Regression testing for wrapper maintenance

Nicholas Kushmerick
Department of Computer Science, University College Dublin, Dublin 4, Ireland
nick@ucd.ie

Abstract

Recent work on Internet information integration assumes a library of *wrappers*, specialized information extraction procedures. Maintaining wrappers is difficult, because the formatting regularities on which they rely often change. The *wrapper verification* problem is to determine whether a wrapper is correct. Standard regression testing approaches are inappropriate, because both the formatting regularities and a site's underlying content may change. We introduce RAPTURE, a fully-implemented, domain-independent verification algorithm. RAPTURE uses well-motivated heuristics to compute the similarity between a wrapper's expected and observed output. Experiments with 27 actual Internet sites show a substantial performance improvement over standard regression testing.

Introduction

Systems that integrate heterogeneous information sources have recently received substantial research attention (*e.g.* (Wiederhold 1996; Knoblock *et al.* 1998; Levy *et al.* 1998)). A 'movie information' integrator, for example, might provide a single interface to the review, cast list, and schedule information available from dozens of Internet sites.

Such systems rely on a library of *wrappers*, specialized procedures for extracting the content from a particular site. For example, the site in Fig. 1 lists countries and their telephone country codes. The information extraction task is to identify the ⟨county, code⟩ pairs in this site's pages. The ccwrap wrapper does so by scanning for the delimiters <B>···</B> and <I>···</I>, which works because of a formatting regularity: **countries** are bold and *codes* are italic.

Scalability is the main challenge to building wrappers. While they are usually rather short programs, writing wrappers by hand is tedious and error-prone. Recently, there has been substantial progress on *wrapper induction*, techniques for automatically generating wrappers (Kushmerick, Weld, & Doorenbos 1997; Kushmerick 1997; Muslea, Minton, & Knoblock 1998; Hsu & Dung 1998).

However, this work ignores an important complication. Suppose the owners decide to 'remodel' the country/code site, so that pages look like p_d instead of p_a–p_c. The ccwrap wrapper fails for p_d, because the formatting regularities ccwrap exploits no longer hold.

Wrapper maintenance is the task of repairing a broken wrapper. For example, given p_d, a wrapper maintenance system would modify ccwrap to expect italic *countries* and bold **codes**.

Wrapper maintenance is our ultimate goal. In this paper, we define an important subproblem, the *wrapper verification* problem, and present RAPTURE, a fully-implemented, domain-independent, heuristic algorithm for solving this problem.

Wrapper verification involves determining whether a wrapper correctly processes a given page. Our approach is based on the black-box or *regression testing* paradigm (*e.g.* (Beizer 1995)): we give the wrapper a page for which the correct output is known, and check that the wrapper in fact generates this output.

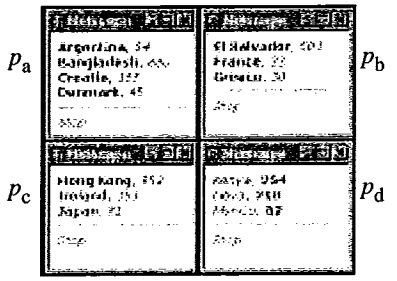
The simplest such regression tester is STRAWMAN. To verify a wrapper, STRAWMAN invokes it on the page returned in response to a given query, and also on an earlier page for the same query when the wrapper was known to be correct. STRAWMAN declares the wrapper verified if the two outputs are identical.

While STRAWMAN works for some Internet sites, it fails for most. STRAWMAN assumes that sites always return the same information for a fixed query. This assumption sometimes holds (*e.g.* a list of *historical* commodity prices), but is often violated (*e.g.* a site serving *today's* prices). Wrapper verification is thus complicated by two moving targets: the formatting regularities of the pages, and the site's underlying content.

In the remainder of this paper, we formalize the wrapper verification problem, describe RAPTURE,and empirically compare RAPTURE's performance to STRAWMAN.

Problem statement

Our work is based on an a simple yet generic information extraction task: a *site* accepts a *query*, returning a *page* in response. In our implementation, sites are HTTP servers, queries are CGI form values, and pages are HTML text. (While the our examples are posed in



p_a's HTML source (p_b's and p_c's are similar):

```
<HTML><BODY>
<B>Argentina</B>, <I>54</I><BR>
<B>Bangladesh</B>, <I>880</I><BR>
<B>Croatia</B>, <I>385</I><BR>
<B>Denmark</B>, <I>45</I><BR>
<HR><I>Stop</I></BODY></HTML>
```

p_d's HTML source:

```
<HTML><BODY>
<I>Kenya</I>, <B>254</B><BR>
<I>Libya</I>, <B>218</B><BR>
<I>Mexico</I>, <B>52</B><BR>
<HR><I>Stop</I></BODY></HTML>
```

The ccwrap wrapper is correct for p_a, p_b and p_c, but incorrect for p_d:

```
procedure ccwrap:
  while there are more occurrences of '<B>'
    skip to next '<B>'
    extract country through next '</B>'
    skip to next '<I>'
    extract code through next '</I>'
  return all extracted ⟨country,code⟩ pairs
```

Figure 1: An example Internet site listing countries and their telephone country codes, and the ccwrap wrapper.

terms of HTML, this is for simplicity only. Our techniques do not assume that pages are HTML.)

A *wrapper* is an information extraction procedure tailored to a particular site. A wrapper takes as input a page, and outputs a *label*, a representation of the page's content. We assume a standard *relational* data model: a page's label is a set of *tuples*, each a vector of K *attributes*. For example, $K = 2$ for page p_a, and its label is $\ell_a = \{\langle$`Argentina, 54`$\rangle, \langle$`Bangladesh, 880`$\rangle, \langle$`Croatia, 385`$\rangle, \langle$`Denmark, 45`$\rangle\}$.

For wrapper w and page p, we write $w(p) = \ell$ to indicate that w returns label ℓ when invoked on p.

As we are concerned with cases in which a wrapper's output is wrong, we say that a w is *correct* for p iff p's label is in fact $w(p)$. Thus, ccwrap is incorrect for p_d, because ccwrap$(p_d) = \{\langle$`254, Libya`$\rangle, \langle$`218, Mexico`$\rangle, \langle$`52, Stop`$\rangle\}$ instead of $\{\langle$`Hong Kong, 254`$\rangle, \langle$`Libya, 218`$\rangle, \langle$`Mexico, 52`$\rangle\}$.

Wrapper verification is the problem of determining whether a wrapper is correct. We assume access to a collection of previously verified pages for which the wrapper is known to be correct. We also assume that we know the query used to generate each verified page.

The *wrapper verification problem* is the following. The input is a wrapper w, page p, query q, sequence $L = \{\ell_1, \ldots, \ell_M\}$ of labels, and sequence $Q = \{q_1, \ldots, q_M\}$ of queries. The output should be TRUE if w is correct for p and FALSE otherwise.

The intent is that page p was retrieved using query q, L contains the verified labels, and the page to which label $\ell_i \in L$ corresponds was retrieved with $q_i \in Q$. While queries can be structured objects, RAPTURE treats them "atomically", only checking whether two queries are equal.

The RAPTURE algorithm

RAPTURE is a domain-independent, heuristic algoraithm for solving the wrapper verification problem. RAPTURE compares the verified labels with the label output by the wrapper being verified.

Specifically, RAPTURE compares the value of various numeric features of the strings comprising the label output by the wrapper. For example, the **word count** feature is 2 for `Great Britain`, and 1 for `Iraq`. RAPTURE computes the values of such features for each extracted attribute.[1]

These values are compared with those for the verified label attributes. The mean feature values are calculated, and then the probabilities that each extracted attribute's feature values agree with these values are calculated. Each such probability captures the strength of the evidence provided by a particular feature that a specific attribute was correctly extracted. Finally, the individual probabilities are combined, producing an overall probability that the wrapper is correct for the page.

We use upper-case letters (*e.g.* R) for random variables and lower-case (*e.g.* r) for values of such variables. μ_R is R's mean, and σ_R is R's standard deviation. Let R be normally distributed with parameters μ_R and σ_R; $\mathrm{P}[r; \mu_R; \sigma_R]$ is the probability that R equals r, and $\mathrm{P}[\leq r; \mu_R; \sigma_R]$ is the probability that R does not exceed r. These probabilities are taken to be the probability and cumulative density functions of the normal distribution: $\mathrm{P}[r; \mu_R; \sigma_R] = \frac{1}{\sigma_R\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{r-\mu_R}{\sigma_R})^2}$ and $\mathrm{P}[\leq r; \mu_R; \sigma_R] = \int_{-\infty}^{r} \mathrm{P}[r'; \mu_R; \sigma_R] \mathrm{d}r'$.

Example

We begin by demonstrating RAPTURE on the country/code example. Suppose ccwrap has been verified for pages p_a and p_b, and we want to verify ccwrap for pages p_c and p_d. Since ccwrap is in fact correct for p_c, RAPTURE$(p_c,$ ccwrap$, \{\ell_a, \ell_b\})$ should return TRUE; in contrast, RAPTURE$(p_d,$ ccwrap$, \{\ell_a, \ell_b\})$ should return FALSE. (We temporarily ignore the q and Q inputs to RAPTURE; we extend the algorithm below.)

Step 1: Number-of-tuple distribution parameters. RAPTURE assumes that the number of tuples in a label is described by the normally distributed random variable N. RAPTURE computes the distribution parameters μ_N and σ_N by treating the number of tuples n in p_a and p_b as samples of N (column 2 will be

[1] A wrapper's execution can be undefined. For example, ccwrap fails if it sees a <B> with no subsequent </B>, <I> or </I>. For simplicity, we do not discuss this scenario further, though our implementations of STRAWMAN and RAPTURE immediately reject failed wrappers.

explained in Step 5):

	n	$P[n;\mu_N;\sigma_N]$
p_a	4	0.44
p_b	3	0.44

μ_N=3.5
σ_N=0.71

Step 2: Feature value distribution parameters. RAPTURE examines the text fragments extracted from p_a and p_b, and computes the values of a set of features. These values are assumed to be normally distributed for each feature/attribute combination.

In this example, we consider two features: word count, and mean word length. For example, `El Salvador` has 2 words and mean word length 6. RAPTURE reasons about a unique random variable for each feature/attribute combination: C_1 is the word count of the first attribute (country), C_2 is the word count of the second attribute (code), U_1 is the country's mean word length, and U_2 is the code's mean word length.

RAPTURE uses p_a and p_b to estimate the 8 distribution parameters. For example, $\mu_{C_1} = 1.1$ indicates that country names usually contain a single word, while $\sigma_{U_2} = 0.52$ indicates that the mean word length of codes have relatively low variance (columns 3 and 5 will be explained in Step 5):

	country	c	$P[c;\mu_{C_1};\sigma_{C_1}]$	u	$P[u;\mu_{U_1};\sigma_{U_1}]$
p_a	`Argentina`	1	0.98	9	0.12
	`Bangladesh`	1	0.98	10	0.057
	`China`	1	0.98	5	0.12
	`Denmark`	1	0.98	7	0.22
p_b	`El Salvador`	2	0.081	6	0.19
	`France`	1	0.98	6	0.19
	`Greece`	1	0.98	6	0.19

μ_{C_1}=1.1 μ_{U_1}=7.0
σ_{C_1}=0.38 σ_{U_1}=1.8

	code	c	$P[c;\mu_{C_2};\sigma_{C_2}]$	u	$P[u;\mu_{U_2};\sigma_{U_2}]$
p_a	54	1	1.0	2	0.54
	880	1	1.0	3	0.42
	385	1	1.0	3	0.42
	32	1	1.0	2	0.54
p_b	503	1	1.0	3	0.42
	33	1	1.0	2	0.54
	30	1	1.0	2	0.54

μ_{C_2}=1.0 μ_{U_2}=2.4
σ_{C_2}=0.0 σ_{U2}=0.53

Step 3: Feature probabilities. RAPTURE uses the distribution parameters from Steps 1–2 to compare ccwrap's labels for p_a and p_b, with those for p_c and p_d. Recall that ccwrap is correct for p_c but incorrect for p_d.

RAPTURE begins by examining the number of tuples extracted by ccwrap: $n = 3$ for both pages. In Step 1, we estimated N's distribution parameters $\mu_N = 3.5$ and $\sigma_N = 0.71$. We can thus compute the probability of the observed number of tuples n for each page:

	n	$P[n;\mu_N;\sigma_N]$
p_c	3	0.44
p_d	3	0.44

RAPTURE then computes the feature values for the extracted fragments. Using the distribution parameters from Step 2, we compute the probability of each observed feature value. In the example, we compute 12 probabilities each for p_c and p_d: 1 per feature, for each of the 3 extracted fragments, for each of the 2 attributes. For example, $P[2;\mu_{C_1};\sigma_{C_1}] = 0.081$ indicates that a country name is somewhat unlikely to contain 2 words, while $P[4;\mu_{U_2};\sigma_{U_2}] = 9.9\times10^{-4}$ indicates that the mean word length of a code is rarely 4.

	country	c	$P[c;\mu_{C_1};\sigma_{C_1}]$	u	$P[u;\mu_{U_1};\sigma_{U_1}]$
p_c	`Hong Kong`	2	0.081	4	0.057
	`Ireland`	1	0.98	7	0.22
	`Japan`	1	0.98	5	0.12
p_d	254	1	0.98	3	0.020
	218	1	0.98	3	0.020
	52	1	0.98	2	5.1×10^{-3}

	code	c	$P[c;\mu_{C_2};\sigma_{C_2}]$	u	$P[u;\mu_{U_2};\sigma_{U_2}]$
p_c	852	1	1.0	3	0.42
	353	1	1.0	3	0.42
	81	1	1.0	2	0.54
p_d	`Libya`	1	1.0	5	7.0×10^{-6}
	`Mexico`	1	1.0	6	1.5×10^{-10}
	`Stop`	1	1.0	4	9.9×10^{-4}

Step 4: Verification probability. Each probability in Step 3 represents the evidence from one feature that a particular text fragment is correct. RAPTURE now combines this evidence conjunctively, deriving an overall *verification probability* that ccwrap is correct. RAPTURE assumes independence, and derives the verification probability by multiplying the conjuncts' probabilities. (Our experiments consider other assumptions.) The following table shows the verification probability v for each page (column 3 will be explained in Step 5):

	probabilities from Step 3	v	$P[\leq v;\mu_V;\sigma_V]$
p_c	.44, .081, .98, .98, .057, .22, .12, 1, 1, 1, .42, .42, .54	4.9×10^{-6}	0.26
p_d	.44, .98, .98, .98, .020, .020, 5.1×10^{-3}, 1, 1, 1, 7.0×10^{-6}, 1.5×10^{-10}, 9.9×10^{-4}	8.8×10^{-25}	0.18

Note that p_d has a low v because the mean word length of the countries is so small, and so large for the codes.

Step 5: Evaluating verification probability. As expected, ccwrap is much more likely to be correct for p_a than for p_d. To complete our analysis, we could simply compare such probabilities to a fixed threshold. However, since we assumed independence and normality, the calculated verification probabilities may deviate greatly from their 'true' values.

To address this problem, we treat each verification probability v as a sample of a normal random variable V. To calculate μ_V and σ_V, we repeat Step 4 for p_a and p_b:

	probabilities from Steps 1–2	v
p_a	.44, .98, .98, .98, .98, .12, .057, .12, .22, 1, 1, 1, 1, .54, .42, .42, .54	3.8×10^{-6}
p_b	.44, .081, .98, .98, .19, .19, .19, 1, 1, 1, .42, .54, .54	2.9×10^{-5}

$\mu_V = 1.6\times10^{-5}$
$\sigma_V = 1.8\times10^{-5}$

At this point, RAPTURE can compare the verification probabilities with those of p_c or p_d. The simplest approach is to return FALSE if $v < \mu_V$. However, we would

```
function RAPTURE(wrapper w, page p, label set L)
  if LABELPR(w(p), L) < τ return FALSE else return TRUE
function LABELPR(label ℓ, label set L)
  ⟨μ_V, σ_V⟩ ← VERIFPRPARAMS(L)
  return P[≤VERIFPR(ℓ, L); μ_V; σ_V]
function VERIFPRPARAMS(label set L)
  probs ← {VERIFPR(ℓ, L) | ℓ∈L}
  compute μ_V and σ_V from probs
  return ⟨μ_V, σ_V⟩
function VERIFPR(label ℓ, label set L)
  ⟨μ_N, σ_N⟩, {..., ⟨μ_{F_{i,k}}, σ_{F_{i,k}}⟩, ...} ← FEAPARAMS(L)
  q ← P[|ℓ|; μ_N; σ_N]
  probs ← {q} [a]
  for each feature f_i ∈ F
    for each attribute 1 ≤ k ≤ K
      for each tuple ⟨..., s_k, ...⟩ ∈ ℓ
        q ← P[f_i(s_k); μ_{F_{i,k}}; σ_{F_{i,k}}]
        probs ← probs ∪ {q}
  return A(probs) [b]
function FEAPARAMS(label set L)
  values ← {|ℓ| | ℓ∈L} [c]
  compute μ_N and σ_N from values
  for each feature f_i ∈ F
    for each attribute 1 ≤ k ≤ K
      values ← {f_i(s) | s in column k of a label in L}
      compute μ_{F_{i,k}} and σ_{F_{i,k}} from values
  return ⟨μ_N, σ_N⟩, {..., ⟨μ_{F_{i,k}}, σ_{F_{i,k}}⟩, ...}
```

Figure 2: The RAPTURE algorithm.

like to be able to adjust RAPTURE's 'pessimism', making it more or less likely to declare a wrapper correct. We thus use the cumulative normal density function to evaluate v. Specifically, the third column in Step 4 lists the probabilities that $V < v$, for p_c and p_d. RAPTURE then compares this probability to a threshold τ. For example, if $\tau = \frac{1}{4}$, RAPTURE returns TRUE for p_c and FALSE for p_d. Note that $\tau = \frac{1}{2}$ corresponds to the simple approach of returning FALSE if $v < \mu_V$.

Details

With the ideas underlying RAPTURE now in place, we list the RAPTURE algorithm in detail; see Fig. 2. The main subroutine is VERIFPR, which computes a label's verification probability v.

RAPTURE refers to three parameters: a **threshold** τ against which verification probabilities are compared; a **feature set** $\mathcal{F}$, where each feature is function from a string to a number; and a **dependency assumption** $\mathcal{A}$, a function from a set of numbers to a number.

Feature set $\mathcal{F}$

RAPTURE is parameterized by a *feature set* $\mathcal{F} = \{\ldots, f_i, \ldots\}$. Each feature f_i is a function from a string to a number. Ideally, features can be rapidly computed (so RAPTURE runs quickly) and are not domain specific (so RAPTURE can be applied without modification to new sites).

We used the following nine features in our experiments (numbers in parentheses are values for the string '20 Maple St.'): **digit density:** fraction of numeric characters ($\frac{2}{12} = 0.167$); **letter density**, fraction of letters ($\frac{7}{12} = 0.583$); **upper-case density**, fraction of upper-case letters ($\frac{2}{12} = 0.167$); **lower-case density** ($\frac{5}{12} = 0.417$); **punctuation density**, ($\frac{1}{12} = 0.083$); **HTML density**, fraction of < and > characters ($\frac{0}{12} = 0$); **length** (12); **word count** (3); and **mean word length** ($\frac{2+5+2}{3} = 3$).

Dependency assumption $\mathcal{A}$

Abstractly, RAPTURE reasons about E events $e_1, \ldots, e_E$. e_{27} might represent the event 'the number of words in the third extracted country is consistent with the countries extracted from the verified pages'. The algorithm has derived $\mathrm{P}[e_1], \ldots, \mathrm{P}[e_E]$, and must compute the probability that all feature values are consistent with the verified labels, $\mathrm{P}[\wedge_i e_i]$. RAPTURE uses the dependency assumption $\mathcal{A}$ to compute this probability; see Step 4 above and line [b] in Fig. 2.

Computing $\mathrm{P}[\wedge_i e_i]$ exactly requires knowledge of the dependencies between the e_i. In principle, this information could be derived for our domain, resulting in a set of conditional probabilities for exactly calculating $\mathrm{P}[\wedge_i e_i] = \mathrm{P}[e_1]\mathrm{P}[e_2|e_1]\cdots\mathrm{P}[e_E|e_{E-1}, \ldots, e_1]$. But as this analysis would be extremely cumbersome, we instead simply make assumptions about the dependencies of the domain. While the resulting probability can be inaccurate, our experiments demonstrate that these assumptions yield reasonable performance.

We have investigated three specific assumptions: the *independence*, *entailment*, and *equivalence* assumptions. Each corresponds to a particular function for computing $\mathrm{P}[\wedge_i e_i]$ from $\mathrm{P}[e_1], \ldots, \mathrm{P}[e_E]$.

Independence assumption: If the e_i are independent, then $\mathbf{P}[\wedge_i e_i] = \prod_i \mathbf{P}[e_i]$.

Entailment assumption: If there exists an $1 \le I \le E$ such that e_I logically entails every other e_i, then $\mathbf{P}[\wedge_i e_i] = \mathbf{min}_i\, \mathbf{P}[e_i]$. The entailment assumption presumes that one piece of evidence (*e.g.*, perhaps the number of extraced tuples) completely determines the rest.

Equivalence assumption: If the e_i are all logically equivalent, then $\mathbf{P}[\wedge_i e_i] = \mathbf{P}[e_1] = \cdots = \mathbf{P}[e_E]$. Under the equivalence assumption, any piece of evidence is as reliable as any other. This assumption can not be invoked directly, because the $\mathrm{P}[e_i]$ are often unequal. RAPTURE treats the $\mathrm{P}[e_i]$ as noisy samples of the 'true' value, which is estimated as the sample's geometric mean: $\mathbf{P}[\wedge_i e_i] = (\prod_i \mathbf{P}[e_i])^{1/E}$.

Using q and Q

In the presentation so far, we have ignored RAPTURE's q and Q inputs. Recall that Q is the sequence of queries used to generated the verified labels L, and q is the query from which input page p was generated.

RAPTURE uses Q and q to improve verification in a relatively simple manner. The intuition is that the number of tuples in p's label often depends on q. If the site lists people, for example, queries for `Jones` return more hits than for `Jablonsky`. If this dependency is ignored, RAPTURE will assess the evidence provided by the number of tuples ($\mathrm{P}[n; \mu_N, \sigma_N]$) incorrectly.

Extending RAPTURE to make use of this information is straightforward. Arguments Q and q are added to every function, and then line [c] in Fig. 2 is changed so that only verified labels with query q are used to estimate μ_N and σ_N: 'values $\leftarrow \{|\ell_i| \mid \ell_i \in L \wedge q_i = q\}$'.

Evaluation

Methodology. We tested RAPTURE on 27 actual Internet sites. We systematically varied $\mathcal{F}$, $\mathcal{A}$ and τ, and compared RAPTURE's performance with the STRAWMAN algorithm described earlier. Our experiment was designed to model the scenario in which the goal is to continually monitor whether a site's wrapper has changed.

The sites were chosen to be representative of the sort that the information integration community wants to wrap.[2] Fifteen queries were selected for each site. Each query is a keyword appropriate to the site. For example, ALTA's queries included `frog`, `dog`, `happy`, `apple` and `tomato`. While many sites allow complex queries, we do not think that focusing on keyword queries invalidates our verification results.

As shown in Fig. 3, the 27×15 queries were issued approximately every 3 days over 6 months (5–10/1998); in total, 23,416 pages were gathered.

We then generated a wrapper for every page using semi-automated techniques, extracting between $K = 2$ and $K = 8$ attributes. Like ccwrap in Fig. 1, the wrappers are instances of the LR wrapper classes (Kushmerick 1997). However, since we examine only the output of the wrappers, the choices of wrappers is immaterial.

These wrappers provide the 'gold standard' against which STRAWMAN's and RAPTURE's performance were judged. For a fixed site, call the sequence of P gathered pages $p_1, p_2, \ldots, p_P$. For each p_i, we stored the query q_i from which p_i was generated, and a wrapper w_i that is correct for p_i. We then invoked RAPTURE for each page in turn, using the previous page's wrapper: $\textsc{rapture}(p_i, w_{i-1}, q_i, \{w_j(p_j)\}_{j=1}^{P}, \{q_j\}_{j=1}^{P})$.

Performance metrics. RAPTURE should return TRUE iff the site's wrapper does not changes at p_i—*ie.* if $w_i = w_{i-1}$. We measure performance in terms of 2×2 matrix of integers:

[2] AltaVista (ALTA), Bible (BIBL), CD-Plus, Computer ESP, Cinemachine, Cost-of-living calculator (COLC), Corel stock photographs, Expedia currency calculator, Fortune-500 list (FOR5), Internet address finder, Irish Times, Lycos, Metacrawler, Monster Job Search, NewJour, CNET News, Rain or Shine, Shops.net, Time, Thrive, US Constitution (USCO), US News & World Report, US Patents, US Income Tax code (USTX), Virtual Garden, Webcrawler, and Yahoo people search.

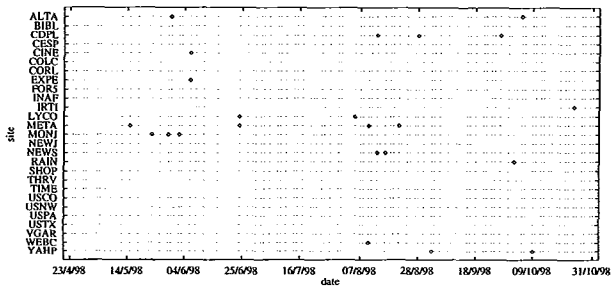


Figure 3: The experimental data: dots are fetched pages (15 per site), and diamonds are wrapper changes.

	$w_i = w_{i-1}$	$w_i \neq w_{i-1}$
predict TRUE	n_1	n_2
predict FALSE	n_3	n_4

Perfect performance yields $n_2 = n_3 = 0$. Several performance metrics are derived from this matrix (higher values indicate better performance): **accuracy** $= \frac{n_1+n_4}{n_1+n_2+n_3+n_4}$ measures overall performance; **precision** $= \frac{n_1}{n_1+n_2}$ is the fraction of correct TRUE responses; **recall** $= \frac{n_1}{n_1+n_3}$ is the fraction of unchanged wrappers for which the system responds TRUE; and $\mathbf{F} = \frac{2 \cdot \text{recall} \cdot \text{precision}}{\text{recall}+\text{precision}}$ combines precision and recall into a single metric.

Results. The wrappers for many sites (56%) did not change at all; see Fig. 3. A total of 23 wrapper changes were seen over the 6 months, for an average of 0.85 per site. For the 44% of sites with at least one wrapper change, the average number of changes was 1.9, and the maximum was 4.

STRAWMAN performs perfectly for only 19% of the sites (BIBL, COLC, FOR5, USCO and USTX). Thus 81% of the sites can not be handled by standard regression testing. Note that STRAWMAN's overall accuracy is rather high (64%) because so few wrappers changed; indeed, a verifier that always returns TRUE has accuracy exceeding 99% (though it will have $F = 0$).

Fig. 4 shows RAPTURE's performance for $\tau = \frac{1}{2}$, and for 22 settings for $\mathcal{F}$ and $\mathcal{A}$. RAPTURE outperforms STRAWMAN for every parameter setting, with an average gain of 30% in accuracy and 16% in F.

Settings (1–3) compare the three dependency assumptions. Since **equivalence** generally performs best, setting (4-21) use **equivalence** as well.

Settings (4–12) examine the importance of each feature: each setting uses just a single feature. **HTML density** appears to significantly outperforms the others, because the attributes extracted by an incorrect wrapper are likely to contain HTML tags.

Settings (13–21) examine whether any feature harms performance: each setting uses all *except* a particular feature. None of the features stands out as being particularly damaging. (4–12) generally outperform (1–3) and (13–21), suggesting that fewer features are better.

	assumption $\mathcal{A}$	digit density feature	letter density feature	UC density feature	LC density feature	punc density feature	HTML density feature	length feature	word count feature	mean word len feature	Accuracy × 100	F × 100	
1	equiv	√	√	√	√	√	√	√	√	√	82	90	
2	entail	√	√	√	√	√	√	√	√	√	79	88	
3	indep	√	√	√	√	√	√	√	√	√	65	79	
4	equiv	√	×	×	×	×	×	×	×	×	89	94	
5	equiv	×	√	×	×	×	×	×	×	×	87	93	
6	equiv	×	×	√	×	×	×	×	×	×	85	92	
7	equiv	×	×	×	√	×	×	×	×	×	84	91	
8	equiv	×	×	×	×	√	×	×	×	×	86	93	
9	equiv	×	×	×	×	×	√	×	×	×	97	98	
10	equiv	×	×	×	×	×	×	√	×	×	79	88	
11	equiv	×	×	×	×	×	×	×	√	×	81	90	
12	equiv	×	×	×	×	×	×	×	×	√	80	89	
13	equiv	×	√	√	√	√	√	√	√	√	81	90	
14	equiv	√	×	√	√	√	√	√	√	√	81	90	
15	equiv	√	√	×	√	√	√	√	√	√	81	90	
16	equiv	√	√	√	×	√	√	√	√	√	81	90	
17	equiv	√	√	√	√	×	√	√	√	√	81	90	
18	equiv	√	√	√	√	√	×	√	√	√	81	90	
19	equiv	√	√	√	√	√	√	×	√	√	84	91	
20	equiv	√	√	√	√	√	√	√	×	√	81	89	
21	equiv	√	√	√	√	√	√	√	√	×	81	90	
22	indep	×	×	×	×	×	√	×	×	×	>99	>99	*
	STRAWMAN										64	78	

Figure 4: RAPTURE's performance for several settings of $\mathcal{F}$ and $\mathcal{A}$, and $\tau = \frac{1}{2}$.

Setting (22) shows the best performing parameters: the **independence** assumption, and just the **HTML density** feature. (22) is marked '*' to stress two additional changes to RAPTURE: the number-of-tuples information is ignored (line [a] in Fig. 2 is skipped); and FEAPARAMS calculates the probability of the mean feature value across all tuples for each attribute (instead of a separate probability for each tuple).

While these modifications are not well motivated, the results are impressive: this modified version of RAPTURE performs nearly perfectly, with just 3 mistakes over 23,416 predictions, and gains over STRAWMAN of 56% in accuracy and 28% in F. Furthermore, performance according to two additional metrics—$\frac{n_4}{n_4+n_3}$, the fraction of correct FALSE responses, and $\frac{n_4}{n_4+n_2}$, the fraction of noticed wrapper changes—is 344-fold better than STRAWMAN.

Finally, we can change RAPTURE's "pessimism" by varying τ. For example, in setting (1), varying τ from 1 to 0 decreases recall from 1 to 0.003, while precision (always very high) increases from 0.999 to 1.

Conclusions

We are motivated by the task of Internet information extraction, which is relevant to a wide variety of information-management applications. Data-exchange standards such as XML will simplify this process, but they are not widely used. Furthermore, they do not entirely solve the problem, since they force the data consumer to accept the producer's ontological decisions. We conclude that the thorny problems of wrapper construction and maintenance will remain for some time.

We have introduced the wrapper verification problem, and presented RAPTURE, a fully-implemented, domain-independent, heuristic solution. Verification is difficult because at many Internet sites, both the formatting regularities used by wrappers, and the underlying content, can change. Standard regression testing approaches (*e.g.* see (Beizer 1995)) are inapplicable, as they assume the underlying content is static.

RAPTURE uses a set of syntactic features to compute the similarity between a wrapper's output and the output for pages for which the wrapper is known to be correct. RAPTURE combines the similarities to derive an overall probability that the wrapper is correct. Our experiments demonstrate significant performance improvements over standard regression testing.

We are currently exploring several extensions to our techniques. First, the features were selected in an *ad-hoc* manner; we are investigating additional features. Second, we are identifying additional probabilistic assumptions; while they were effective, the three assumptions examined here are intuitively unsatisfactory. Third, we assumed normal distributions throughout, but have not verified this assumption, and some of the statistical computations are not well founded; while the normality assumption delivers reasonable performance, we are exploring alternatives, such as the Gamma distribution, which may model our data more accurately. Finally, we are generalizing our techniques to object-oriented, semi-structured, and other non-relational data models.

References

Beizer, B. 1995. *Black-Box Testing.* John Wiley & Sons.

Hsu, C., and Dung, M. 1998. Generating finite-state transducers for semistructured data extraction from the web. *J. Information Systems* 23(8).

Knoblock, A.; Levy, A.; Duschka, O.; Florescu, D.; and Kushmerick, N., eds. 1998. *Proc. 1998 Workshop on AI and Information Integration.* AAAI Press.

Kushmerick, N.; Weld, D.; and Doorenbos, R. 1997. Wrapper Induction for Information Extraction. In *Proc. 15th Int. Joint Conf. AI*, 729–35.

Kushmerick, N. 1997. *Wrapper Induction for Information Extraction.* Ph.D. Dissertation, Univ. of Washington.

Levy, A.; Knoblock, C.; Minton, S.; and Cohen, W. 1998. Trends and controversies: Information integration. *IEEE Intelligent Systems* 13(5).

Muslea, I.; Minton, S.; and Knoblock, C. 1998. Wrapper Induction for Semi-structured, Web-based Information Sources. In *Proc. Conf. Automatic Learning & Discovery.*

Wiederhold, G. 1996. *Intelligent Information Integration.* Kluwer.