Using Grammatical Inference to Automate Information Extraction from the Web

Theodore W. Hong and Keith L. Clark

Department of Computing
Imperial College of Science, Technology, and Medicine
180 Queen's Gate, London SW7 2BZ
United Kingdom
{twh1,klc}@doc.ic.ac.uk

Abstract. The World-Wide Web contains a wealth of semistructured information sources that often give partial/overlapping views on the same domains, such as real estate listings or book prices. These partial sources could be used more effectively if integrated into a single view; however, since they are typically formatted in diverse ways for human viewing, extracting their data for integration is a difficult challenge. Existing learning systems for this task generally use hardcoded *ad hoc* heuristics, are restricted in the domains and structures they can recognize, and/or require manual training. We describe a principled method for automatically generating extraction wrappers using grammatical inference that can recognize general structures and does not rely on manually-labelled examples. Domain-specific knowledge is explicitly separated out in the form of declarative rules. The method is demonstrated in a test setting by extracting real estate listings from web pages and integrating them into an interactive data visualization tool based on dynamic queries.

1 Introduction

The World-Wide Web contains a wealth of information resources, many of which can be considered as semistructured[1] data sources: that is, sources containing data that is fielded but not constrained by a global schema. For example, documents such as product catalogs, staff directories, and classified advertisement listings fall into this category. Often, multiple sources provide partial or overlapping views on the same underlying domain. As a result, there has been much interest in trying to combine and cross-reference disparate data sources into a single integrated view.

Parsing web pages for information extraction is a significant obstacle, however. Although the markup formatting of web sources provides some hints about their record and field structure, this structure is also obscured by the presentation aspects of formatting intended for human viewing and the wide variation in formats from site to site. Manually constructing extraction wrappers is tedious and time-consuming, because of the large number of sites to be covered and the need to keep up-to-date with frequent formatting changes.

We propose the use of grammatical inference to automate the construction of wrappers and facilitate the process of information extraction. Grammatical inference is a subfield of machine learning concerned with inferring formal descriptions of sets from examples. One application is the inference of formal grammars as generalized structural descriptions for documents.

By applying an inference algorithm to a training sample of web pages from a given site, we can learn a grammar describing their format structure. Using domain-specific knowledge encoded in declarative rules, we can identify productions corresponding to records and fields. The grammar can then be compiled into a wrapper which extracts data from those pages. Since the data pages on a given website typically follow a common site format, particularly if they are dynamically created from scripts, such wrappers should be able to operate on the rest of the pages as well.

This process can be largely automated, making it easy to re-generate wrappers when sites change formatting. Although others have explored the use of machine learning for wrapper creation, previous systems have generally relied on hardcoded *ad hoc* heuristics or the manual labelling of examples, and/or have been restricted in the domains and structures they can recognize. Our method is more principled, based on an objective method for inferring structure and a description language (context-free grammars) of high expressive power. We avoid manual intervention as far as possible and explicitly separate out domain-specific knowledge using declarative rules. These characteristics should make our system easier to use and more broadly applicable in different domains.

Taking the real estate domain as an example, we demonstrate the use of our approach to extract property listings from a set of mock web pages. We also briefly show an interactive data visualization tool based on dynamic queries for exploring the resulting high-dimensional data space.

The rest of this paper is organized as follows: in Sect. 2, we introduce the formal background to grammatical inference before describing our inference algorithm in Sect. 3. We then apply the algorithm to the real estate domain in Sect. 4. Related work is discussed in Sect. 5, and finally Sect. 6 gives conclusions and future work.

2 Grammatical Inference

Grammatical inference[17] is a class of inductive inference in which the target is a formal language (a set of strings over some alphabet Σ) and the hypothesis space is some family of grammars. The objective is to infer a consistent grammar for the unknown target language, given a finite set of examples.

The classical approach to grammatical inference was first given by Gold[13], who introduced the notion of *identification in the limit.* This notion is concerned with the limiting behavior of an inference algorithm on an infinite sequence of examples. Formally, a *complete presentation* of a language L is an infinite sequence of ordered pairs (w, l) in $\Sigma^* \times \{0,1\}$, where $l = 1$ if $l \in L$ and 0 otherwise, and every string $w \in \Sigma^*$ appears at least once. If an inference method

$\mathcal{M}$ is run on larger and larger initial segments of a complete presentation, it will generate an infinite sequence of guesses g_1, g_2, g_3, etc. $\mathcal{M}$ is said to *identify L in the limit* if there exists some number n such that all of the guesses g_i are the same for $i \geq n$, and g_n is equivalent to L.

This approach is not directly applicable to the web document task, since only positive examples are available (these being the actual documents existing at a site). Gold showed that any class of languages containing all the finite languages and at least one infinite language cannot be identified in the limit from only positive examples without negative ones. For example, the classes of regular and context-free languages both fit this criterion. The problem is that the task is under-constrained. Given only positive examples, the inferencer has no basis for choosing among hypotheses which are too general (e.g. the language consisting of all strings), too specific (e.g. the language consisting of exactly the examples seen so far), or somewhere in between.

3 Inference Algorithm

We approach the inference problem differently as a search for the *simplest* grammar which has a consistent *fit* with the provided sample, on the assumption that simple grammars are more likely to convey meaningful structure. We introduce a learning bias to constrain the search by starting from a specialized grammar which has high fit but low simplicity, and applying various transformations to generalize and simplify it while retaining fit. To guide this process, we take the set of *stochastic context-free grammars* as the hypothesis space and define a complexity function on it in terms of description length. Stochastic context-free grammars[4] are context-free grammars with probabilities attached to their productions. The probabilities aid inference by providing additional information about the relative weight of alternative productions—for example, given two alternatives for some nonterminal, are both equally important or is one likely to be just noise? This information is useful for assessing relative complexity and performing simplifications or extracting data later on.

3.1 Measuring Grammar Complexity

Let G be a stochastic context-free grammar with productions and associated probabilities given by:

$$\begin{array}{ll} X_1 \rightarrow w_{11} \mid w_{12} \mid \ldots \mid w_{1,m_1} & [P_{11}, P_{12}, \ldots, P_{1,m_1}] \\ X_2 \rightarrow w_{21} \mid w_{22} \mid \ldots \mid w_{2,m_2} & [P_{21}, P_{22}, \ldots, P_{2,m_2}] \\ \quad\vdots & \\ X_n \rightarrow w_{n1} \mid w_{n2} \mid \ldots \mid w_{n,m_n} & [P_{n1}, P_{n2}, \ldots, P_{n,m_n}] \ , \end{array} \tag{1}$$

where the X_i are nonterminals, the w_{ij} are alternatives, and the P_{ij} are the probabilities associated to those alternatives (i.e. $\sum_{j=1}^{m_n} P_{ij} = 1$ for each i).

Following Cook *et al.*[7], we define the complexity $C(G)$ as:

$$C(G) = \sum_{i=1}^{n} \sum_{j=1}^{m_n} - \log P_{ij} + c(w_{ij}) \tag{2}$$

where $c(w_{ij})$ is a second complexity function on the w_{ij} strings. This definition has the intuitively desirable properties that the complexity of G is the sum of the complexities of its productions and that the complexity of a production is the sum of the complexities of its alternatives. The complexity of an alternative has two components, the information-theoretic information content of its probability and the complexity of the string produced. Finally, the complexity of a string w is a function of its length and the proportions of distinct symbols in it:

$$c(w) = (K+1)\log(K+1) - \sum_{i=1}^{r} k_i \log k_i \tag{3}$$

where w has length K and contains r distinct symbols each occurring k_1, k_2, $\ldots$, k_r times, respectively. Longer and more varied strings are rated more complex.

3.2 Inference as Search

We can formulate the goal of looking for the simplest consistent grammar as a search in the space of grammars where the cost function is the complexity function C. Our starting point is the overspecific grammar that simply generates the training set with perfect fit:

$$S \to w_1 \mid w_2 \mid \ldots \mid w_m \qquad [P_1, P_2, \ldots, P_m] \ , \tag{4}$$

where $w_1 \ldots w_m$ are the strings occurring in the set and $P_1 \ldots P_m$ are their relative frequencies. If all the strings are different, then the P_i will all be equal to $1/m$; however, the P_i may vary if some strings appear more than once in the set. This initial grammar will generally have very high complexity.

We can then perform a search by considering various transformation steps which might lower the complexity and generalize the grammar while retaining good fit. Some of the transformations used (again following [7]) are:

1. Substitution: If a substring s occurs multiple times in different alternatives (e.g. in the grammar $X_1 \to asb$, $X_2 \to csd$), create a new rule $Y \to s$ and replace all occurrences of s by Y's. This transformation helps identify subunits of structure. For example, when applied to the productions "John is eating cake" and "Mary is eating bread," it will separate "is eating" into another rule.
2. Disjunction: If two substrings s and t occur in similar contexts (e.g. in the grammar $X_1 \to asb$, $X_2 \to atb$), create a new rule $Y \to s \mid t$ and replace all occurrences of s and t by Y's. This transformation introduces generalization based on context. For example, when applied to the productions "John throws baseballs" and "John catches baseballs," it will propose "throws" and "catches" as alternatives for the same production.

3. Expansion: Remove a rule $Y \rightarrow s \mid t \mid \ldots \mid v$ by replacing every alternative that mentions Y with a set of alternatives in which Y is replaced with s, t, etc. This can reverse previous substitutions and disjunctions later on.
4. Truncation: Remove alternatives having very low probability and redistribute their probability among the remaining alternatives. This can be used to remove noise below some threshold.
5. Normalization: Merge redundant alternatives (e.g. $X \rightarrow s \mid s$) and drop productions that are inaccessible (cannot be reached from the start symbol) or blocking (result in some nonterminal that cannot be rewritten). This is often necessary to "clean up" grammars to show the full extent of simplification resulting from another transformation.

Other variations on these transformations are also considered. For practical reasons, since the branching factor of possible search steps can be very large (sometimes exceeding 100), we perform searching using a greedy deterministic hill-climbing strategy. Simulated annealing is another possibility we are examining.

3.3 Example: Parenthesis Expressions

To demonstrate the algorithm, we consider the language of balanced parenthesis strings. Take the set of all such strings up to length 6 as the training set, with frequencies as shown (to be justified later):

$$\begin{array}{l}\text{sample} = \{(),\ ()(),\ (()),\ ()()(),\ ()(()),\ (())(),\ (()()),\ ((()))\} \\ \qquad [0.5, 0.125, 0.125, 0.0625, 0.03125, 0.03125, 0.0625, 0.0625]\ .\end{array} \tag{5}$$

The initial grammar, with complexity 99.68, is:

$$\begin{array}{l}S \rightarrow () \mid ()() \mid (()) \mid ()()() \mid ()(()) \mid (())() \mid (()()) \mid ((())) \\ \qquad [0.5, 0.125, 0.125, 0.0625, 0.03125, 0.03125, 0.0625, 0.0625]\ .\end{array} \tag{6}$$

Ten substrings are candidates for substitution: (),)(, ((,)), ()(,)(), ((), ()), ()(), and (()). The greatest reduction in complexity is obtained by substituting on (). Since it already appears as an alternative for S, we simply substitute S for () everywhere. The resulting grammar has complexity 88.09:

$$\begin{array}{l}S \rightarrow () \mid SS \mid (S) \mid SSS \mid S(S) \mid (S)S \mid (SS) \mid ((S)) \\ \qquad [0.5, 0.125, 0.125, 0.0625, 0.03125, 0.03125, 0.0625, 0.0625]\ ,\end{array} \tag{7}$$

Now the repeated substrings are SS, $(S$, $S)$, and (S). Choosing (S) gives:

$$\begin{array}{l}S \rightarrow () \mid SS \mid (S) \mid SSS \mid SS \mid SS \mid (SS) \mid (S) \\ \qquad [0.5, 0.125, 0.125, 0.0625, 0.03125, 0.03125, 0.0625, 0.0625]\ .\end{array} \tag{8}$$

After normalizing by merging redundant alternatives and summing their associated probabilities, we obtain a grammar with complexity 42.19:

$$\begin{array}{l}S \rightarrow () \mid SS \mid (S) \mid SSS \mid (SS) \\ \qquad [0.5, 0.1875, 0.1875, 0.0625, 0.0625]\ .\end{array} \tag{9}$$

The final set of repeated substrings are SS, $(S$, and $S)$, of which SS lowers complexity the most. This gives:

$$\begin{array}{l} S \rightarrow () \mid SS \mid (S) \mid SS \mid (S) \\ \qquad [0.5, 0.1875, 0.1875, 0.0625, 0.0625] \ , \end{array} \tag{10}$$

which after normalizing is the usual grammar for the parenthesis language:

$$\begin{array}{l} S \ \rightarrow \ () \mid SS \mid (S) \\ \qquad [0.5, 0.25, 0.25] \ . \end{array} \tag{11}$$

The final complexity is 20.51. Notice that if these probabilities are used to generate a set of strings up to length 6, we recover the string frequencies in the original sample. In this example, only substitution and normalization operations were used, but in general other transformations may be needed as well.

4 Information Extraction

We tested our algorithm on a set of mock web pages containing London real estate listings. The pages all followed the same general layout (see Figs. 1 and 2) but contained varying numbers of listings on each page, some containing pictures of the described property and some without. The set of pages was taken as the training set and the algorithm attempted to construct a suitable description from which extraction wrappers could be generated.

4.1 Grammatical Inference Phase

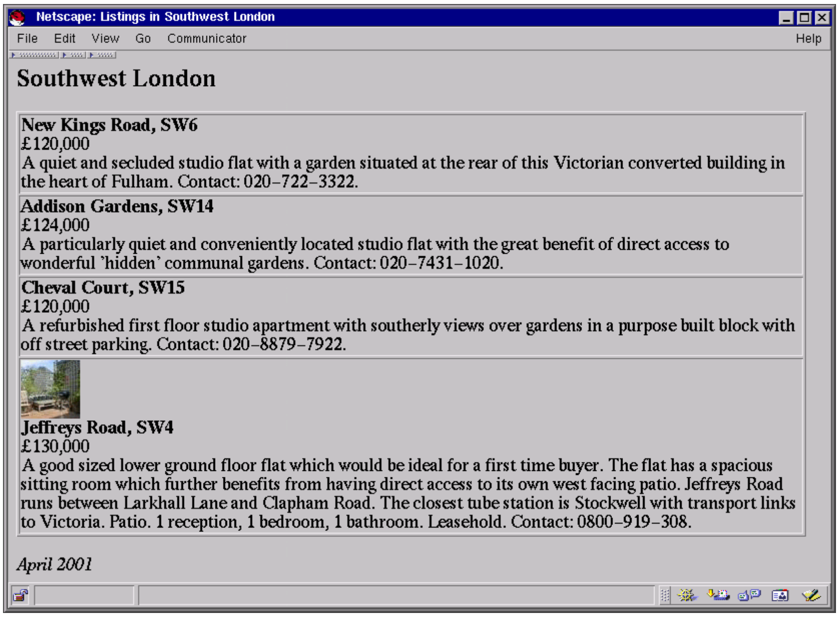
The web pages in the training set were first converted to abstract strings over the alphabet {HTML tag types} ∪ {`text`}. This was done by discarding HTML attributes from the tags encountered, so that tags of the same type (e.g. anchor start tags) would be treated as the same alphabet symbol (e.g. `a`). Free text occurring between tags was converted to the symbol `text`. In using this transformation, we assume that structure is mainly present at the tag type level and focus on that level by ignoring variations in text and attributes (e.g. `href` values). For example, two contact links:

```
<hr><a href="mailto:sales@a.com">A-1 Realtors</a>
<hr><a href="mailto:help@b.com">Bee Estate Agents</a>
```

would both be transformed to the same abstract string, `hr a text /a`.

Each page string then became an alternative in the initial grammar, which had perfect fit but high complexity (1056.32):

$$\begin{array}{ll} & \texttt{html head...table tr td b text /b br text.../html} \\ S \rightarrow & \mid \texttt{html head...table tr td img b text /b br.../html} \\ & \mid \textit{etc.} \end{array} \tag{12}$$


Netscape: Listings in Southwest London
File Edit View Go Communicator Help

Southwest London

New Kings Road, SW6
£120,000
A quiet and secluded studio flat with a garden situated at the rear of this Victorian converted building in the heart of Fulham. Contact: 020-722-3322.

Addison Gardens, SW14
£124,000
A particularly quiet and conveniently located studio flat with the great benefit of direct access to wonderful 'hidden' communal gardens. Contact: 020-7431-1020.

Cheval Court, SW15
£120,000
A refurbished first floor studio apartment with southerly views over gardens in a purpose built block with off street parking. Contact: 020-8879-7922.

Jeffreys Road, SW4
£130,000
A good sized lower ground floor flat which would be ideal for a first time buyer. The flat has a spacious sitting room which further benefits from having direct access to its own west facing patio. Jeffreys Road runs between Larkhall Lane and Clapham Road. The closest tube station is Stockwell with transport links to Victoria. Patio. 1 reception, 1 bedroom, 1 bathroom. Leasehold. Contact: 0800-919-308.

April 2001

Fig. 1. A sample real estate listing page

```
<html><head>
<title>Listings in Southwest London</title>
</head>
<body>
<h1>Southwest London</h1>
<table border=1 width=100%>
<tr><td><b>New Kings Road, SW6</b><br>
&pound;120,000<br>
A quiet and secluded studio flat with a garden situated at
the rear of this Victorian converted building in the heart
of Fulham. Contact: 020-722-3322.
</td></tr>...
```

Fig. 2. Part of the HTML source for Fig. 1

The inference algorithm ran in five seconds on a Pentium 233 and examined 386 candidate grammars, lowering the complexity to a value of 119.19 (see Fig. 3 for the quality curve). The final grammar was:

$$\begin{aligned} S &\rightarrow \texttt{html head title text /title /head body h1 text /h1} \\ &\quad\;\texttt{table } T \texttt{ /table p address text /address /body /html} \\ T &\rightarrow TT \mid \texttt{tr td } U \texttt{ b text /b br text br text /td /tr} \\ U &\rightarrow \; e \mid \texttt{img br} \;. \end{aligned} \tag{13}$$

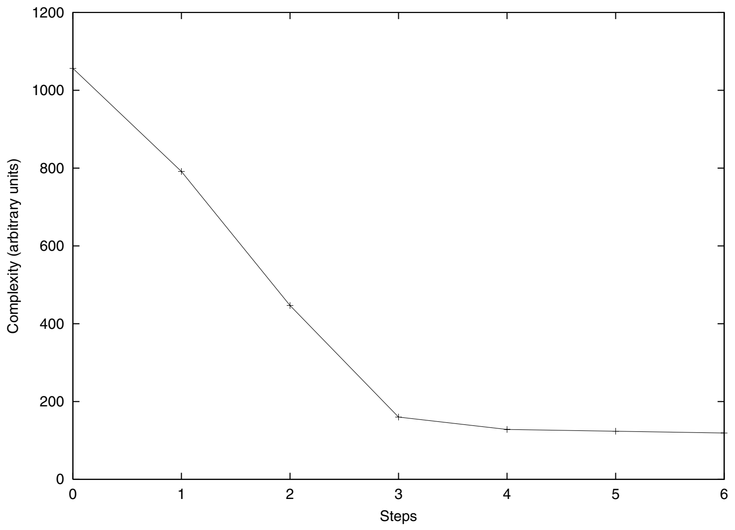


Fig. 3. Quality curve for the real estate grammar search

We can interpret this structure as follows. The start symbol S represents a complete page. A page begins with a fixed header, followed by one or more occurrences of T, each of which represents a single listing. A listing consists of a table row of data optionally containing an image U. Finally, the page is terminated by a fixed trailer.

In this process we have not made use of the DTD (document type definition) specification which defines the HTML language. Since by definition the HTML DTD is a general structure describing all HTML documents, it is not useful as a close fit to any particular set of pages. However, taking account of constraints from the DTD during transformations (for example, to keep blocks of elements from being split) might be a useful refinement to the algorithm. Alternately, a different approach altogether might be to start from the general HTML DTD and specialize it to the training set, rather than starting from a specific grammar and generalizing.

4.2 Domain-Specific Phase

The grammatical inference phase performs a coarse segmentation of the page into units of varying sizes, corresponding to different nonterminals. To complete the segmentation, we need to apply domain-specific knowledge to determine which units correspond to records and to segment the fields within records.

Domain knowledge is expressed as declarative information extraction rules for domain fields. Each rule consists of a field name and type plus a regular expression defining the context in which that field might appear. Rules are executed by applying the regular expression to a chunk of text. If a match is found, a specified portion of the matching text is extracted as the value of the field. Disjunctions can be used to define multiple contexts for a field. For example, in the real estate domain we might have a default set of rules such as the following:

```
number price     = (&pound;|&#163;|£){[0-9]+}
string telephone = {[0-9]+-[0-9]+-[0-9]+}
boolean garden   = garden|yard
```

The first rule says that a price looks like some form of pound sign followed by a number. The part of the match delimited by braces is returned as the field value. The second declares a telephone number as a string of digits interspersed with dashes, all of which becomes the field value. The third defines a boolean attribute which is true if either of the strings "garden" or "yard" is present.

These rules are then applied to the units discovered by the grammatical inference phase. More precisely, the following procedure is used. Parse the training pages according to the inferred grammar. For each occurrence of a nonterminal, collect all of the text appearing below it in the parse tree into an associated chunk. For the page shown in Fig. 1, the nonterminal S would be associated with one chunk containing all of the text on the page; T would be associated with four chunks, each containing the text of one listing; and U would be associated with four chunks containing no text.

Now apply the information extraction rules to each chunk. If a chunk yields multiple matches for several rules, as S does, it probably contains more than one record. However, if few or no rules match, as in U, the chunk is probably smaller than a record. The nonterminal matching the most rules without duplicate matches is assumed to correspond to a record—in this case, T.

4.3 Wrapper Generation Phase

Having identified the nonterminal T as corresponding to a listing record, we can now compile the grammar into a wrapper that extracts records from pages as chunks of text and applies domain rules to extract typed fields from those records. At this point, the user can manually add additional site-specific rules for fine-tuning. These rules may be optionally qualified by a piece number to restrict the match range to a particular piece (i.e. the section of text corresponding to a specific `text` symbol) within a chunk. For example, the following rules:

```
string address     = 1 : {.*},
string description = 3
```

specify that the address is the part of the first piece appearing before a comma, and that the description is the entire content of the third piece.

Running the wrapper on the sample page yields the records shown in Table 1. Once wrappers for all of the data sources to be used have been generated, the system can extract records from each and integrate the resulting data into a combined database. If a partial database is already available, it may be of use in helping to identify domain fields and formulate extraction rules for them.

Table 1. Partial listing of extracted records

Address	**Price**	**Garden**	**Description**
New Kings Road	120,000	yes	A quiet and secluded studio flat...
Addison Gardens	124,000	yes	A particularly quiet and convenient...
Cheval Court	120,000	yes	A refurbished first floor studio...
Jeffreys Road	130,000	no	A good sized lower ground floor...

Note that complete integration may not be possible, by nature of the data's origin in multiple collections of semistructured text. Some extraction rules may fail on some records, and not all fields may be present on all sites in the first place or even present in all records within a site. However, a mostly-complete overview can still be of significant value.

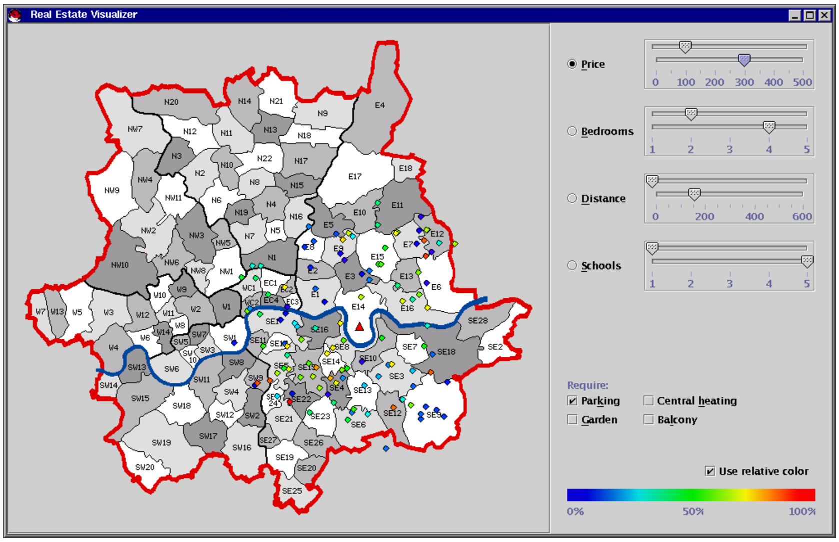
As a final step, this database can then be used as input into a data mining or information integration system. See for example [14], which describes an interactive real estate visualization system based on dynamic queries (see Fig. 4 for a screenshot). In this system, sliders continuously set selection criteria for properties (shown as color-coded points on a map) and can be used to naturally and rapidly explore the data space.

Although these preliminary results are qualitatively encouraging, more rigorous testing remains to be done to quantify performance on real-world data in terms of recognition rates, etc. Further work is also necessary to determine how robust the method is under different conditions and whether it might get stuck in local optima (switching to simulated annealing may be useful) or have difficulty identifying records properly.

5 Related Work

Ahonen[2] has used grammatical inference to generate structural descriptions for tagged SGML documents such as dictionaries and textbooks, while Freitag[10] explored the use of grammatical inference to find field boundaries in free text. A large amount of work has been done on developing inference algorithms; [17] presents a useful overview.

Work on integrating data from multiple websites has been carried out by a number of researchers. One of the first, Krulwich's BargainFinder[15], was able to scan product listings and prices from a set of on-line web stores and extract them into a unified ordered table. However, it was based entirely on hand-coded wrappers tailored specifically to each source site. ShopBot[9] went a step further by using various *ad hoc* heuristics to automate wrapper building for online stores, but was extremely domain-specific. Kushmerick[16] extended this work by defining some classes of wrappers that could be induced from labelled examples, while Ashish and Knoblock[3] built a toolkit for semi-automatically generating wrappers using hardcoded heuristics.


Real Estate Visualizer

(● Price) 0 100 200 300 400 500

○ Bedrooms 1 2 3 4 5

○ Distance 0 200 400 600

○ Schools 1 2 3 4 5

Require:

[✓] Parking

☐ Garden

☐ Central heating

☐ Balcony

[✓] Use relative color

0% 50% 100%

Fig. 4. Screenshot of an interactive real estate visualization system

Craven *et al.*[8] describe an inductive logic programming algorithm for learning wrappers, also using labelled examples. Cohen[6] introduced a method for learning a general extraction procedure from pairs of page-specific wrappers and the pages they wrap, although the method was restricted to simple list structures. AutoWrapper[11] induces wrappers from unlabelled examples but is restricted to simple table structures. Ghani *et al.*[12] combined extraction of data from corporate websites with data mining on the resulting information. The TSIMMIS project[5] is another system aimed at integrating web data sources; however, its main focus is on query planning and reasoning about source capabilities rather than information extraction (performed by hand-coded wrappers).

6 Conclusions and Future Work

In conclusion, we have demonstrated a principled method for generating information extraction wrappers using grammatical inference that enables the integration of information from multiple web sources. Our approach does not require the overhead of manually-labelled examples, should be applicable to general structures, and ought to be easily adaptable to a variety of domains using domain knowledge expressed in simple declarative rules.

These are still preliminary results, and further work is necessary to test the inference algorithm on more complicated web pages from real-world sources in different domains, and to conduct a more rigorous quantitative evaluation. We would also like to examine the use of simulated annealing in the search.

Acknowledgments. The first author acknowledges the support of the Marshall Aid Commemoration Commission. This material is based upon work supported under a National Science Foundation Graduate Fellowship.

References

1. S. Abiteboul, "Querying semi-structured data," in *Database Theory, 6th International Conference (ICDT '97)*, Delphi, Greece, 1–18. Springer (1997).
2. H. Ahonen, "Automatic generation of SGML content models," Electronic Publishing—Origination, Dissemination and Design **8**(2&3), 195–206 (1995).
3. N. Ashish and C.A. Knoblock, "Semi-automatic wrapper generation for Internet information sources," in *Second IFCIS International Conference on Cooperative Information Systems (CoopIS '97)*, Kiawah Island, SC, USA. IEEE-CS Press (1997).
4. J.K. Baker, "Trainable grammars for speech recognition," *Speech Communication Papers for the 97th Meeting of the Acoustical Society of America*, 547–550 (1979).
5. S. Chawathe, H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. Ullman, and J. Widom, "The TSIMMIS project: integration of heterogenous information sources," in *Proceedings of the 10th Meeting of the Information Processing Society of Japan (IPSJ '94)*, 7–18. (1994).
6. W.W. Cohen, "Recognizing structure in web pages using similarity queries," in *Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI '99)*, Orlando, FL, USA. AAAI Press (1999).
7. C.M. Cook, A. Rosenfeld, and A.R. Aronson, "Grammatical inference by hill climbing," Informational Sciences **10**, 59–80 (1976).
8. M. Craven, D. DiPasquo, D. Freitag, A. McCallum, T. Mitchell, K. Nigam, and S. Slattery, "Learning to construct knowledge bases from the world wide web," Artificial Intelligence **118**, 69–113 (2000).
9. R. Doorenbos, O. Etzioni, and D. Weld, "A scalable comparison-shopping agent for the world-wide web," in *First International Conference on Autonomous Agents (Agents '97)*, Marina del Rey, CA, USA, 39–48. ACM Press (1997).
10. D. Freitag, "Using grammatical inference to improve precision in information extraction," in *ICML '97 Workshop on Automata Induction, Grammatical Inference, and Language Acquisition*, Nashville, TN, USA. (1997).
11. X. Gao and L. Sterling, "AutoWrapper: automatic wrapper generation for multiple online services," in *Asia Pacific Web Conference '99*, Hong Kong. (1999).
12. R. Ghani, R. Jones, D. Mladenić, K. Nigam, and S. Slattery, "Data mining on symbolic knowledge extracted from the web," in *KDD-2000 Workshop on Text Mining*, Boston, MA, USA. (2000).
13. E.M. Gold, "Language identification in the limit," Information and Control **10**, 447–474 (1967).
14. T. Hong, "Visualizing real estate property information on the web," *Information Visualization '99*. IEEE Computer Society, Los Alamitos, CA (1999).
15. B. Krulwich, "The BargainFinder agent: comparison price shopping on the Internet," in *Bots and Other Internet Beasties.* Sams Publishing (1996).
16. N. Kushmerick, "Wrapper induction: efficiency and expressiveness," Artificial Intelligence **118**, 15–68 (2000).
17. Y. Sakakibara, "Recent advances of grammatical inference," Theoretical Computer Science **185**, 15–45 (1997).