Character-level Analysis of Semi-Structured Documents for Set Expansion

Richard C. Wang
Language Technologies Institute
Carnegie Mellon University
rcwang@cs.cmu.edu

William W. Cohen
Machine Learning Department
Carnegie Mellon University
wcohen@cs.cmu.edu

Abstract

Set expansion refers to expanding a partial set of "seed" objects into a more complete set. One system that does set expansion is SEAL (Set Expander for Any Language), which expands entities automatically by utilizing resources from the Web in a language-independent fashion. In this paper, we illustrated in detail the construction of character-level wrappers for set expansion implemented in SEAL. We also evaluated several kinds of wrappers for set expansion and showed that character-based wrappers perform better than HTML-based wrappers. In addition, we demonstrated a technique that extends SEAL to learn binary relational concepts (e.g., "x is the mayor of the city y") from only two seeds. We also show that the extended SEAL has good performance on our evaluation datasets, which includes English and Chinese, thus demonstrating language-independence.

1 Introduction

SEAL[1] (Set Expander for Any Language) is a set expansions system that accepts input elements (seeds) of some target set S and automatically finds other probable elements of S in semi-structured documents such as web pages. SEAL is a research system that has shown good performance in previously published results (Wang and Cohen, 2007). By using only three seeds and top one hundred documents returned by Google, SEAL achieved 90% in mean average precision (MAP), averaged over 36 datasets from three languages: English, Chinese, and Japanese. Unlike other published research work (Etzioni et al., 2005), SEAL focuses on finding small closed sets of items (e.g., Disney movies) rather than large and more open sets (e.g., scientists).

In this paper, we explore the impact on performance of one of the innovations in SEAL, specifically, the use of character-level techniques to detect candidate regular structures, or *wrappers*, in web pages. Although some early systems for web-page analysis induce rules at character-level (e.g., such as WIEN (Kushmerick et al., 1997) and DIPRE (Brin, 1998)), most recent approaches for set expansion have used either tokenized and/or parsed free-text (Carlson et al., 2009; Talukdar et al., 2006; Snow et al., 2006; Pantel and Pennacchiotti, 2006), or have incorporated heuristics for exploiting HTML structures that are likely to encode lists and tables (Nadeau et al., 2006; Etzioni et al., 2005).

In this paper, we experimentally evaluate SEAL's performance under two settings: 1) using the character-level page analysis techniques of the original SEAL, and 2) using page analysis techniques constrained to identify only HTML-related wrappers. Our conjecture is that the less constrained character-level methods will produce more candidate wrappers than HTML-based techniques. We also conjecture that a larger number of candidate wrappers will lead to better performance overall, due to SEAL's robust methods for ranking candidate wrappers.

The experiments in this paper largely validate this conjecture. We show that the HTML-restricted version of SEAL performs less well, losing 13 points in MAP on a dozen Chinese-language benchmark problems, 8 points in MAP on a dozen English-language problems, and 2 points in MAP on a dozen Japanese-language problems.

SEAL currently only handles unary relationships (e.g., "x" is a mayor). In this paper, we show that SEAL's character-level analysis techniques can, like HTML-based methods, be read-

[1] http://rcwang.com/seal

Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing, pages 1503–1512,
Singapore, 6-7 August 2009. ©2009 ACL and AFNLP

ily extended to handle binary relationships. We then demonstrate that this extension of SEAL can learn binary concepts (e.g., "x is the mayor of the city y") from a small number of seeds, and show that, as with unary relationships, MAP performance is 26 points lower when wrappers are restricted to be HTML-related. Furthermore, we also illustrate that the learning of binary concepts can be bootstrapped to improve its performance.

Section 2.1 explains how SEAL constructs wrappers and rank candidate items for unary relations. Section 3 describes the experiments and results for unary relations. Section 4 presents the method for extending SEAL to handle binary relationships, as well as their experimental results. Related work is discussed in Section 5, and the paper concludes in Section 6.

2 SEAL

2.1 Identifying Wrappers for Unary Relations

When SEAL performs set expansion, it accepts a small number of *seeds* from the user (e.g., "Ford", "Nissan", and "Toyota"). It then uses a web search engine to retrieve some documents that contain these instances, and then analyzes these documents to find *candidate wrappers* (i.e., regular structures on a page that contain the seed instances). Strings that are extracted by a candidate wrapper (but are not equivalent to any seed) are called *candidate instances*. SEAL then statistically ranks the candidate instances (and wrappers), using the techniques outlined below, and outputs a ranked list of instances to the user.

One key step in this process is identifying candidate wrappers. In SEAL, a candidate wrapper is defined by a pair of left and right character strings, ℓ and r. A wrapper "extracts" items from a particular document by locating all strings in the document that are bracketed by the wrapper's left and right strings, but do not contain either of the two strings. In SEAL, wrappers are always learned from, and applied to, a single document.

Table 1 illustrates some candidate wrappers learned by SEAL. (Here, a wrapper is written as ℓ`[...]`r, with the `[...]` to be filled by an extracted string.) Notice that the instances extracted by wrappers can and do appear in surprising places, such as embedded in URLs or in HTML tag attributes. Our experience with these character-based wrappers lead us to conjecture that existing heuristics for identifying structure in HTML are fundamentally limited, in that many potentially useful structures will not be identified by analyzing HTML structure only.

SEAL uses these rules to find wrappers. Each candidate wrapper ℓ, r is a maximally long pair of strings that bracket at least one occurrence of every seed in a document: in other words, for each pair ℓ, r, the set of strings C extracted by ℓ, r has the properties that:

1. For every seed s, there exists some $c \in C$ that is equivalent to s; and
2. There are no strings ℓ', r' that satisfy property (1) above such that ℓ is a proper suffix of ℓ' and r is a proper prefix of r'.

SEAL's wrappers can be found quite efficiently. The algorithm we use has been described previously (Wang and Cohen, 2007), but will be explained again here for completeness. As an example, below shows a mock document, written in an unknown mark-up language, that has the seeds: Ford, Nissan, and Toyota located (and boldfaced). There are two other car makers hidden inside this document (can you spot them?). In this section, we will show you how to automatically construct wrappers that reveal them.

```
GtpKxH<b>nIsSaN</b>xjHJglekuDialcLBxKH<b>forD</b>xkrpW
NaCMwAAHO<b>FoRd</b>uohdEXocUvaGKxHaCuRAxjHjnOx
o<b>TOyOTa</b>zxKHAUdIxkrOyQKxH<b>ToYotA</b>xjHCRdmLxa
puRAPprtqOVKxH<b>foRd</b>xjHaJAScRFrla<b>FoRD</b>ofwNL
WxKH<b>tOYota</b>xkrHxQKlacXlGEKtxKH<b>NisSan</b>xkrEq
```

Given a set of seeds and a semi-structured document, the wrapper construction algorithm starts by locating all strings equivalent to a seed in the document; these strings are called *seed instances* below. (In SEAL, we always use case-insensitive string matching, so a string is "equivalent to" any case variant of itself.) The algorithm then inserts all the instances into a list and assigns a unique id to each of them by its index in the list (i.e., the id of an instance is its position in the list.)

For every seed instance in the document, its immediate left character string (starting from the first character of the document) and right character string (ending at the last character of the document) are extracted and inserted into a *left-context* trie and a *right-context* trie respectively, where the left context is inserted in reversed character order. (Here, we implemented a compact trie called

URL:	`http://www.shopcarparts.com/`
Wrapper:	`.html" CLASS="shopcp">[...] Parts</A>  `
Content:	acura, audi, bmw, buick, cadillac, chevrolet, chevy, chrysler, daewoo, daihatsu, dodge, eagle, ford, ...
URL:	`http://www.allautoreviews.com/`
Wrapper:	`</a>  <a href="auto_reviews/[...]/`
Content:	acura, audi, bmw, buick, cadillac, chevrolet, chrysler, dodge, ford, gmc, honda, hyundai, infiniti, isuzu, ...
URL:	`http://www.hertrichs.com/`
Wrapper:	`<li class="franchise [...]"> <h4><a href="#">`
Content:	buick, chevrolet, chrysler, dodge, ford, gmc, isuzu, jeep, lincoln, mazda, mercury, nissan, pontiac, scion, ...
URL:	`http://www.metacafe.com/watch/1872759/2009_nissan_maxima_performance/`
Wrapper:	`videos">[...]</a> <a href="/tags/`
Content:	avalon, cars, carscom, driving, ford, maxima, nissan, performance, speed, toyota
URL:	`http://www.worldstyling.com/`
Wrapper:	`'>[...] Accessories</option><option value='`
Content:	chevy, ford, isuzu, mitsubishi, nissan, pickup, stainless steel, suv, toyota

Table 1: Examples of wrappers constructed from web pages given the seeds: Ford, Nissan, Toyota.

Patricia trie where every node stores a substring.) Every node in the left-context trie maintains a list of ids for keeping track of the seed instances that follow the string associated with that node. Same thing applies to the right-context trie symmetrically. Figure 1 shows the two context tries and the list of seed instances when provided the mock document with the seeds: Ford, Nissan, and Toyota.

Provided that the left and right context tries are populated with all the contextual strings of every seed instance, the algorithm then finds maximally long contextual strings that bracket at least one seed instance of every seed. The pseudo-code for finding these strings for building wrappers is illustrated in Table 2, where $Seeds$ is the set of input seeds and ℓ is the minimum length of the strings. We observed that longer strings produce higher precision but lower recall. This is an interesting parameter that is worth exploring, but for this paper, we consider and use only a minimum length of one throughout the experiments. The basic idea behind the pseudo-code is to first find all the longest possible strings from one trie given some constraints, then for every such string s, find the longest possible string s' from another trie such that s and s' bracket at least one occurrence of every given seed in a document.

The wrappers constructed as well as the items extracted given the mock document and the example seeds are shown below. Notice that Audi and Acura are uncovered (did you spot them?).

Wrapper:	`xKH[...]xkr`
Content:	**audi**, ford, nissan, toyota
Wrapper:	`KxH[...]xjH`
Content:	**acura**, ford, nissan, toyota

Wrappers MakeWrappers(Trie ℓ, Trie r)
Return Wraps(l, r) $\cup$ Wraps(r, l)

Wrappers Wraps(Trie t_1, Trie t_2)
For each $n_1 \in$ TopNodes(t_1, ℓ)
For each $n_2 \in$ BottomNodes(t_2, n_1)
For each $n_1 \in$ BottomNodes(t_1, n_2)
Construct a new Wrapper(Text(n_1), Text(n_2))
Return a union of all wrappers constructed

Nodes BottomNodes(Trie t_1, Node n')
Find node $n \in t_1$ such that:
(1) NumCommonSeeds(n, n') == $|Seeds|$, and
(2) All children nodes of n (if exist) fail on (1)
Return a union of all nodes found

Nodes TopNodes(Trie t, int ℓ)
Find node $n \in t$ such that:
(1) Text(n).$length \geq \ell$, and
(2) Parent node of n (if exist) fails on (1)
Return a union of all nodes found

String Text(Node n)
Return the textual string represented by the path from root to n in the trie containing n

---

Integer NumCommonSeeds(Node n_1, Node n_2)
For each index $i \in$ Intersect(n_1, n_2):
Find the seed at index i of seed instance list
Return the size of the union of all seeds found

Integers Intersect(Node n_1, Node n_2)
Return $n_1.indexes \cap n_2.indexes$

Table 2: Pseudo-code for constructing wrappers.

Table 1 shows examples of wrappers constructed from real web documents. We have also observed items extracted from plain text (.txt), comma/tab-separated text (.csv/.tsv), latex (.tex), and even Word documents (.doc) of which the wrappers have binary character strings. These observations support our claim that the algorithm is independent of mark-up language. In our experimental results, we will show that it is independent of human language as well.

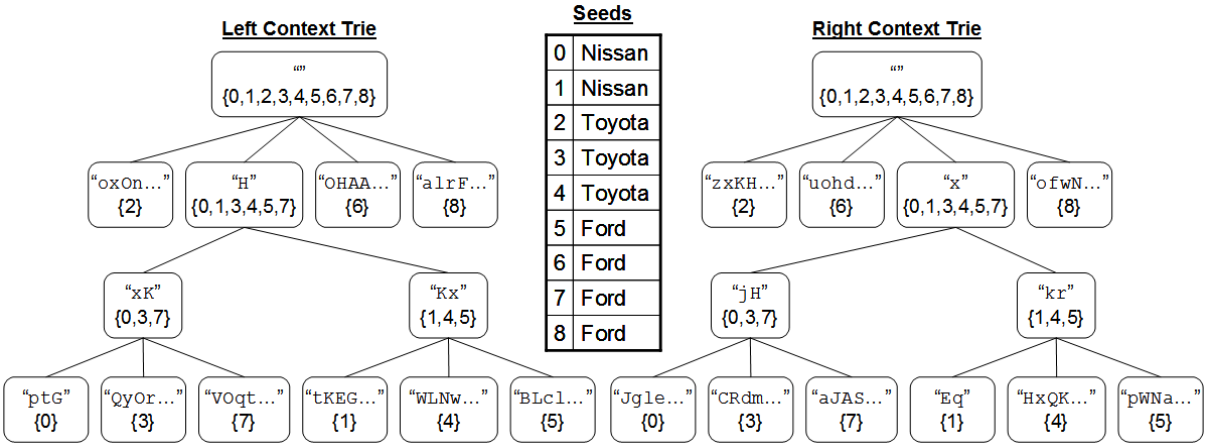

Figure 1: The context tries and the seed instance list constructed given the mock document presented in Section 2.1 and the seeds: Ford, Nissan and Toyota.

2.2 Ranking Wrappers and Candidate Instances

In previous work (Wang and Cohen, 2007), we presented a graph-walk based technique that is effective for ranking sets and wrappers. This model encapsulates the relations between documents, wrappers, and extracted instances (entity mentions). Similarly, our graph also consists of a set of nodes and a set of labeled directed edges. Figure 2 shows an example graph where each node d_i represents a document, w_i a wrapper, and m_i an extracted entity mention. A directed edge connects a node d_i to a w_i if d_i contains w_i, a w_i to a m_i if w_i extracts m_i, and a d_i to a m_i if d_i contains m_i. Although not shown in the figure, every edge from node x to y actually has an inverse relation edge from node y to x (e.g., m_i is extracted by w_i) to ensure that the graph is cyclic.

We will use letters such as x, y, and z to denote nodes, and $x \xrightarrow{r} y$ to denote an edge from x to y with labeled relation r. Each node represents an object (document, wrapper, or mention), and each edge $x \xrightarrow{r} y$ asserts that a binary relation $r(x, y)$ holds. We want to find entity mention nodes that are *similar* to the seed nodes. We define the similarity between two nodes by random walk with restart (Tong et al., 2006). In this algorithm, to walk away from a source node x, one first chooses an edge relation r; then given r, one picks a target node y such that $x \xrightarrow{r} y$. When given a source node x, we assume that the probability of picking an edge relation r is uniformly distributed among the set of all r, where there exist a target node y such that $x \xrightarrow{r} y$. More specifically,

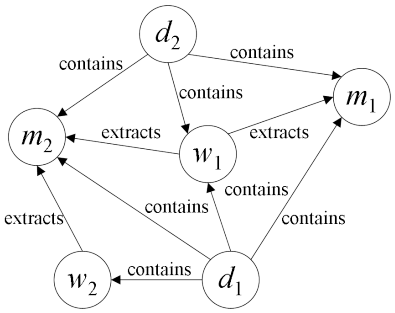

Figure 2: Example graph built by Random Walk.

$$P(r|x) = \frac{1}{|r : \exists y \; x \xrightarrow{r} y|} \tag{1}$$

We also assume that once an edge relation r is chosen, a target node y is picked uniformly from the set of all y such that $x \xrightarrow{r} y$. More specifically,

$$P(y|r, x) = \frac{1}{|y : x \xrightarrow{r} y|} \tag{2}$$

In order to perform random walk, we will build a transition matrix M where each entry at (x, y) represents the probability of traveling one step from a source node x to a target node y, or more specifically,

$$M_{xy} = \sum_r P(r|x)P(y|r, x) \tag{3}$$

We will also define a state vector $\vec{v}_t$ which represents the probability at each node after iterating through the entire graph t times, where one iteration means to walk one step away from every node. The state vector at $t + 1$ iteration is defined as:

$$\vec{v}_{t+1} = \lambda \vec{v}_0 + (1 - \lambda) M \vec{v}_t \tag{4}$$

Since we want to start our walk from the seeds, we initialize v_0 to have probabilities uniformly distributed over the seed nodes. In each step of our walk, there is a small probability λ of teleporting back to the seed nodes, which prevents us from walking too far away from the seeds. We iterate our graph until the state vector converges, and rank the extracted mentions by their probabilities in the final state vector. We use a constant λ of 0.01 in the experiments below.

2.3 Bootstrapping Candidate Instances

Bootstrapping refers to iterative unsupervised set expansion. This process requires minimal supervision, but is very sensitive to the system's performance because errors can easily propagate from one iteration to another. As shown in previous work (Wang and Cohen, 2008), carefully designed seeding strategies can minimize the propagated errors. Below, we show the pseudo-code for our bootstrapping strategy.

$stats \leftarrow \emptyset, used \leftarrow inputs$
for i = 1 to M **do**
  $m = min(3, |used|)$
  $seeds \leftarrow select_m(used) \cup top(list)$
  $stats \leftarrow expand(seeds, stats)$
  $list \leftarrow rank(stats)$
  $used \leftarrow used \cup seeds$
end for

where M is the total number of iterations, $inputs$ are the two initial input seeds, $select_m(S)$ randomly selects m different seeds from the set S, $used$ is a set that contains previously expanded seeds, $top(list)$ returns an item that has the highest rank in $list$, $expand(seeds, stats)$ expands the selected $seeds$ using $stats$ and outputs accumulated statistics, and $rank(stats)$ applies Random Walk described in Section 2.2 on the accumulated $stats$ to produce a $list$ of items. This strategy dumps the highest-ranked item into the $used$ bucket after every iteration. It starts by expanding two input seeds. For the second iteration, it expands three seeds: two *used* plus one from last iteration. For every successive iteration, it expands four seeds: three randomly selected $used$ ones plus one from last iteration.

3 Experiments with Unary Relations

We would like to determine whether character-based or HTML-based wrappers are more suited for the task of set expansion. In order to do that, we introduce five types of wrappers, as illustrated in Table 3. The first type is the character-based wrapper that does not have any restriction on the alphabets of its characters. Starting from the second type, the allowable alphabets in a wrapper become more restrictive. The fifth type requires that an item must be tightly bracketed by two complete HTML tags in order to be extracted.

#	L. Context `[...]` R. Context	Eng	Jap	Chi	Avg
1	`.+[...].+`	87.6	96.9	95.4	93.3
2	`.*[<>].*[...].*[<>].*`	85.7	96.8	90.7	91.1
3	`.*>[...]<.*`	85.7	96.7	90.7	91.0
4	`.*<.+?>.*[...].*<.+?>.*`	80.1	95.8	83.7	86.5
5	`.*<.+?>[...]<.+?>.*`	79.6	94.9	82.4	85.6

Table 3: The performance (MAP) of various types of wrappers on semi-structured web pages.

All pure HTML-based wrappers are type 5, possibly with additional restrictions imposed (Nadeau et al., 2006; Etzioni et al., 2005). SEAL currently does not use an HTML parser (or any other kinds of parser), so restrictions cannot be easily imposed. As far as we know, there isn't an agreement on what restrictions make the most sense or work the best. Therefore, we evaluate performance for varying wrapper constraints from type 1 (most general) to type 5 (most strict) in our experiments.

For set expansion, we use the same evaluation set as in (Wang and Cohen, 2007) which contains 36 manually constructed lists across three different languages: English, Chinese, and Japanese (12 lists per language). Each list contains all instances of a particular semantic class in a certain language, and each instance contains a set of synonyms (e.g., USA, America).

Since the output of our system is a ranked list of extracted instances, we choose mean average precision (MAP) as our evaluation metric. MAP is commonly used in the field of Information Retrieval for evaluating ranked lists because it is sensitive to the entire ranking and it contains both recall and precision-oriented aspects. The MAP for multiple ranked lists is simply the mean value of average precisions calculated separately for each ranked list. We define the average precision of a single ranked list as:

$$AvgPrec(L) = \frac{\sum_{r=1}^{|L|} \text{Prec}(r) \times \text{isFresh}(r)}{\text{Total \# of Correct Instances}}$$

where L is a ranked list of extracted instances, r is the rank ranging from 1 to $|L|$, Prec(r) is the precision at rank r, or the percentage of correct synonyms above rank r (inclusively). isFresh(r) is a binary function for ensuring that, if a list contains multiple synonyms of the same instance (or instance pair), we do not evaluate that instance (or instance pair) more than once. More specifically, the function returns 1 if a) the synonym at r is correct, and b) it is the highest-ranked synonym of its instance in the list; it returns 0 otherwise.

We evaluate the performance of each type of wrapper by conducting set expansion on the 36 datasets across three languages. For each dataset, we randomly select two seeds, expand them by bootstrapping ten iterations (where each iteration retrieves at most 200 web pages only), and evaluate the final result. We repeat this process three times for every dataset and report the average MAP for English, Japanese, and Chinese in Table 3. As illustrated, the more restrictive a wrapper is, the worse it performs. As a result, this indicates that further restrictions on wrappers of type 5 will not improve performance.

4 Set Expansion for Binary Relations

4.1 Identifying Wrappers for Binary Relations

We extend the wrapper construction algorithm described in Section 2.1 to support relational set expansion. The major difference is that we introduce a third type of context called the *middle* context that occurs between the left and right contexts of a wrapper for separating any two items. We execute the same algorithm as before, except that a seed instance in the algorithm is now a seed instance pair bracketing some middle context (i.e., "$s_1 \cdot middle \cdot s_2$").

Given some seed pairs (e.g., Ford and USA), the algorithm first locates the seeds in some given documents. For every pair of seeds located, it extracts their left, middle, and right contexts. The left and right contexts are inserted into their corresponding tries, while the middle context is inserted into a list. Every middle context is assigned a flag indicating whether the two instances bracketing it were found in the same or reversed order as the input seed pairs. Every entry in the seed instance list described previously now stores a pair of instances as one single string (e.g. "Ford/USA"). An id stored in a node now matches the index of a pair of instances as well as a middle context.

Shown below is a mock example document of which the seed pairs: Ford and USA, Nissan and Japan, Toyota and Japan are located (and boldfaced).

GtpKxH**nIsSaN**oKp**jaPaN**xjHJgleTuoLpBlcLBxKH
forDEFc**uSA**xkrpWNapnIkAAHO**FoRd**awHDa**USa**uoh
deQsKxHaCuRAoKpJapANxjHdIjWnOxo**TOyOTa**Vaq
jApaNzxKHAUdIEFcgErmANyxkrOyQKxH**ToYotA**oK
p**JApaN**xjHCRdmtqOVKxH**foRd**oKp**usA**xjHaJASzEi
nSfrla**FoRD**LMmp**uSa**ofwNLWxKH**tOYota**EFc**jAPan**
xkrHxQKzrHpoKdGEKtxKH**NisSan**EFc**JApAn**xkrEq

After performing the abovementioned procedures on this mock document, we now have context tries that are much more complicated than those illustrated in Figure 1, as well as a list of middle contexts similar to the one shown below:

id	Seed Pairs	r	Middle Context
0	Nissan/Japan	No	oKp
1	Nissan/Japan	No	EFc
2	Nissan/Japan	Yes	xkrHxQKzrHpoKd...
4	Toyota/Japan	No	oKp
6	Toyota/Japan	Yes	xjHdIjWnOxo
9	Ford/USA	No	EFc
13	Ford/USA	Yes	xkrpWNapnIkAAHO

where r indicates if the two instances bracketing the middle context were found in the reversed order as the input seed pairs. In order to find the maximally long contextual strings, the "Intersect" function in the set expansion pseudo-code presented in Table 2 needs to be replaced with the following:

Integers Intersect(Node n_1, Node n_2)
  Define $S = n_1.indexes \cap n_2.indexes$
  Return the largest subset s of S such that:
    Every index $\in s$ corresponds to same middle context

which returns those seed pairs that are bracketed by the strings associated with the two input nodes with the same middle context. A wrapper for relational set expansion, or *relational wrapper*, is defined by the left, middle, and right contextual strings. The relational wrappers constructed from the mock document given the example seed pairs are shown below. Notice that Audi/Germany and Acura/Japan are discovered.

Wrapper:	xKH[.1.]EFc[.2.]xkr
Content:	**audi/germany**, ford/usa, nissan/japan, toyota/japan
Wrapper:	KxH[.1.]oKp[.2.]xjH
Content:	**acura/japan**, ford/usa, nissan/japan, toyota/japan

Dataset ID	Item #1 vs. Item #2	Lang. #1	Lang. #2	Size	Complete?
US Governor	US State/Territory vs. Governor	English	English	56	Yes
Taiwan Mayor	Taiwanese City vs. Mayor	Chinese	Chinese	26	Yes
NBA Team	NBA Team vs. NBA Team	Chinese	English	30	Yes
Fed. Agency	US Federal Agency Acronym vs. Full Name	English	English	387	No
Car Maker	Car Manufacturer vs. Headquartered Country	English	English	122	No

Table 4: The five relational datasets for evaluating relational set expansion.

	Mean Avg. Precision					Precision@100				
Datasets	**1**	**2**	**3**	**4**	**5**	**1**	**2**	**3**	**4**	**5**
US Governor	97.4	89.3	89.2	89.3	89.2	55	50	51	50	50
Taiwan Mayor	99.8	95.6	94.3	91.3	90.8	25	25	24	23	23
NBA Team	100.0	99.9	99.9	99.9	99.2	30	30	30	30	30
Fed. Agency	43.7	14.5	5.2	11.1	5.2	96	55	20	40	20
Car Maker	61.7	0.0	0.0	0.0	0.0	74	0	0	0	0
Average	80.5	59.9	57.7	58.3	56.9	56	32	25	29	25

Table 5: Performance of various types of wrappers on the five relational datasets after first iteration.

	Mean Avg. Precision					Precision@100				
Datasets	**1**	**2**	**3**	**4**	**5**	**1**	**2**	**3**	**4**	**5**
US Governor	98.9	97.0	95.3	94.1	93.9	55	55	54	53	53
Taiwan Mayor	99.8	98.3	96.9	93.8	94.3	25	25	25	24	24
NBA Team	100.0	100.0	99.2	98.4	98.6	30	30	30	30	30
Fed. Agency	65.5	54.5	27.9	55.3	30.0	97	97	61	95	69
Car Maker	81.6	0.0	0.0	0.0	0.0	90	0	0	0	0
Average	89.2	70.0	63.9	68.3	63.4	59	41	34	40	35

Table 6: Performance of various types of wrappers on the five relational datasets after 10^{th} iteration.

4.2 Experiments with Binary Relations

For binary relations, we performed the same experiment as with unary relations described in Section 3. A relational wrapper is of type t if the wrapper's left and right context match t's constraint for left and right respectively, and also that the wrapper's middle context match both constraints.

For choosing the evaluation datasets for relational set expansion, we surveyed and obtained a dozen relationships, from which we randomly selected five of them and present in Table 4. Each dataset was then manually constructed. For the last two datasets, since there are too many items, we tried our best to make the lists as exhaustive as possible.

To evaluate relational wrappers, we performed relational set expansion on randomly selected seeds from the five relational datasets. For every dataset, we select two seeds randomly and bootstrap the relational set expansion ten times. The results after the first iteration are shown in Table 5 and after the tenth iteration in Table 6. When computing precision at 100 for each resulting list, we kept only the top-most-ranked synonym of every instance and remove all other synonyms from the list; this ensures that every instance is unique. Notice that for the "Car Maker" dataset, there exists no wrappers of types 2 to 5; thus resulting in zero performance for those wrapper types. In each table, the results indicate that character-based wrappers perform the best, while those HTML-based wrappers that require tight HTML bracketing of items (type 3 and 5) perform the worse.

In addition, the results illustrate that bootstrapping is effective for expanding relational pairs of items. As illustrated in Table 6, the result of finding translation pairs of NBA team names is perfect, and it is almost perfect for finding pairs of U.S. states/territories and governors, as well as Taiwanese cities and mayors. In finding pairs of acronyms and full names of federal agencies, the precision at top 100 is nearly perfect (97%). The results for finding pairs of car makers and countries is good as well, with a high precision of 90%. For the last two datasets, we believe that MAP could be improved by increasing the number of bootstrapping iterations. Table 7 shows some example wrappers constructed and instances extracted for wrappers of type 1.

Seeds: 台中縣 / 黃仲生, 嘉義縣 / 陳明文

URL: http://www.chinareviewnews.org/crn-webapp/attcoluOutline.jsp?coluid=40
Wrapper: 黨[.2.]宣布當選[.1.]長 (2005-12-03 1
URL: http://www.siongpo.com/20051204/localk.htm
Wrapper: 黨)；[.1.]：[.2.]（
URL: http://www.books.com.tw/exep/prod/booksfile.php?item=0010430903
Wrapper: 1 [.1.]長[.2.]最愛私房餐廳
URL: http://mag.udn.com/mag/vote2005/storypage.jsp?f_ART_ID=19060&pno=1
Wrapper: (國民黨)
 [.1.]長[.2.](
Content: 南投縣 / 蔡煌瑯, 台中市 / 胡志強, 台中縣 / 黃仲生, 台南縣 / 蘇煥智, 台東縣 / 劉櫂豪, 嘉義縣 / 陳明振
嘉義縣 / 陳明文, 宜蘭縣 / 陳定南, 彰化市 / 柯金德, 新竹市 / 林政則, 新竹市 / 鄭貴元, 新竹縣 / 鄭永金
桃園縣 / 朱立倫, 花蓮縣 / 盧博基, 花蓮縣 / 謝深山, 苗栗市 / 彭維彬, 苗栗縣 / 劉政鴻, 連江縣 / 陳雪生

Seeds: 鳳凰城太陽 / phoenix suns, 密爾瓦基公鹿 / milwaukee bucks

URL: http://www.xxlbasketball.com.tw/NBA-5/index.htm
Wrapper: >[.1.]
[.2.]</td>
URL: http://sandy0932266920.pixnet.net/blog/post/21155354
Wrapper: >[.1.]隊([.2.])
URL: http://www.nbamag.com.tw/html/oldmags/mag18.html
Wrapper: /updatemid.html">[.1.]
[.2.]<hr><td align=center>
URL: http://tw.myblog.yahoo.com/jw!sNocSW.QEkbxPfvMO5U8qk82LSGu/article?mid=23495
Wrapper:
[.1.]([.2.])
Content: 丹佛金塊 / denver nuggets, 錫拉丘茲民族隊 / syracuse nations, 亞特蘭大老鷹 / atlanta hawks, 休士頓火箭 / houston rockets
克裡夫蘭騎士 / cleveland cavaliers, 印第安那溜馬 / indiana pacers, 夏洛特山貓 / charlotte bobcats, 多倫多暴龍 / toronto raptors
奧蘭多魔術 / orlando magic, 孟菲斯灰熊 / memphis grizzlies, 密爾瓦基公鹿 / milwaukee bucks, 底特律活塞 / detroit pistons
明尼蘇達灰狼 / minnesota timberwolves, 沙加緬度國王 / sacramento kings, 波士頓塞爾蒂克 / boston celtics
波特蘭拓荒者 / portland trail blazers, 洛杉磯快艇 / l.a. clippers, 洛杉磯湖人 / l.a. lakers, 猶他爵士 / utah jazz

Seeds: **kentucky / steve beshear, north dakota / john hoeven**

URL: http://wikifoia.pbworks.com/Alaska-Governor-Sarah-Palin
Wrapper: Governor [.2.]">[.1.] Governor
URL: http://blogs.suntimes.com/sweet/2008/02/sweet_state_dinner_for_governo.html
Wrapper:

 The Honorable [.2.], Governor of [.1.]

URL: http://en.wikipedia.org/wiki/United_States_Senate_elections,_2010
Wrapper: " title="Governor of [.1.]">Governor <a href="/wiki/[.2.]" title="
URL: http://ballotbox.governing.com/2008/07/index.html
Wrapper: , [.1.]'s [.2.],
Content: alabama / bob riley, alaska / sarah palin, arizona / janet napolitano, arkansas / mike huckabee, california / arnold schwarzenegger, colorado / bill ritter, connecticut / mary jodi rell, delaware / ruth ann minner, florida / charlie crist, georgia / sonny perdue, hawaii / linda lingle, idaho / butch otter, illinois / rod blagojevich. . .

Seeds: **cia / central intelligence agency, usps / united states postal service**

URL: http://www1.american.edu/dccampus/links/whitehouse.html
Wrapper: [.2.]
URL: http://www.usembassy.at/en/us/gov.htm
Wrapper: /" target="_blank">[.2.] ([.1.]) -
URL: http://www.nationmaster.com/encyclopedia/List-of-United-States-federal-agencies
Wrapper: The [.2.] ([.1.]) is
URL: http://www.nationmaster.com/encyclopedia/List-of-United-States-federal-agencies
Wrapper: [.1.]- <a href="/encyclopedia/[.2.]" onmouseover="pv(event, 2
Content: achp / advisory council on historic preservation, arc / appalachian regional commission, cftc / commodity futures trading commission, cia / central intelligence agency, cms / centers for medicare and medicaid services, exim bank / export import bank of the united states, ntrc / national transportation research center. . .

Seeds: **mazda / japan, venturi / france**

URL: http://www.jrfilters.com/filtres/index.php?lng=en
Wrapper: &page=filtres&lng=en">[.1.]   ([.2.])</option><option value="index.php?
URL: http://www.jrfilters.com/suspensions/index.php?famille=1&lng=en
Wrapper: &lng=en">[.1.]   ([.2.])</option><option value="index.php?famille=1&rubrique1
URL: http://www.street-car.net/forums/forumdisplay.php?f=10
Wrapper: ">[.1.] Country of origin:[.2.].
URL: http://www.allcarcentral.com/
Wrapper: file.html">[.1.],[.2.]
Content: abarth / italy, acura / japan, alfa romeo / italy, aston martin / england, auburn / usa, audi / germany, austin healey / england, austin / england, auto union / germany, balwin / usa, bandini / italy, bentley / england, bmw / germany, brabham / england, bricklin / usa, bristol / england, brm / england, bucciali / france. . .

Table 7: Examples of (type 1) wrappers constructed and instances (contents) extracted.

5 Related Work

In recent years, many research has been done on extracting relations from free text (e.g., (Pantel and Pennacchiotti, 2006; Agichtein and Gravano, 2000; Snow et al., 2006)); however, almost all of them require some language-dependent parsers or taggers for English, which restrict the language of their extractions to English only (or languages that have these parsers). There has also been work done on extracting relations from HTML-structured tables (e.g., (Etzioni et al., 2005; Nadeau et al., 2006; Cafarella et al., 2008)); however, they all incorporated heuristics for exploiting HTML structures; thus, they cannot handle documents written in other mark-up languages.

Extracting relations at character-level from semi-structured documents has been proposed (e.g., (Kushmerick et al., 1997),(Brin, 1998)). In particular, Brin's approach (DIPRE) is the most similar to ours in terms of expanding relational items. One difference is that it requires maximally-long contextual strings to bracket *all* seed occurrences. This technique has been experimentally illustrated to perform worse than SEAL's approach on unary relations (Wang and Cohen, 2007). Brin presented five seed pairs of author names and book titles that he used in the experiment (unfortunately, he did not provide detailed results). We input the top two seed pairs listed in his paper into the relational SEAL, performed ten bootstrapping iterations (took about 3 minutes), and obtained 26,000 author name/book title pairs of which the precision at 100 is perfect (100%).

6 Conclusions

In this paper, we have described in detail an algorithm for constructing document-specific wrappers automatically for set expansion. In the experimental results, we have illustrated that character-based wrappers are better suited than HTML-based wrappers for the task of set expansion. We also presented a method that utilizes an additional middle context for constructing relational wrappers. We also showed that our relational set expansion approach is language-independent; it can be applied to non-English and even cross-lingual seeds and documents. Furthermore, we have illustrated that bootstrapping improves the performance of relational set expansion. In the future, we will explore automatic mining of binary concepts given only the relation (e.g., "mayor of").

7 Acknowledgments

This work was supported by the Google Research Awards program.

References

Eugene Agichtein and Luis Gravano. 2000. Snowball: Extracting relations from large plain-text collections. In *In Proceedings of the 5th ACM International Conference on Digital Libraries*, pages 85–94.

Sergey Brin. 1998. Extracting patterns and relations from the world wide web. In *In WebDB Workshop at 6th International Conference on Extending Database Technology, EDBT98*, pages 172–183.

Michael J. Cafarella, Alon Y. Halevy, Daisy Z. Wang, Eugene W. 0002, and Yang Zhang. 2008. Webtables: exploring the power of tables on the web. *PVLDB*, 1(1):538–549.

A. Carlson, J. Betteridge, E.R. Hruschka Junior, and T.M. Mitchell. 2009. Coupling semi-supervised learning of categories and relations. In *NAACL HLT Workshop on Semi-supervised Learning for Natural Language Processing*, pages 1–9. Association for Computational Linguistics.

Oren Etzioni, Michael J. Cafarella, Doug Downey, Ana-Maria Popescu, Tal Shaked, Stephen Soderland, Daniel S. Weld, and Alexander Yates. 2005. Unsupervised named-entity extraction from the web: An experimental study. *Artif. Intell.*, 165(1):91–134.

N. Kushmerick, D. Weld, and B. Doorenbos. 1997. Wrapper induction for information extraction. In *Proc. Int. Joint Conf. Artificial Intelligence*.

David Nadeau, Peter D. Turney, and Stan Matwin. 2006. Unsupervised named-entity recognition: Generating gazetteers and resolving ambiguity. In Luc Lamontagne and Mario Marchand, editors, *Canadian Conference on AI*, volume 4013 of *Lecture Notes in Computer Science*, pages 266–277. Springer.

Patrick Pantel and Marco Pennacchiotti. 2006. Espresso: leveraging generic patterns for automatically harvesting semantic relations. In *ACL-44: Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, pages 113–120, Morristown, NJ, USA. Association for Computational Linguistics.

Rion Snow, Daniel Jurafsky, and Andrew Y. Ng. 2006. Semantic taxonomy induction from heterogenous evidence. In *ACL '06: Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the ACL*, pages 801–808, Morristown, NJ, USA. Association for Computational Linguistics.

Partha P. Talukdar, Thorsten Brants, Mark Liberman, and Fernando Pereira. 2006. A context pattern induction method for named entity extraction. In *Tenth Conference on Computational Natural Language Learning (CoNLL-X)*.

Hanghang Tong, Christos Faloutsos, and Jia-Yu Pan. 2006. Fast random walk with restart and its applications. In *ICDM*, pages 613–622. IEEE Computer Society.

Richard C. Wang and William W. Cohen. 2007. Language-independent set expansion of named entities using the web. In *ICDM*, pages 342–350. IEEE Computer Society.

Richard C. Wang and William W. Cohen. 2008. Iterative set expansion of named entities using the web. In *ICDM*, pages 1091–1096. IEEE Computer Society.