ancestor-or-self::h2`, which matches nodes of type `h2` or an ancestor of type `h2` (later, e.g., in Fig. 2, we write `aos` as a shorthand for `ancestor-or-self`),
- `@itemprop="author"`, which matches nodes with the attribute `itemprop` having the value `author`, and
- `@*[contains(.,"author")]`, which matches nodes that *contain* the value `author`.

The feature `@*[contains(.,"author")]` is a generalization of the feature `@itemprop="author"`. Our feature extraction step generates such generalized features, in addition to the precise ones, in order to later enable the use of generalized structural features in the decision tree when possible (when the generalization does not result in loss of precision).

Learning a Decision Tree: Once the feature extraction step is complete, our method uses a recursive procedure to learn the decision tree, based on the extracted features. Our algorithm is a variant of the ID3 algorithm [23]. One of the differences is that our algorithm prioritizes generalized features when selecting the feature to use as a test in inner nodes. Technically, this is done by assigning costs to features, with generalized features having lower costs. Additional details and differences are described in Section 3.2.

Fig. 2 (T1) presents the decision tree generated by our algorithm to identify *author* nodes in ABE, B&N and ALIBRIS. The root node of T1 has `@*[contains(.,"author")]` as a test. This is because this feature has the highest improvement-to-cost ratio: it results in the highest information gain, when prioritizing low-cost (generalized) features. The *true*-branch of the root leads to a node with test `aos::h2`, while the *false*-branch leads to `@*[contains(.,"contributer")]` as a test.

A decision tree has a natural recursive translation into a valid XPath query that extracts the nodes identified by the decision tree. The XPath query x1 presented at the bottom of Fig. 2 is the extraction query generated from T1 for the *author* attribute.

Dynamic Generalization using Forgiving XPaths The decision tree learned from the annotated HTML documents, and the corresponding XPath query, identify the annotated nodes as precisely as possible (depending on the extracted features). However, in order to improve the ability to extract semantically similar nodes in other versions of the web pages, some generalization is desired. A natural generalization is by pruning the decision tree, turning some inner nodes to accepting nodes. Such generalization trades off precision on the training set for a potentially higher recall on other pages. However, the question remains where to prune the decision tree, and how much precision to sacrifice.

As an example, consider a modified version of one of the sites where books also contain an author attribute. Fig. 3 shows such an example HTML fragment. Fig. 2 shows three trees T1, T2 and T3 and their respective XPath translations x1,x2 and x3. As explained above, T1 is the tree learned by our algorithm when applied on ABE, B&N and ALIBRIS. T2 is a pruned version of T1, with a slightly lower precision, while T3 is a fully-pruned tree accepting every node. Running x1 on the modified site returns no results. Using x2 will have better performance on the modified site, however, it will have a lower precision on sites from the training set.

Forgiving XPaths: Our goal is to find the highest level of precision that obtains sufficient recall. As the books example demonstrates, this level might be different for different sites. An important aspect of our approach is the introduction of *forgiving XPaths*, which allow us to postpone the decision of how much precision to sacrifice to the evaluation time of the XPath, and hence adjust the generalization to the web page at hand.

Technically, a forgiving XPath query is an XPath query obtained as a union of several queries (using the "`|`" XPath operator). Each query exhibits a different precision level on the training set, and is conditioned by the more precise queries not extracting any nodes. This means that at run time, when invoked on a HTML document, the forgiving XPath will evaluate to the query with the maximal precision level that extracts a non-empty set of nodes. This property allows the forgiving XPath to perform well on both seen and unseen documents, avoiding the trade-off between precision on seen data and recall on unseen data. It is important to note that our forgiving XPath query is a standard XPath query that may be used in any XPath interpreter without the need for modifications of any kind.

We construct the forgiving XPaths from a series of XPaths with monotonically decreasing precision, such as x1, x2 and x3 above. In the books example, we derive the following forgiving XPath:

$$fx = \texttt{/*/x1|/*[not(x1)]/x2}.$$

fx uses the root node (which never contains data), to enable returning the result of invoking x2 only when x1 returns no results.

3 Decision Tree Learning

In this section we present the first step of our approach for synthesizing forgiving XPaths, which constructs a decision tree based on a given set of annotated web pages.

Notation Given a set of web documents $D = \{d_1, \ldots, d_n\}$, where each d_i is represented as a DOM tree with a set A_i of nodes, and given a target set $N_i \subseteq A_i$ of nodes of interest for each $d_i \in D$, we denote by $A = \bigcup_{i=1}^{n} A_i$ the global set of DOM nodes, and by $N = \bigcup_{i=1}^{n} N_i$ the global target set of nodes of interest. We also denote the remainder of the nodes by $\overline{N} = A \setminus N$.

Our approach starts with a feature extraction phase, where we generate a set F of features based on the DOM nodes in N. We then construct a decision tree that uses the features in F to classify A into N and $\overline{N}$. The decision tree will serve as the basis for the creation of a forgiving XPath, as described in Section 4.

3.1 Feature Extraction

For every individual document d_i in the training set D, there are many correct XPaths for the extraction of N_i. Each such XPath might use different structural features of the document. To improve the ability to generate a concise *common* XPath for all of the documents in D, we first generate a large set of structural features.

In the feature extraction phase, our method extracts features of N that are expressible by valid XPath queries. Each feature f is defined by an XPath predicate, with the meaning that the value of

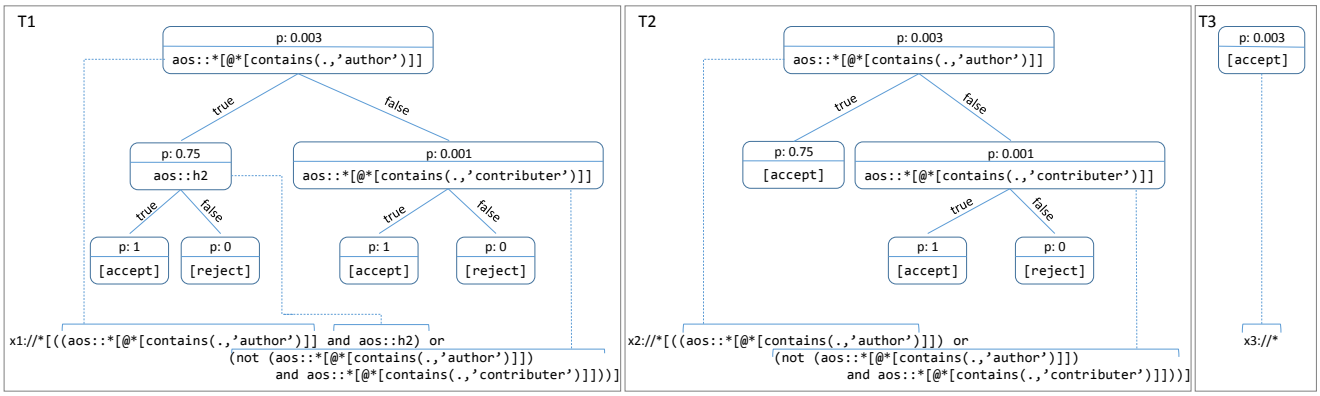

Figure 2: Example decision trees over XPath predicates and their respective XPath translations.

```
<p id="product_subtitle">.. in Simple Words</p>
<p id="product_author">By
 <a class="blue_link" href="...">Randall Munroe</a></p>
```

Figure 3: Fragment of a modified book information page.

f in a DOM node is 1 iff the value of the XPath predicate on the same node is *true*.

The extracted features of a node in N are divided to *node features*, and *context features*. The extent of the surrounding context that we consider is configurable. In our implementation, context features consist of *children features* and *ancestor features*.

Node features consist of:

- **Node Name** the node name is used to create a node-test feature (for instance, `self::a`).
- **Text Content** the text content of the text children of a node is used to define text-equality features and text-containment features. For instance, the predicate `text()="Sale price:"` is a text-equality feature, which holds iff the current node has a text child node whose value equals "Sale price:". On the other hand, the predicate `contains(text(),"price")` is a text-containment feature, that requires the current node to have a text child node containing the string "price".
- **Node Attributes** the node attributes define attribute features. Each attribute consists of a name and value pair (for instance, `id = "product_title"`). Attribute features are of three different types:
  1. Attribute name features (e.g. `@class` requires having an attribute with name "class")
  2. Attribute value features with or without conditions on the attribute name (e.g. `@*="title"` requires having an attribute with value "title", while `@id="title"` requires having an attribute named "id" and value "title")
  3. Text-containment features on attribute values, with or without conditions on the attribute name. For instance, `@*[contains(.,"title")]` requires that some attribute has a value that contains the token "title".

Context features are divided to:

- **Children Features** node features as defined above are extracted also for the children of the node (e.g., `child::div` is a predicate that evaluates to *true* on nodes that have children with node name 'div').
- **Ancestor Features** node features and children features as defined above are extracted also for ancestors of the node (e.g., `ancestor-or-self::*[child::span]` evaluates to *true* on nodes with ancestors that have children named 'span').

Technically, the feature extraction process is performed on each document $d_i \in D$ separately. For each document d_i, our method starts from the nodes in N_i. For each node it computes its node and children features and ascends up the path to the DOM tree root, while gathering ancestor features. To generate text-containment features we tokenize the relevant text and use the resulting tokens to define containment conditions `contains(text(),token)`.

Features as XPath Queries Each of the extracted features f is an XPath predicate. It induces an XPath query `//*[f]` that, when invoked on the root of a DOM tree, extracts the set of all nodes in the DOM tree that satisfy f. With abuse of notation we sometimes refer to the feature f itself as the induced XPath query.

3.2 Decision Tree Learning

Given the extracted set of features F, we use a variant of ID3 to learn a decision tree that classifies A into N and $\overline{N}$.

Decision tree A decision tree T is a complete binary labeled directed tree, in which each inner node (including the root) is labeled by a feature $f \in F$, and has exactly two children: a 0-child, and a 1-child (each child corresponds to a value of f). We also refer to the children as *branches*. The leaves in the tree are labeled by 0 or 1, where each value corresponds to a possible class. The elements in A whose feature vectors match paths in T from the root to a leaf labeled by 1 form the 1-class induced by T. The 0-class is dual.

Decision Tree Learning Given the partitioning of A into N and $\overline{N}$, and given a set of features F, ID3 constructs a decision tree T whose 1-class is N and whose 0-class is $\overline{N}$. (Note that a correct classifier exists only if whenever two elements $e_1, e_2 \in A$ share a feature vector, they also share a class.)

The construction of the decision tree is recursive, starting from the root. Each node t in the tree represents the subset of the samples, denoted $S(t)$, whose feature vector matches the path from the root to t. If all samples in $S(t)$ have the same classification, then t becomes a leaf with the corresponding label. Otherwise, ID3 may choose to partition $S(t)$ based on the value of some feature $f \in F$. The node t is then labeled by the feature f, and the construction continues recursively with its children, where its 1-child represents the subset of $S(t)$, denoted $S(t)_f$, in which f has value 1, and its 0-child represents the set $S(t)_{\overline{f}} = S(t) \setminus S(t)_f$ where f has value 0. In our setting these sets are computed by running the feature $f \in F$

on every $d \in D$ and computing $S(t)_f = (\bigcup_{d \in D} [\![f]\!]_d) \cap S(t)$, and $S(t)_{\overline{f}} = S(t) \setminus S(t)_f$.

ID3 selects the feature $f \in F$ to split on as the feature with the highest *information gain* for $S(t)$, where the information gain [18], denoted $IG(S(t), f)$, of $S(t)$ based on f measures the reduction in uncertainty after splitting $S(t)$ into $S(t)_f$ and $S(t)_{\overline{f}}$. Our learning algorithm modifies ID3 in two ways: (i) features have costs that are taken into account in the computation of the information gain. (ii) the choice of a feature f to split on is restricted in a way that ensures a correlation between an element having value 1 for f in the feature vector and the element being classified into the 1-class. We elaborate on these modifications below.

Feature Costs We consider features with different costs. Intuitively, the more specific the feature is, the higher its cost. We denote the cost of $f \in F$ by $Cost(f)$. In order to prioritize generalized features over more specific ones, we define the information-gain-to-cost ratio and use it instead of the information gain:

DEFINITION 1. *The* information-gain-to-cost ratio *of a feature f for a set $S \subseteq A$, denoted $IGR(S, f)$, is:*

$$IGR(S, f) = IG(S, f)/Cost(f)$$

Feature-Class Correlation Guided by the intuition that, in many cases, existence of features is important for being classified as part of the 1-class, while their nonexistence is incidental, we create a decision tree that correlates feature values with the classification: specifically, value 1 for a feature is correlated with class 1.

The feature-class correlation is obtained by restricting the set of features to split $S(t)$ on only to the subset $F_t \subseteq F$ for which $S(t)_f$ (which will form the 1-branch if the split by f takes place) has more instances from N than $S(t)_{\overline{f}}$ (which will form the 0-branch).

Among all the features in F_t, the feature f with the highest $IGR(S(t), f)$ is selected as the feature to split t.

4 Forgiving XPath Synthesis

Building a decision tree is an intermediate step towards synthesizing a forgiving XPath query. In this section we define the notion of forgiving XPaths and describe their synthesis.

Forgiving XPaths A forgiving XPath is constructed based on a series of XPaths $x_0, x_1, \ldots, x_k$ s.t. for every $0 \leq j < k$ and $d \in D$, $[\![x_j]\!]_d \subseteq [\![x_{j+1}]\!]_d$, where $[\![x]\!]_d$ is the set of nodes extracted by x from d. Note that if x_0 is precise for every seen document, then the precision of the XPaths in the sequence is monotonically decreasing, but the recall on unseen documents is potentially increasing.

DEFINITION 2. *Given a series of XPaths $x_0, x_1, \ldots, x_k$ as defined above, a* forgiving XPath *is defined by*

$$fx = /*/x_0 \mid /*[not(x_0)]/x_1 \mid \ldots \mid /*[not(x_{k-1})]/x_k$$

We say that x_0 is the base *of fx.*

fx uses the union operator ("|"), which means that it extracts the union of the nodes extracted by the individual queries. However, the queries are constructed in such a way that for every document d, seen or unseen, $[\![fx]\!]_d = [\![x_j]\!]_d$ for the least j such that $[\![x_j]\!]_d \neq \emptyset$. This is ensured since the XPath x_j is applied on $/*[not(x_{j-1})]$, which will extract an empty set of nodes if $[\![x_{j-1}]\!]_d \neq \emptyset$, and will extract the root node of d otherwise. Therefore, at run time, for every document, *fx* evaluates to the most precise query in the series that actually extracts some nodes. In particular, if x_0 has perfect precision and recall for the training set, so will *fx*.

In the following, we present the construction of a forgiving XPath from a sequence of XPaths that are monotonically decreasing in their precision. We obtain such a sequence by first obtaining a sequence of decision trees with monotonically decreasing precision.

4.1 Decision Trees with Varying Precision

We measure the precision and recall of a decision tree T by the precision and recall of its 1-class w.r.t. the "real" class, N. We use pruning to define a sequence of decision trees with monotonically decreasing precision scores. Our motivation for doing so is to gradually trade off precision (on the training set) for a better recall on new documents. Pruning is based on the nodes precision:

DEFINITION 3. *The* precision *of a node $t \in T$ is defined as* $precision(T, t) = \frac{|S(t) \cap N|}{|S(t)|}$.

In Fig. 2, the "p" labels of the inner nodes denote their precision. For instance, the node `aso::h2` in T1 has a precision of 0.75, which means that 75% of the elements in the training set that reach it when classified by T1, are from N.

Tree Pruning: Limiting Precision In the construction of the decision tree, a node with precision 1 becomes a 1-labeled leaf. We prune a tree by limiting the maximal precision of its nodes; Formally, given a precision threshold α, the method $prune(T, \alpha)$ prunes every subtree of T whose root t has $precision(T, t) \geq \alpha$, and turns t to a leaf labeled by 1. Thus, the induced 1-class is increased.

For example, T2 in Fig. 2 results from pruning T1 by limiting the precision of its nodes to 0.75.

Clearly, the tree T' obtained after pruning has a lower precision score than T (since the 1-class defined by T' is a superset of the 1-class defined by T). However, the recall score of T' can only increase compared to T: The recall score of T on the training set is already maximal, hence it is not improved by pruning. However, when considering new documents, the recall score of the pruned tree, T', can be strictly higher than that of T.

Layered Decision Tree Sequence Given the decision tree T learnt in Section 3.2, we create a sequence of decision trees with a monotonically decreasing precision and a monotonically increasing recall as follows:

DEFINITION 4. *Let $\alpha_0 > \alpha_1 > \ldots > \alpha_k$ be the sequence of precision scores of nodes in T, in decreasing order, i.e., $\{\alpha_0, \ldots, \alpha_k\} = \{precision(T, t) \mid t \in T\}$. The* layered decision tree sequence *based on T is the sequence $T_0, T_1, \ldots, T_k$ of decision trees, such that $T_0 = T$, and for every $0 \leq i < k$, $T_{i+1} = prune(T_i, \alpha_{i+1})$.*

Note that it is possible that as a result of pruning, a node with precision score α_{i+1} no longer exists in T_i. In this case, the definition implies that $T_{i+1} = T_i$. However, we will simply omit T_{i+1} from the sequence. Therefore, each layer in the layered sequence reflects a "one step" reduction in the precision of the decision tree (and hence, potentially, an increase in the recall).

For example, the decision trees T1, T2 and T3 depicted in Fig. 2 form a layered decision tree sequence based on T1.

4.2 Translation of Decision Trees to Forgiving XPaths

The features used in the decision trees are all XPath predicates. This allows for a natural translation of a decision tree to an XPath query that extracts the set of nodes classified as belonging to the 1-class by the decision tree. In the following we describe this translation, which we use to generate a series of XPaths from the layered decision tree sequence generated from T (see Definition 4). We use the series to construct a forgiving XPath as defined in Definition 2

Decision Tree to XPath The translation of a decision tree $\tilde{T}$ to an XPath is performed by a recursive procedure, *GetXPath*(t), which is invoked on the root t of $\tilde{T}$ and returns an XPath predicate.

The procedure *GetXPath*(t) works as follows:

1. if t is a leaf node, return *true*() if it labeled 1, and *false*() if it is labeled 0.
2. if t is an inner node, labeled by feature f, then call *GetXPath* on its 0-child, t_0, and on its 1-child, t_1. Let $p_0 = \mathit{GetXPath}(t_0)$, and $p_1 = \mathit{GetXPath}(t_1)$. Then return the XPath predicate:

$$\texttt{(f and } p_1\texttt{) or (not(f) and } p_0\texttt{)} \qquad (1)$$

Finally, from the XPath predicate p returned by the invocation of *GetXPath* on the root of $\tilde{T}$, an XPath query `//*[p]` is constructed. When invoked on the root of a DOM tree, `//*[p]` extracts the set of all nodes in the DOM tree that satisfy p. We denote the XPath query `//*[p]` by $\mathit{XPath}(\tilde{T})$. The translation ensures that a DOM node is in the 1-class induced by $\tilde{T}$ iff it is in $[\![\mathit{XPath}(\tilde{T})]\!]$.

Fig. 2 shows the XPath queries x1, x2, x3 generated from T1, T2, T3, respectively.

5 Evaluation

In this section we evaluate the effectiveness of our approach. Our experiments focus on three different aspects: (i) the accuracy of the synthesized queries on different pages from sites that have some pages in the training set (seen sites), (ii) their accuracy on different versions (newer and older versions) of the same pages, and (iii) their accuracy on pages from sites that have no pages in the training set (unseen sites).

5.1 Implementation

We implemented our approach in a tool called TREEX, and used it to synthesize extraction queries for multiple websites. For comparison, we also implemented four other extractors:

- To represent model-based approaches we trained two classifiers, C4.5 [24] (J48 implementation) and naive bayes (NB) [11] (using Weka [9]). In each experiment we train the classifiers on the same training set based on the same features used to construct our extractor.
- To represent other XPath-based approaches, we have implemented an alignment-based XPath (XA) generation technique proposed in [20].
- In addition to forgiving XPath (FX) queries, we used our approach to generate non-forgiving XPath queries (NFX) by directly generating an XPath from the obtained decision trees.

5.2 Experimental Settings

Datasets We constructed three datasets (available at: goo.gl/a16tiG) to evaluate our approach and the baselines. The first dataset, denoted DS1, contains pages from 30 real-life largescale websites divided to four different categories: books, shopping, hotels and movies. For each category, a set of (typically 2 or 3) common attributes were selected as the target data to be extracted. DS1 contains 166 manually annotated pages. To construct the second dataset, denoted DS2, we used archive.org to obtain five different versions (current version and four older ones) of the same page from four sites (with two or three different attributes to extract). DS2 contains 55 manually tagged webpages. The third dataset, denoted DS3, contains 5025 product pages from five widely known websites. The pages were annotated (with two attributes, product name and price) using manually written XPaths.

In our experiments we used different subsets of the dataset as training sets and evaluation sets.

Performance Metrics Given a page d from one of the datasets and a target attribute for extraction $attr$, we denote by $N_{attr}(d)$ the set of tagged nodes containing instances of $attr$ in d. For a data extractor x extracting instances of $attr$, we measure the precision, recall and F-measure of x on d by considering $N_{attr}(d)$ as a target set. The precision, recall and F-measure of x (for $attr$) are defined as the averages over all pages d in the evaluation set containing instances of $attr$. The reported precision (P), recall (R) and F-measure (F1) for each approach are the averages over all the data extractors x that it synthesizes (for all categories and all attributes).

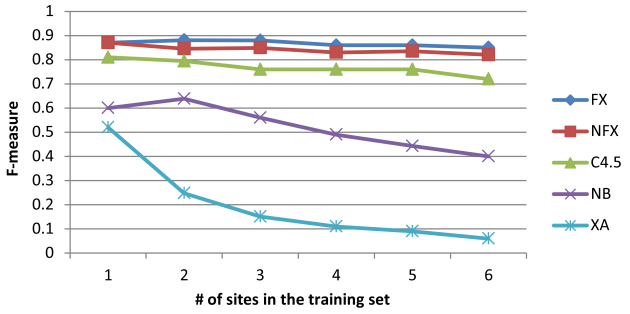

Figure 4: The F-measure values of different approaches for seen sites, as a function of number of sites in training set.

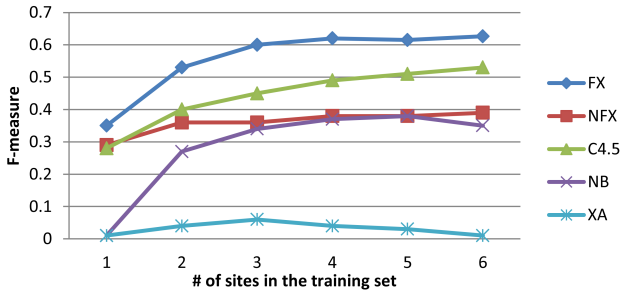

Figure 5: The F-measure values of different approaches for unseen sites, as a function of number of sites in training set.

	$s=2$	$s=3$	$s=4$	$s=5$	$s=6$
FX	0.6%	0.8%	-0.8%	-1.5%	-2.6%
NFX	-3.9%	-3.0%	-4.2%	-5.0%	-5.8%
C4.5	-1.8%	-5.6%	-6.5%	-7.7%	-10.6%
NB	6.8%	-7.3%	-17.9%	-26.0%	-34.4%
XA	-51.3%	-70.1%	-77.8%	-83.3%	-89.2%

Table 1: Performance (F-measure) decrease on seen sites as a function of the size of the training set (s). Lower decreases (negative values with smaller absolute value) indicate higher robustness to structural differences in the training set.

5.3 Evaluating the Different Performance Aspects

Accuracy on Seen Sites We evaluated the effectiveness of our approach (and the baseline approaches) for data extraction by evaluating the accuracy of the queries it generates on the different sites in the dataset. This is the standard use case, where we generate a query and evaluate it on each site separately.

In addition, we used our approach (and the baselines) for synthesizing a *single* generalized query for *multiple* sites. In particular, this allowed us to evaluate the robustness of our approach to structural differences in the training set. Such robustness is important to allow training the tool on a larger training set with a variety of websites without harming accuracy.

Accuracy on Different Page Versions To evaluate the robustness

	$s=1$			$s=2$			$s=3$			$s=4$			$s=5$			$s=6$		
	P	R	F1	P	R	F1	P	R	F1	P	R	F1	P	R	F1	P	R	F1
Performance on Documents from Seen Sites																		
FX	0.86	0.93	0.87	0.86	0.94	0.87	0.87	0.93	0.88	0.86	0.91	0.86	0.85	0.90	0.86	0.85	0.89	0.85
NFX	0.86	0.90	0.87	0.83	0.87	0.84	0.84	0.87	0.85	0.84	0.85	0.83	0.83	0.85	0.83	0.84	0.84	0.82
C4.5	0.80	0.87	0.81	0.72	0.93	0.79	0.69	0.91	0.76	0.68	0.91	0.76	0.67	0.92	0.75	0.65	0.91	0.72
NB	0.55	0.92	0.60	0.55	0.93	0.64	0.45	0.93	0.56	0.39	0.92	0.49	0.34	0.92	0.45	0.29	0.92	0.40
XA	0.53	0.54	0.52	0.24	0.38	0.25	0.14	0.33	0.15	0.10	0.29	0.11	0.07	0.28	0.09	0.04	0.26	0.06
Performance on Documents from Unseen Sites																		
FX	0.31	0.66	0.35	0.48	0.75	0.53	0.56	0.79	0.60	0.60	0.79	0.62	0.60	0.77	0.61	0.63	0.78	0.62
NFX	0.30	0.31	0.29	0.36	0.38	0.36	0.39	0.38	0.36	0.41	0.40	0.38	0.42	0.40	0.38	0.43	0.41	0.39
C4.5	0.28	0.32	0.28	0.38	0.48	0.40	0.43	0.56	0.45	0.46	0.60	0.49	0.48	0.63	0.51	0.51	0.66	0.53
NB	0.01	0.01	0.01	0.26	0.35	0.27	0.31	0.49	0.34	0.34	0.56	0.37	0.34	0.62	0.38	0.32	0.65	0.35
XA	0.01	0.01	0.01	0.04	0.07	0.04	0.05	0.12	0.06	0.04	0.14	0.04	0.03	0.14	0.03	0.01	0.11	0.01

Table 2: The average precision (P), recall (R) and F-measure (F1) of different approaches on seen and unseen sites, for different numbers of sites (s) in the dataset

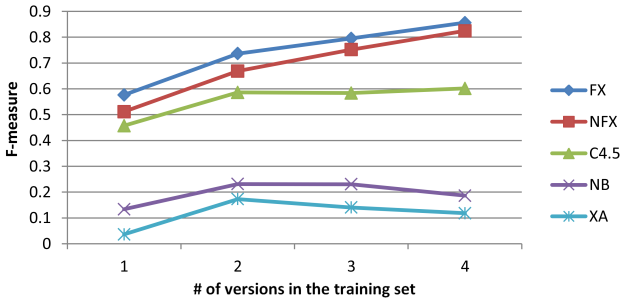

Figure 6: The F-measure values when evaluated on different page versions, as a function of number of versions in training set.

	$s=1$	3 $s=2$	$s=3$	$s=4$	$s=5$	$s=6$
author	0.18	0.25	0.52	0.68	0.78	0.82
price	0.36	0.62	0.66	0.66	0.62	0.55
title	0.94	0.92	0.93	0.93	0.96	0.98

Table 3: The average F-measure (F1) for forgiving XPaths (FX) on different book attributes in unseen sites.

	$s=1$			$s=2$			$s=3$			$s=4$		
	P	R	F1	P	R	F1	P	R	F1	P	R	F1
FX	0.53	0.88	0.58	0.72	0.88	0.74	0.81	0.88	0.80	0.87	0.93	0.86
NFX	0.52	0.52	0.51	0.69	0.67	0.67	0.78	0.75	0.75	0.85	0.84	0.82
C4.5	0.46	0.48	0.46	0.53	0.75	0.59	0.52	0.80	0.58	0.55	0.84	0.60
NB	0.11	0.27	0.13	0.18	0.59	0.23	0.18	0.71	0.23	0.15	0.76	0.19
XA	0.04	0.04	0.04	0.17	0.23	0.17	0.14	0.20	0.14	0.13	0.21	0.12

Table 4: The average precision (P), recall (R) and F-measure (F1) of different approaches trained on a set of size s of page versions and tested on the rest.

of queries generated by our method (and other baselines) to structural changes in the pages, we synthesized the queries using some versions of the page and evaluated them on different versions. Robustness to future changes on the page structure is important, as a more robust query will break less frequently.

Accuracy on Unseen Sites One major advantage of our technique compared to other XPath-based extraction techniques (in addition to its ability to synthesize a single query for multiple sites) is its generalization ability and the flexibility of its resulting queries. Other XPath based techniques work on pages from a single site, and generate XPath expressions that are strictly related to the structure of these pages.

To evaluate the generalization ability of our approach, we evaluated the extractors on unseen sites. Note that our goal is only to compare the generality of queries synthesized by our method to the other baselines; we do not expect the accuracy on unseen sites to be as good as for seen sites, as this is a much more challenging task.

	FX						XA					
	Name			Price			Name			Price		
	P	R	F1	P	R	F1	P	R	F1	P	R	F1
currys	1.00	1.00	1.00	0.92	1.00	0.96	1.00	1.00	1.00	1.00	0.65	0.79
pricespy	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.26	0.42
bestbuy	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.99	0.99
pricerunner	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.67	0.80	1.00	0.67	0.80
ebuyer	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.26	0.42	1.00	1.00	1.00
overall	1.00	1.00	1.00	0.98	1.00	0.99	1.00	0.79	0.84	1.00	0.71	0.80

Table 5: Performance of forgiving-XPath (FX) and Alignment based approach (XA) on the XPath generated data set (DS3)

5.4 Methodology

We evaluated our approach on seen and unseen sites by applying it on sets of sites from the DS1 dataset of different sizes (i.e., with different number of sites in the training set). We denote the number of sites in the training set in each experiment by s.

For every category among books, shopping, hotels and movies, for which the dataset contains documents from 7 to 8 sites (we denote by n the number of sites in the category), we considered all the possible $\binom{n}{s}$ subsets of size s of sites in the dataset, where $s \in [1, 6]$ (while s can be as large as 8, we limit it to 6 to be consistent with the unseen case). For each subset of s sites, we created the subset of the dataset restricted to the pages of the selected sites. We denote the corresponding subset of the dataset by D. For evaluating the performance on seen sites, we split D to two disjoint subsets D_t (for training) and D_e (for evaluation). Each of the two subsets D_t and D_e contains document pages from each website. Hence the training set and the evaluation set refer to exactly the same sites (allowing us to evaluate the performance on seen sites), but they have no documents in common. We used our approach to synthesize an extraction query with D_t as a training set, and evaluated the precision, recall and F-measure of the resulting query on D_e.

In addition, we used DS3 to compare the performance of our approach to that of the alignment based approach [20]. For each site and attribute (among product name and price) DS3 contains 1005 pages, five of which are used for training and the rest are used for testing.

For evaluating the performance on unseen sites, we used our approach to synthesize an extraction query with D as a training set and evaluated the precision, recall and F-measure of the resulting query on $\overline{D}$. The set $\overline{D}$ contains documents in the dataset from the

rest of the sites from the same category (these sites are considered *unseen* – as their documents are not in D).

We did the same for all the other baseline approaches we have implemented. For every value of $s \in [1, 6]$ and for every category, we calculated the average precision, recall and F-measure over the different subsets D of the dataset obtained with s sites for both seen and unseen cases. Finally, for each value of $s \in [1, 6]$, we calculated the average over all categories. The results obtained for the two variants of our approach, as well as for the baselines, are reported in Table 2, Fig. 4 and Fig. 5 (as a function of s).

To evaluate our approach (and the baselines) on different page versions, we used a similar process as before, this time on pages from the DS2 dataset. In these experiments, s refers to the number of versions considered in the training set, D consists of a subset of page versions and $\overline{D}$ consists of the remaining versions. Results are reported in Table 4 and Fig. 6.

5.5 Results

Results for Seen Sites The results for performance on seen sites are reported in Fig. 4 and Table 2. The results show that the XPath queries generated by our approach (FX and NFX) have the best accuracy (F-measure) among the different approaches for all different values of s, well ahead of the closest competing approach C4.5. Both classifier based methods (C4.5 and NB) suffer from loss in precision, especially for higher s values. The alignment-based XPath generation (XA), which has the lowest precision and recall among all the approaches, suffers from significant decrease in precision for $s > 1$. According to Table 1, which reports the performance loss on seen sites as a function of the number of sites in the training set, our forgiving XPath method (FX) is the most robust approach to diversity in the training set. The results for performance of the alignment based XPath generation (XA) and our forgiving-XPath (FX) on DS3 are reported in Table 5. The XPath queries generated by our approach have better precision and recall than those generated by the alignment based approach. However, both approaches perform better on DS3 than on DS1 (our approach has perfect recall and close to perfect precision). This is because pages in DS3 were annotated using XPaths while pages in DS1 were manually annotated. It is therefore hard even for a human programmer to write an XPath that accurately extracts the data from them.

Results for Different Page Versions The results of invoking the data extractors on different (older and newer) versions of pages are reported in Fig. 6 and Table 4. The results show that when trained with a large enough set of different archived versions of a page, our forgiving-XPath (FX) has an F-measure of 0.86 on newer (different) yet unseen version of the page (0.93 recall and 0.87 precision, which is close to its performance on seen pages). This shows the usefulness of our approach for generating a robust extractor when used on a set containing a variety of previous versions of a target page. That is, it is more tolerant to structural changes in future versions of the page. Classifier-based methods show poor precision for higher s values, making them unsuitable for such an application (of generating robust extractor). Alignment-based XPath generation (XA) has the lowest scores (for all values of s).

Results for Unseen Sites Fig. 5 shows that our forgiving XPath approach (FX) has the best accuracy (F-measure) for every number of sites in the training set. Table 2 shows that our forgiving XPath technique significantly outperforms the other approaches in terms of both precision and recall.

While the non-forgiving XPath (NFX) performs well (very close to the forgiving XPath), and outperforms C4.5 on seen sites, it is outperformed by C4.5 in the case of unseen sites. This is because we use a pruned C4.5 tree, while we keep our non-forgiving XPath strict. This is also the reason that C4.5 is the closest competing approach to NFX. This highlights the trade-off between accuracy on seen and unseen sites when pruning is used (or not, as in NFX).

Table 3 shows an interesting aspect of the evaluation. The accuracy results that we report are averaged over all categories and all attributes extracted from a site. However, it is common for some attributes to be more robust and generalizable than others. For example, in sites listing books, the accuracy of extracting the book author and the book title is very high, and improves as more sites are added to the training set. In contrast, the accuracy of extracting the price, might deteriorate with more sites, as the sites may really have very little in common in the way that a price is represented (sites may even differ on the number of prices that they present, e.g., special discounts, shipping, etc.).

5.6 Discussion

The evaluation results show that forgiving XPath queries, synthesized by our approach, beat the accuracy of other approaches for all the three different performance aspects.

While some approaches (like NB and C4.5) have a trade-off between precision and generality (performance on seen and unseen sites), the use of forgiving XPaths enables our approach to have good performance on unseen sites without noticeable loss of precision on seen sites. In addition, the robustness of our approach to the structural differences of documents in the training set enables using datasets with high structural variety (containing different versions of the page), which improves the robustness of the synthesized XPath to future structural changes.

6 Related Work

There has been a lot of work on data extraction from web pages. In the following we survey closely-related work.

XPath Data Extractors The adaptation of XPath based data extractors has been studied widely. Several works on automatic generation of XPath based data extractors have been proposed [25, 20, 19, 30, 1]. Nielandt et al. [20] propose a method for generating XPath wrappers, which revolves around aligning the steps within the XPaths. Given a set of XPath samples describing the data nodes of interest in a DOM tree, the method uses an alignment algorithm based on a modification of the Levenshtein edit distance to align the sample XPaths and merge them to a single generalized XPath. The XPaths generated by this method are not robust to structural changes. Our evaluation show that our technique outperforms this method both on seen and unseen sites. Their latest work [19] builds on [20] and enriches the resulting generalised XPaths with predicates, based on context and structure of the data sources, to improve the precision of the resulting XPath on the training data with minimal recall decrease. While this method generates richer XPaths compared to [20], and yields more precise queries in some cases, the resulting XPaths are still too specific and suffer from the same robustness issue. This results from using an alignment based algorithm which finds the most specific generalized XPath, keeping unnecessary features as long as they do not affect its recall. Zheng et al. [30] propose a record-level wrapper system. The generated wrappers include the complete tag-path from the root to record boundary nodes, making them sensitive to tag modifications. The major weakness of these techniques is related to the lack of flexibility of their generated XPath queries. These techniques work on pages from a single site, and generate XPath expressions that are strictly related to the structure of the pages from the site on top of which they are defined. Omari et al.[22] propose a method for generating XPath extraction queries for a family of websites that

contain the same type of information. They learn the logical representation of a website by utilizing the shared data instances among these sites, which allows them to identify commonality even when the structure is different.

Robustness There has been some work on the problem of XPath *robustness* to site changes [6, 15, 17], trying to pick the most robust XPath query for extracting a particular piece of information. The generalization applied by these techniques is based on alignment of XPaths, and picks a single query with a fixed generalization/precision tradeoff. In contrast, our approach uses decision-trees and forgiving XPaths to adjust precision dynamically.

Cohen et al. [5] propose an extension of XPath called XTPath. XTPath stores additional information from the DOM tree and uses recursive tree matching to fix XPath wrappers automatically when shifts in the tree structure happen. Instead of constructing a robust XPath, the method uses the XTPath whenever an XPath fails.

Pattern based techniques Several works [4, 2, 7, 16, 27] propose methods that use repeated pattern mining to discover and extract data records. Kayed et al. [12] propose a technique that works on a set of pages to automatically detect their shared schema and extract the data. Their method (called FivaTech) uses techniques such as *alignment* and *pattern mining* to construct a "fixed/variant pattern tree," which can be used to construct an extraction regular expression. DEPTA [29] uses partial tree alignment to align generalized nodes and extract their data. DeLa [28] automatically generate regular expression wrappers based on the page HTML-tag structures to extract data objects from the result pages. IEPAD [4] discovers repeated patterns in a document by coding it into a binary sequence and mining maximal repeated patterns. These patterns are then used for data extraction. Omari et al. [21] propose a method for separating webpages into layout code and data. Their method uses a tree alignment based algorithm to calculate the cost of the shared representation of two given sub-trees, then it uses the calculated shared representation cost to decide whether it should fold them or not in order to minimize the representation cost of the given DOM tree. The result of the folding process is a separation of the page to a layout-code component and a data component. Chang el al. [3] propose a system for page-level schema induction, and uses wrapper verification to extract data. Given a large amount of webpages, they use a subset of these pages to learn the schema in unsupervised manner, and then use scheme verification procedure to extract data from the rest of the pages.

Model based techniques Several approaches relying on Machine Learning algorithms were presented for data extraction [10, 8, 26, 13]. HAO et al. [10] present a method for data extraction from a group of sites. Their method is based on a classifier that is trained on a seed site using a set of predefined feature types. The classifier is then used as a base for identification and extraction of attribute instances in unseen sites. [8] foucses on detecting the boundaries of interesting entities in the text and treats information extraction as a classification problem. It uses an SVM classifier to identify the start and end tags of the data. Song et al. [26] propose a dynamic learning framework to extract structured data of various verticals from web pages without human effort. The technique is based on the observation that there is adequate information with similar layout structures among different web sites that can be learned for structured data extraction. In contrast to model based techniques, our method synthesizes a (standard) XPath query, that is human readable and easy to understand and modify, while maintaining the generalization ability and flexibility of these techniques. In addition, the use of forgiving XPath queries enables our technique to avoid the tradeoff that classifier based techniques have, between precision on seen sites and precision on unseen sites.

7 Conclusion

We have presented, and implemented, a novel approach for synthesizing a robust cross-site data extractor for a family of sites.

The cross-site extractor that we synthesize combines the benefits of generalization based on decision tree learning, with the efficient realization of a query as a forgiving XPath that can be directly and efficiently executed by browsers (and extraction tools).

As our experiments show, the synthesized extractor manages to remain precise for the training set, despite its generality. In addition, it can extract information from unseen pages and sites with a relatively high accuracy due to its novel forgiveness property, which allows it to *dynamically adjust its precision* for the web page at hand. Interestingly, our extractors outperform not only other pattern-based extractors, but also classifier-based extractors which are typically more suited for handling unseen sites. This is achieved while keeping the many benefits of XPath queries such as readability and efficient execution.

Acknowledgements

The research leading to these results has received funding from the European Union's - Seventh Framework Programme (FP7) under grant agreement no. 615688, ERC-COG-PRIME.

8 References

[1] ANTON, T. Xpath-wrapper induction by generalizing tree traversal patterns. In *Lernen, Wissensentdeckung und Adaptivitt (LWA) 2005, GI Workshops, Saarbrcken* (2005), pp. 126–133.

[2] ARASU, A., AND GARCIA-MOLINA, H. Extracting structured data from web pages. In *SIGMOD* (2003).

[3] CHANG, C.-H., CHEN, T.-S., CHEN, M.-C., AND DING, J.-L. Efficient page-level data extraction via schema induction and verification. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining* (2016), Springer, pp. 478–490.

[4] CHANG, C.-H., AND LUI, S.-C. IEPAD: information extraction based on pattern discovery. In *WWW* (2001).

[5] COHEN, J. P., DING, W., AND BAGHERJEIRAN, A. Semi-supervised web wrapper repair via recursive tree matching. *arXiv preprint arXiv:1505.01303* (2015).

[6] DALVI, N., BOHANNON, P., AND SHA, F. Robust web extraction: an approach based on a probabilistic tree-edit model. In *SIGMOD* (2009).

[7] DALVI, N., KUMAR, R., AND SOLIMAN, M. Automatic wrappers for large scale web extraction. *Proceedings of the VLDB Endowment 4*, 4 (2011), 219–230.

[8] FINN, A., AND KUSHMERICK, N. *Multi-level boundary classification for information extraction*. Springer, 2004.

[9] HALL, M., FRANK, E., HOLMES, G., PFAHRINGER, B., REUTEMANN, P., AND WITTEN, I. H. The weka data mining software: an update. *ACM SIGKDD explorations newsletter 11*, 1 (2009), 10–18.

[10] HAO, Q., CAI, R., PANG, Y., AND ZHANG, L. From one tree to a forest: a unified solution for structured web data extraction. In *SIGIR* (2011).

[11] JOHN, G. H., AND LANGLEY, P. Estimating continuous distributions in bayesian classifiers. In *Eleventh Conference on Uncertainty in Artificial Intelligence* (San Mateo, 1995), Morgan Kaufmann, pp. 338–345.

[12] KAYED, M., AND CHANG, C.-H. Fivatech: Page-level web data extraction from template pages. *Knowledge and Data Engineering, IEEE Transactions on 22*, 2 (2010), 249–263.

[13] KUSHMERICK, N. Wrapper induction: Efficiency and expressiveness. *Artificial Intelligence 118*, 1 (2000), 15–68.

[14] KUSHMERICK, N., WELD, D. S., AND DOORENBOS, R. B. Wrapper induction for information extraction. In *IJCAI'97*.

[15] LEOTTA, M., STOCCO, A., RICCA, F., AND TONELLA, P. Reducing web test cases aging by means of robust xpath locators. In *Software Reliability Engineering Workshops (ISSREW), 2014 IEEE International Symposium on* (2014), IEEE, pp. 449–454.

[16] LIU, B., GROSSMAN, R., AND ZHAI, Y. Mining data records in web pages. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining* (2003), ACM, pp. 601–606.

[17] LIU, D., WANG, X., LI, H., AND YAN, Z. Robust web extraction based on minimum cost script edit model. *Procedia Engineering 29* (2012), 1119–1125.

[18] MITCHELL, T. M. *Machine Learning*, 1 ed. McGraw-Hill, Inc., New York, NY, USA, 1997.

[19] NIELANDT, J., BRONSELAER, A., AND DE TRÉ, G. Predicate enrichment of aligned xpaths for wrapper induction. *Expert Systems with Applications* (2016).

[20] NIELANDT, J., DE MOL, R., BRONSELAER, A., AND DE TRÉ, G. Wrapper induction by xpath alignment. In *6th International Conference on Knowledge Discovery and Information Retrieval (KDIR 2014)* (2014), vol. 6, Science and Technology Publications, pp. 492–500.

[21] OMARI, A., KIMELFELD, B., YAHAV, E., AND SHOHAM, S. Lossless separation of web pages into layout code and data. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2016), KDD '16, ACM.

[22] OMARI, A., SHOHAM, S., AND YAHAV, E. Cross-supervised synthesis of web-crawlers. In *Proceedings of the 38th International Conference on Software Engineering* (2016), ACM, pp. 368–379.

[23] QUINLAN, J. Induction of decision trees. *Machine Learning 1*, 1, 81–106.

[24] QUINLAN, J. R. *C4. 5: programs for machine learning*. Elsevier, 2014.

[25] REIS, D. D. C., GOLGHER, P. B., SILVA, A. S., AND LAENDER, A. Automatic web news extraction using tree edit distance. In *WWW* (2004).

[26] SONG, D., WU, Y., LIAO, L., LI, L., AND SUN, F. A dynamic learning framework to thoroughly extract structured data from web pages without human efforts. In *Proceedings of the ACM SIGKDD Workshop on Mining Data Semantics* (2012), ACM, p. 9.

[27] THAMVISET, W., AND WONGTHANAVASU, S. Information extraction for deep web using repetitive subject pattern. *World Wide Web* (2013).

[28] WANG, J., AND LOCHOVSKY, F. H. Data extraction and label assignment for web databases. In *WWW* (2003).

[29] ZHAI, Y., AND LIU, B. Web data extraction based on partial tree alignment. In *WWW* (2005).

[30] ZHENG, S., SONG, R., WEN, J.-R., AND GILES, C. L. Efficient record-level wrapper induction. In *Proceedings of the 18th ACM conference on Information and knowledge management* (2009), ACM, pp. 47–56.