A Grammar Inference Algorithm for the World Wide Web

Terrance Goan
Stottler Henke Assoc., Inc.
2016 Belle Monti Ave
Belmont, CA 94002
goan@sirius.com

Nels Benson **Oren Etzioni**
Department of Computer Science and Engineering
University of Washington
Seattle, WA 98105
{nels, etzioni}@cs.washington.edu

Abstract

The World Wide Web is a treasure trove of information. The Web's sheer scale makes automatic location and extraction of information appealing. However, much of the information lies buried in documents designed for human consumption, such as home pages or product catalogs. Before software agents can extract nuggets of information from Web documents, they have to be able to recognize it despite the multitude of formats in which it may appear. In this paper, we take a machine learning approach to the problem. We explain why existing grammar inference techniques face difficulties in this domain, present a new technique, and demonstrate its success on examples drawn from the Web ranging from CMU Tech Report codes to bus schedules. Our algorithm is shown to learn target languages found on the Web in significantly fewer examples than previous methods. In addition, our algorithm is guaranteed to learn in the limit, and runs in time $O(|S|^4)$, where $|S|$ is the size of the sample.

Introduction

A tremendous amount of information is stored in human readable documents on the World Wide Web. Before a software agent, such as the BargainFinder (BargainFinder), can satisfy a goal such as "find the best price for Michael Jackson's recent CD" it has to be able to access such documents and extract price information. Similarly, the Internet softbot (Etzioni & Weld, 1994) has to extract phone numbers and email addresses from home pages and personnel directories. The dynamic nature and sheer size of the Web make automatic information extraction appealing, if not vital. However, the lack of formatting standards for information such as addresses, phone numbers, schedules, product catalog entries and so on, makes their automatic extraction from Web documents difficult. Additionally, it is impractical to expect the users of software agents to have the technical background required to maintain the agent's parsing functions as formats for catalog entries etc. change with time.

The problem of learning to recognize nuggets of structured information on the Web can be decomposed into the following sub-problems:

- **Discovery**: find Web pages containing examples of the target information.
- **Extraction**: identify the location of an example on the page and extract it.
- **Grammar inference**: given a set of examples, learn a grammar characterizing such information.
- **Incorporation**: given a grammar, use it as part of every day activities and refine it if necessary.

The discovery and extraction problems appear hard, and may require human assistance. For example, a human may have to collect and mark up a set of examples as grist for a grammar inference mill. However, tools can be written to facilitate this process. In this paper, we focus exclusively on the grammar inference problem, and leave the others for future work. To demonstrate that our approach to grammar inference is viable, we focus on structured strings commonly found on the Web and test our algorithm on several data sets drawn directly from Web pages. It is difficult if not impossible to argue that our grammar inference algorithm handles the full diversity of information found on the Web; instead we demonstrate empirically that the algorithm is a significant advance in the state of the art of grammar inference, motivated by and tested on Web data. The performance of our learning algorithm forms a base line for future investigations in this domain.

Preliminaries

Grammar inference is defined as the task of learning a grammar from a set of example strings. Since providing positive example strings is much easier for users than providing instructive negative examples (e.g. what is an *instructive* negative example of an address?), we will focus on learning from a positive sample alone. For

tractability we restrict our search to regular grammars (or equivalently regular expressions or deterministic finite automata (DFA)), which provide sufficient expressiveness for our purposes. We can define the input and output of our algorithm WIL (Web Information Learner) as follows:

Input: A set of positive example strings $S^+ \subset L(M)$ where L(M) is the language generated by some automata M.

Output: A finite state automata M' that approximates M.

We further stipulate that the automata M and M' are stochastic, meaning:

- each arc a leaving a node n in M or M' has an associated probability $P_a(a)$
- each node n in M and M' also has an associated termination probability $P_t(n)$
- $P_t(n) + \Sigma P_a(a) = 1$ for nodes in machine M (approximately equal to 1 for nodes in M')
- S^+ is generated stochastically from M

Stochastic grammars are well suited to our task because many real world examples are best seen as being generated by a stochastic process. Further, stochastic grammars provide information as to the likelihood that a string was generated by a particular machine. This property may prove useful in determining the most likely type of a parsed string.

Previous Work

Over the past four decades much work has been done in the area of regular grammar induction. A number of theoretical results have shown it to be a challenging task. For example, Gold (Gold 1978) showed that the identification of arbitrary regular languages from positive examples is undecidable. Despite this, research has continued in a number of directions.

One path can be described broadly as heuristic methods (e.g. Biermann & Feldman 1972; Fu 1982; Gonzalez & Thomason 1978; Miclet 1986). Algorithms of this type guarantee that the output will be a regular grammar, but do not guarantee that the target grammar will be converged upon. Another significant group of algorithms described by Angluin (Angluin & Smith 1983) as "characterizable" (e.g. Angluin 1982; Garcia & Vidal 1990; Schlimmer & Hermens 1993), can in fact guarantee convergence in the limit from positive examples alone. These algorithms focus on special classes of languages such as Angluin's k-reversible languages (Angluin 1982), and guarantee that if the target language is a member of that particular class of languages, the algorithm will converge to the target in the limit. Unfortunately, this approach is also ill suited to our goals, because we simply do not know what class the target languages will be before hand. For example, if we choose Angluin's k-reversible language learning algorithm to learn phone numbers, the results are quite poor because the phone number language is not k-reversible (for any constant k that would yield generalization).

Another significant problem with the above algorithms is that they do not take into consideration the frequency of strings in the sample. The significance of this is best illustrated with an example. Consider the training sample {aa, aaaa, aaa, aaaaa}. It would be reasonable to conclude from the data that the target grammar would accept a^*. But, if the sample was $\{aa^{100}, aaaa^{100}, aaa^{100}, aaaaa^{100}\}$ (where aa^{100} means 100 occurrences of the string aa), a^* would be a far less appealing hypothesis.

Actually, by assuming the data is stochastically generated we can use observed frequencies to learn SRL's (stochastic regular languages) in the limit. These languages fit our needs well (as mentioned above) and allow us to work within a larger hypothesis space while retaining convergence in the limit. Several approaches have been devised to learn SRL's, some of which proved too inefficient to be practical (Wartous and Kuhn 1992, van der Mude 1977), while others based on Bayesian criteria (Stolcke and Omohundro 1994) require the specification of prior probabilities of grammars.

Carrasco and Oncina (Carrasco and Oncina 1994) devised a relatively simple algorithm which provably learns in the limit, and generates stochastic regular grammars in $O(|S|^3)$ time where |S| is the size of the sample. Their algorithm (Algeria) is based on the state-merging method (used in Angluin 1982; Stolcke 1994). At a high level, such algorithms start with a DFA called a prefix tree that represents the training sample and nothing more. The state merging method then searches by some heuristic for equivalence classes of nodes. These classes are then merged, generalizing the DFA by creating new paths from the start node to final states.

While Algeria does quite well on signal processing type data, we have found it to perform poorly on the web domains we are interested in. In this paper we demonstrate why a different approach is required for learning the languages of common data types on the Web. We make use of the following ideas:

1. Often the alphabet of a language is made up of several different types (e.g. numerals, delimiters). If known, this information can help guide generalization decisions.

2. More-data-first generalization: We order generalization decisions by the number of examples supporting the decision. By making better supported decisions first, we increase the likelihood they are correct.

Since we share many of the same motivations as Carrasco and Oncina (Carrasco and Oncina 1994) and our algorithm WIL was based in part on their algorithm, we describe Algeria in the next section. We will then present our method and contrast the two approaches showing how the differences yield a substantial improvement in performance.

Algeria

Briefly stated, the Algeria algorithm starts from a prefix tree representing the training set, and compares pairs of nodes (i, j) in a lexicographical order. Nodes i and j are merged if they are judged to be equivalent by the following measure:

1. The observed outgoing transition frequencies for each symbol $a \in \Sigma$ are equivalent (as measured by the Hoeffding bound (see definition 1)), and

2. the nodes transitioned to from nodes i and j on a particular symbol $a \in \Sigma$, ($\delta(i, a)$, $\delta(j, a)$ respectively) are also judged equivalent (recursively) by this measure.

If

$$\left|\frac{f}{n} - \frac{f'}{n'}\right| > \sqrt{\frac{1}{2}\log\frac{2}{\alpha}}\left(\frac{1}{\sqrt{n}} + \frac{1}{\sqrt{n'}}\right)$$

then observed frequencies $\frac{f}{n}$, $\frac{f'}{n'}$ are equivalent.

n,n': # of strings passing through each node
f,f': # of strings following a given arc
α - 1: confidence

Definition 1. Hoeffding Bound

As we stated in the previous section, Algeria does poorly on domains of interest to us. These domains include such things as: phone numbers, addresses, bibliographic entries, library of congress call numbers, etc. All these example domains have the common characteristic that they generate what we call "bushy" prefix trees. For example, when learning a DFA such as that in Figure 1, prefix trees can contain hundreds of thousands of nodes, creating ample opportunity for faulty generalization.

In order to tackle such grammar induction tasks we need to make better use of the information provided in training examples. Additionally, the search for compatible nodes can be done in a more intelligent manner while maintaining a relatively efficient running time of $O(|S|^4)$.

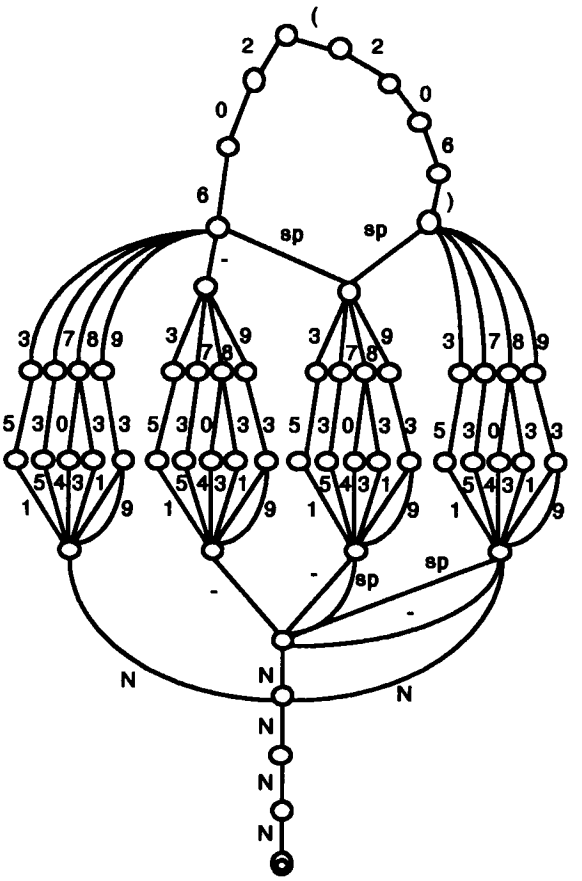

Figure 1. Auburn, Washington phone number DFA. (where sp is a space and N represents 10 arcs labeled 0 to 9)

Getting More Out of Examples

It is often the case that the alphabet of a language being learned can be broken into groups of similar type. Some examples include numerals (0-9), Roman alphabet letters (a-z), and delimiters (such as "," ";" "-"). Utilizing this information allows for more informed decisions about the equality of two states. For example, the root nodes in Figures 2a and 2b could be deemed equivalent by Algeria (depending on the confidence level used and the number of examples). This is because the differences in observed frequencies of output transition symbols are small (|0-.25|). Although the strings generated are of different lengths, this is ignored by Algeria because the root nodes do not share any output symbols. But, if the outputs are viewed by their structure, (see Figures 3a and 3b) we can avoid this merging. We define *structure* as a list of type names describing a string. For example, the string "754-1212" has the structure (n n n d n n n n) where n = number and d = delimiter.

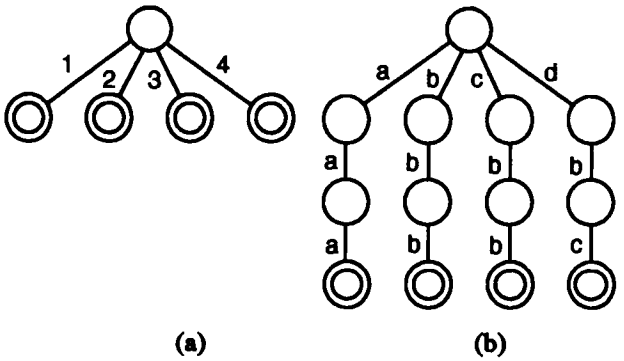

Figure 2. Two prefix trees which can be mistakenly recognized as equivalent by Algeria (Assume equal frequencies for strings).

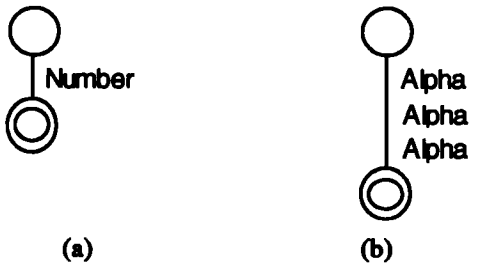

Figure 3. A structural view of the trees in Figure 2. Based on this view, the trees appear much less similar.

Structural information can also be used to uncover valid merges, because strings often look more similar if you focus on their structure. By viewing strings at two levels of abstraction (symbol value and symbol type) we can make better generalization decisions. WIL merges two nodes if they are close based on one measure and not too distant, on the other measure.

Another way in which structure can be useful is when all outputs of a node have the same type description. This is a common occurrence in a wide variety of domains and offers the opportunity to base generalization decisions on the data passing through multiple levels (see Figure 4). When we find a node with all outgoing types being the same, we check each level of nodes below it, merging the nodes if and only if they are all found to be equivalent (we call this "level checking"). This heuristic proves helpful in finding valid merges more quickly (reducing the opportunity for poorly informed merges).

There is yet one more use for type information. It is common in many languages to have locations where any value of a certain type will do ("wildcard" values). For example, the last four digits in a US phone number are wildcard values (0...9). Therefore, it is useful to scan over a DFA after learning and see if there are places where the outgoing arcs are approximately evenly distributed over the range of some type (using the Hoeffding bound again.) If so, then simply fill in the missing arcs, giving each a probability equal to the average of the existing arcs.

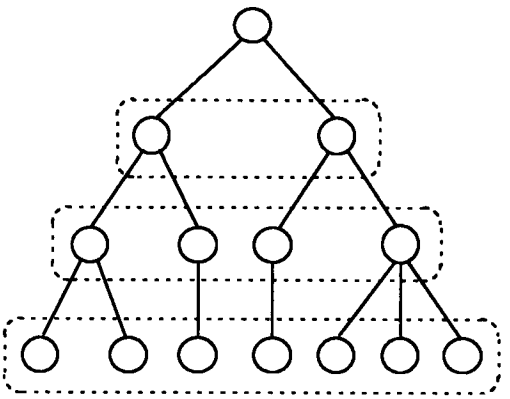

Figure 4. The groups of nodes compared in "level-checking"

More-Data-First Search

Central to many grammar induction algorithms, including Algeria and our algorithm, is the order in which pairs of nodes are compared. We have already described one way in which our search method differs from Algeria (level checking). Another significant difference is our node ordering method. That is, while Algeria makes use of an arbitrary lexicographical ordering of nodes, we take steps to avoid poorly informed merges by focusing our search first among nodes with a higher number of examples passing through them. We call this a more-data-first order:

Order node *i* before node *j* if:

1. the number of examples passing through node *i* > number of examples passing through *j*, or
2. depth(*i*) < depth(*j*) and number of examples passing through *i* and *j* are equal.

This difference in ordering can have a dramatic effect on the accuracy of the learned grammar, and maintains the computational benefits offered by a lexicographical ordering (see the next section).

WIL (Web Information Learner)

```
Algorithm WIL (S alpha beta gamma)
; S     -> sample set
; alpha -> 1 - confidence level (first)
; beta  -> 1 - confidence level (second)
; gamma -> 1 - confidence level (wildcard)

begin
T = stochastic prefix tree ordered by
    more-data-first()

for (k = 0 to last-id)
  if (single-type (node k))
    level-check (node k)
  end-if
end-for

for (j = 1 to last-id)
  for (i = 0 to j - 1)
    if (or equivalent ((node i)
                       (node j) alpha beta)
           equivalent ((node i)
                       (node j) beta alpha))
      merge ((node i) (node j))
      determinize (node i)
      exit (i-loop)
    end-if
  end-for
end-for

for (x = 0 to last-id)
  if (wildcard? ((node x) gamma))
    make-wildcard (node x)
end-for

end
```

The above algorithm can be broken into three simple steps. First, all nodes are checked to see if the examples passing through them are all of the same type. If so, then check each level below that node for equivalence (level checking). Second, examine pairs of nodes in the more-data-first order and merge any equivalent states (as measured by the Hoeffding bound). Third, check the outputs of each state for the possibility that it is equivalent (by Hoeffding bound) to the wildcard value for a given alphabetic type. For this last generalization step to succeed, all output values of the type being checked must share the same destination node.

Efficiency

WIL is guaranteed to generate a deterministic stochastic automata and the computational time for this process can be bound by $O(|S|^4)$. The proof of this is straightforward. First note that the number of nodes which will be compared is at most $O(\binom{|S|}{2}) = O(|S|^2)$. For each such pair (n, n') we will:

1. compare output probabilities of at most $O(|\Sigma|)$ (where $|\Sigma|$ is the size of the alphabet) arcs, because determinism is maintained throughout.
2. compare output probabilities of at most $O(|S|)$ structure-description outputs)
3. compare $O(|\Sigma|)$ pairs of children nodes (of n and n') for equivalence.

Step 3 is the recursive step. The use of the more-data-first ordering guarantees that the graph rooted at least one of n or n' will be acyclical (call this the acyclic node). It follows that at most $O(|S|)$ pairs of nodes in total will be examined during the comparison of n and n', because the number of paths (of output values) leaving the acyclic node is bound by the size of the input $O(|S|)$. Therefore, we have $O(|S| + |\Sigma|)$ work to do for each of $O(|S|)$ equivalence checks of the descendants of n and n'. Now, the runtime of steps 1-3 can be bound by $O(|S|^2) * O(|S|) * O(|S| + |\Sigma|) = O(|S|^4)$.

The remaining significant computation is done in the level checking step. There are $O(|S|)$ nodes (i.e., each node in the prefix tree) for which this computation is done. For each of these nodes there are at most $O(|S|)$ descendant nodes in total. Since the structure of all nodes on a particular level is the same, there is no need to do further structure checking. Therefore, following the above argument, we can place an upper bound of $O(|S|^5)$ on the time needed to compare those $O(|S|)$ nodes. Hence, the cost of the level checking step is $O(|S|) * O(|S|^5) = O(|S|^6)$. Finally, the total runtime cost of the entire algorithm can then be bounded by $O(|S|^4) + O(|S|^6) = O(|S|^6)$.

Learning in the Limit

As pointed out by (Carrasco and Oncina 1994) grammar inference by state merging can be framed as a problem of finding equivalence classes of nodes in a prefix tree acceptor. Therefore, we can show WIL learns in the limit as defined by (Gold 1978) by showing the following:

1. The probability that WIL will mistakenly merge two non-equivalent nodes of a prefix tree acceptor vanishes in the limit.
2. The probability that WIL will mistakenly generalize the output values of a node to a wildcard value vanishes in the limit.

Carrasco and Oncina proved (1) holds for their algorithm Algeria. It is sufficient then to show that our algorithm will never merge two nodes unless Algeria would also. This statement is in fact true, because our test for node equivalence is at least as restrictive as Algeria's due to the addition of structure checks (assuming Algeria's confidence level is at least as great as WIL's two confidence levels). Therefore, (1) holds.

The other concern is the wildcard generalization step. Clearly, if generalizing to a wildcard value is

inappropriate, the difference between the observed output frequencies and the uniformly distributed wildcard outputs will exceed (in the limit) the confidence ranges described by the Hoeffding bound. Therefore, (2) holds, and WIL learns in the limit.

Results and Discussion

To evaluate our ideas we tested WIL on a number of domains and compared the results with those achieved by Algeria. We were particularly interested in learning languages encountered on the Web, but we also wanted to retain the ability to learn pattern recognition domains on which other grammar inference algorithms were shown to do well. The tests included:

A. Phone numbers with a variety of formats from Auburn, Washington (NTC). Figure 5.

B. Library of Congress call numbers for AI books as defined in (Scott 1993). Figure 6.

C. The Reber grammar, as defined in (Reber 1967), and used in the testing of Algeria (Carrasco and Oncina 1994). Figure 7.

D. AT&T's 800 phone numbers (ATT-800). Figure 8.

E. CMU tech report codes (CMU). Figure 9.

F. Bus schedule times (BUS). Figure 10.

The testing procedure commonly utilized by grammar inference researchers (e.g., Stolcke & Omohundro 1994) can be described as follows:

1. Generate a set of training examples using the hand-crafted target automata M.
2. Induce a DFA M' from the training examples.
3. Test for overgeneralization by parsing a set of strings generated by M' with grammar M.
4. Test for overspecificity by parsing additional strings generated by M with grammar M'.
5. The accuracy measurement is taken to be the average of the accuracy levels found in steps 3 and 4.

This is a very good experimental methodology when the goal is to test an algorithm's ability to learn a known abstract automata. We adopted this procedure in tests A, B, and C because it provides for a straightforward comparison of algorithms, and in particular, a clear test for overgeneral hypotheses.

To verify that our method would work on real Web data, we conducted tests D, E, and F using data collected directly from Web sites. This data was used to induce automata as well as test for overspecificity. But, using real data presents a problem with regard to testing for overgeneral hypotheses. The question that arises is "what is a negative example of a phone number?" Clearly every string in Σ^* which is not a phone number could be used as a negative example, but this approach does not make for a very informative test. Therefore, for this second set of tests (D, E, and F) we opted to handcraft automata to generate negative examples which are somewhat similar to, but not members of, the target class. For example, in the case of the AT&T 800 number domain, negative examples were either missing the "800" prefix or had some other three digit prefix, and had five to seven digit suffixes.

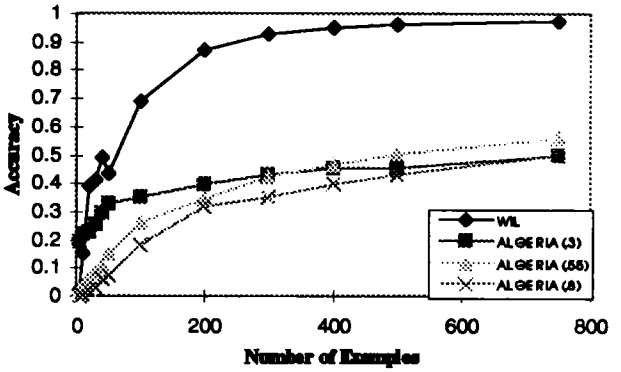

Figure 5. Auburn Phone Numbers

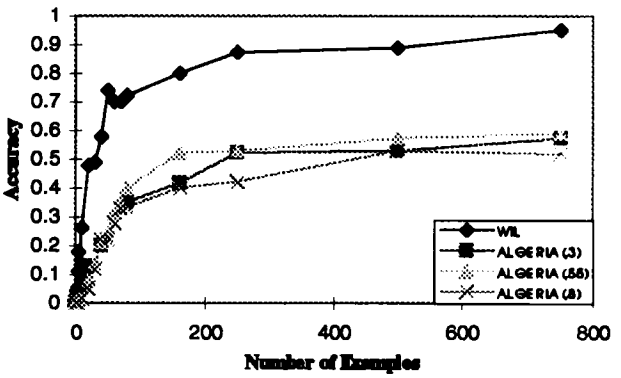

Figure 6. Call Numbers

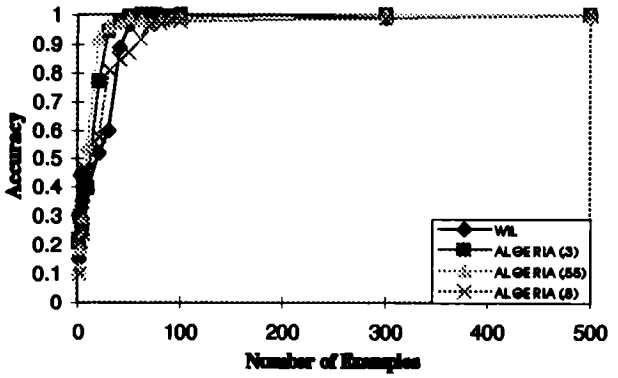

Figure 7. Reber Grammar

In order to thoroughly compare the two algorithms, we conducted tests of Algeria with three different confidence levels in each domain: one with a confidence level equal to WIL's α value, another with WIL's β value, and a third with the average of α and β. As Figures 5 and 6 demonstrate, WIL learns the target automata for the phone number domains and the call number domain in considerably fewer training examples than Algeria. While the rate of improvement for both algorithms starts to taper off at about the same point in each domain, WIL's level of accuracy is double or triple the accuracy of Algeria at that point. Although time did not permit us to carry the tests out further, the shape of the curves suggests that Algeria would take a considerable number of additional examples to catch up. Additionally, test C (Figure 7) showed that WIL retains the ability to learn other pattern recognition domains such as the Reber grammar as well as Algeria does.

Using our second experimental methodology (using training data from the Web) we found similar results. As figures 8, 9, and 10 show, WIL maintains substantially better accuracy levels in each of the domains.

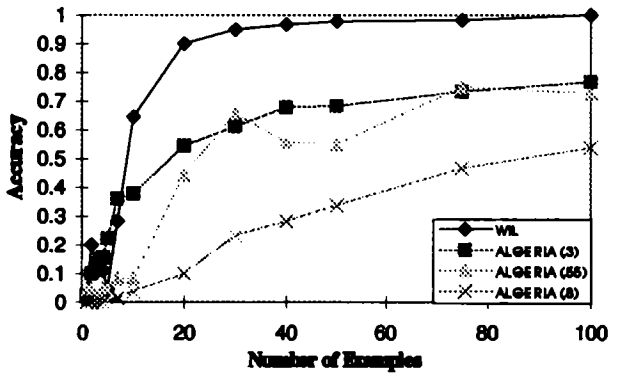

Figure 8. 800 Numbers

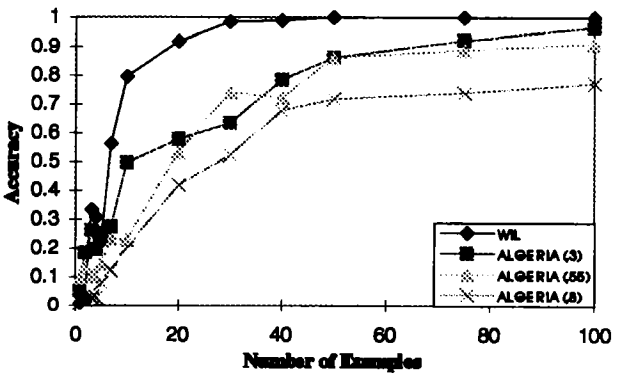

Figure 9. CMU tech report codes

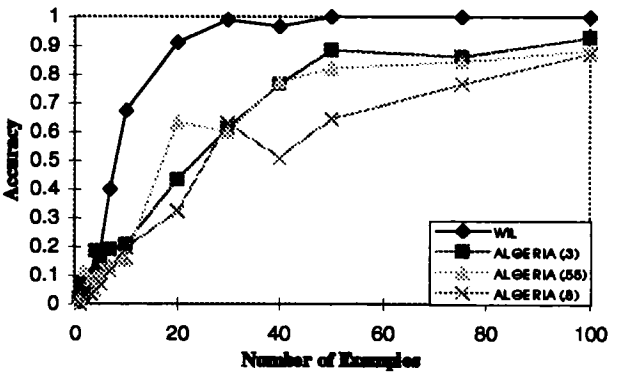

Figure 10. Bus schedule times

On the whole we found structural comparisons, level checking, and wildcard generalization to be quite helpful. Each of these techniques was found to be beneficial when used independently of the others. We also found that these benefits were generally additive. For example, at sample size 100, WIL had an accuracy of 69% on Auburn numbers. When we replaced the more-data-first ordering with a lexicographical ordering, the accuracy fell to 62%. When we further removed the structural comparisons, the accuracy fell to 50%. And finally, removing level checking reduced the accuracy to a mere 26%.

We did however notice that there is a significant danger of overgeneralizing through the use of wildcard generalization if the initial state merging goes awry. We therefore used wildcard generalization in a fairly conservative manner (i.e., we used a low confidence level). While this conservative approach offered little benefit when the number of examples was large, it was quite helpful earlier in learning. For example, the accuracy of WIL on 800 numbers with 20 training examples dropped from 90% to 43% without wildcard.

Future Work

As other experiments demonstrate, WIL appears to be adequate for learning to recognize many different kinds of structured strings commonly found on the Web.

However, there is much more work to be done on improving its performance. One path we are pursuing is the use of more complex type and structural information. Using hierarchical type information and more sophisticated node comparison techniques may yield much better performance. These structural cues may also provide a more intelligent search method for equivalent nodes. It may also help to integrate the goal of finding wildcard values with the state merging process, thus allowing better informed decisions.

Conclusion

Through the use of type information and more-data-first generalization, we have devised a grammar inference technique that appears to be adequate for data commonly found on the web; we have shown our algorithm to be a great improvement over its closest competitor (Algeria). In addition, we have shown our algorithm to run in polynomial time in the size of the sample and have shown that it learns to recognize stochastic regular languages in the limit. While WIL advances the state of the art in grammar inference for the Web, it is only a first step towards the task of learning to recognize information on the Web; the problems of discovery, extraction and incorporation remain open.

References

ATT-800.
http://www.yahoo.com/Business_and_Economy/Companies/Telecommunications/AT_T/AT_T_800_Directory

Angluin, D. 1982. Inference of reversible languages. In *Journal of the ACM*, 29(3):741-765.

Angluin, D., and Smith, C. 1983. Inductive Inference: Theory and Methods. *Computing Surveys, Vol. 15 No. 3.*

BargainFinder.
http://bf.cstar.ac.com/bf/

Biermann, A. W., and Feldman, J. A. 1972. On the synthesis of finite state machines from samples of their behavior. In *IEEE Transactions on Computers*, C-21:592-597.

BUS.
http://transit.metrokc.gov/bus/area_maps/regional.html

Carrasco, R. C., and Oncina, J. 1994. Learning stochastic regular grammars by means of a state merging method. *Second International Colloquium, ICGI-94 Proceedings.* 106-118.

CMU.
http://www.cs.cmu.edu/Web/Reports/index.html

Etzioni, O., and Weld, D. 1994. A softbot-based interface to the internet. *CACM.* 37(7):72-76.

Fu, K-S. 1982. *Syntactic Pattern Recognition and Applications.* Prentice-Hall, Englewood Cliffs.

Gold, E. M. 1978. Complexity of Automaton Identification from Given Data. *Information and Control.* 37:302-320.

Gonzalez, R. C., and Thomason, M. G. 1978. *Syntactic Pattern Recognition: An Introduction.* Massachusetts: Addision-Wesley, Reading.

Garciá P., and Vidal, E. 1990. Inference of *k*-testable languages in the strict sense and application to syntactic pattern recognition. In *IEEE Transactions on Pattern Analysis and Machine Intelligence.* PAMI-12(9):920-925.

Hoeffding, W. Probability inequalities for sums of bounded random variables. In *American Statistical Association Journal.* 58:13-30.

Miclet, L. 1986. *Structural Methods in Pattern Recognition.* Springer-Verlag.

NTC.
http://www.natltele.com/form.html

Reber, A. S. 1967. Implicit Learning of Artificial Grammars. In *Journal of Verbal Learning and Verbal Behaviour.* 6:855-863.

Schlimmer, J., and Hermens, L. 1993. Software Agents: Completing Patterns and Constructing User Interfaces. *Journal of Artificial Intelligence Research.* 1 (61-89).

Scott, M. 1993. *Conversion tables: LC-Dewey, Dewey-LC.* Libraries Unlimited, Englewood, CO.

Stolcke, A., and Omohundro, S. 1994. Inducing probabilistic grammars by Bayesian model merging. *Second International Colloquium, ICGI-94 Proceedings.* 106-118.

van der Mude, A., and Booth, T. L. 1977. Inference of Finite-State Probabilistic Grammars. In *Information and Control.* 38:310-329.

Wartous, R. L., and Kuhn, G. M. 1992. Induction of Finite-state Languages Using Second-Order Recurrent Networks. *Neural Computation.* 4:406-414.