Language-Independent Set Expansion of Named Entities using the Web

Richard C. Wang and William W. Cohen
Language Technologies Institute
Carnegie Mellon University
Pittsburgh, PA 15213 USA
{rcwang, wcohen}@cs.cmu.edu

Abstract

Set expansion refers to expanding a given partial set of objects into a more complete set. A well-known example system that does set expansion using the web is Google Sets. In this paper, we propose a novel method for expanding sets of named entities. The approach can be applied to semi-structured documents written in any markup language and in any human language. We present experimental results on 36 benchmark sets in three languages, showing that our system is superior to Google Sets in terms of mean average precision.

1. Introduction

Have you ever wanted to know all the constellations, or US presidents, but were only able to remember the names of a few of them? In this paper we consider the problem of *set expansion* using the web as a resource. In *set expansion,* the user issues a query consisting of a small number of *seeds* $x_1, x_2, ..., x_k$ (e.g., "ursa major", "orion") where each x_i is a member of some *target set* S_t. The answer to the query is a listing of other probable elements of S_t (e.g., "ursa minor", "cancer", "canis major", etc).

Google Sets™ is a well-known example of a web-based set expansion system[1]. Google Sets has been used for a number of purposes in the research community, including deriving features for named-entity recognition [1], and evaluation of question answering systems [2]; unfortunately, however, Google Sets is a proprietary method that may be changed at any time, so research results based on Google Sets cannot be reliably replicated. Set expansion using the web is also closely related to the problem of *unsupervised relation learning* from the web [3, 4], and set-expansion-like techniques have been used to derive features for concept-learning [5], to construct "pseudo-users" for collaborative filtering [6], and to compute similarity between attribute values in autonomous databases [7].

Here we describe a set-expansion system called the Set Expander for Any Language (SEAL). As we will detail below, SEAL works by automatically finding semi-structured web pages that contain "lists" of items, and then aggregating these "lists" so that the "most promising" items are ranked higher. Unlike earlier systems, the SEAL method is simple enough to be easily described and replicated, and is independent of the human language from which the seeds are taken. SEAL is also independent of the markup language used to annotate the semi-structured documents. Extensive experiments have been conducted with SEAL, based on 36 benchmark problems from three languages, each of which consists of a moderate-sized set of entities that is semantically well-defined (e.g., the constellations, or the major-league baseball teams). With randomly constructed three-seed queries from these domains, SEAL obtains a mean average precision (MAP) of more than 94% for English-language queries, more than 93% for Japanese queries, and nearly 95% for Chinese queries. MAP performance on the English-language queries is more than double that of Google Sets™. (Google Sets cannot be used for non-English queries).

In more detail, SEAL is based on two separate research contributions. To find "lists" of items on semi-structured pages, SEAL uses a novel technique to automatically construct *wrappers* (i.e., page-specific extraction rules) for each page that contains the seeds. Every *wrapper* is defined by two character strings, which specify the left-context and right-context necessary for an entity to be extracted from a page. These strings are chosen to be maximally-long contexts that bracket at least one occurrence of every seed string on a page. The use of character-level wrapper definitions means that SEAL is completely

[1] http://labs.google.com/sets

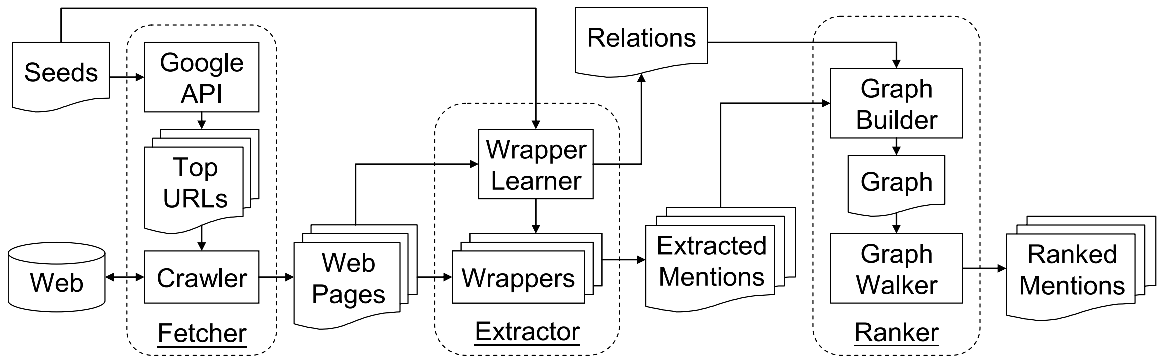

Figure 1. Flow chart of the SEAL system

language-independent: it is not even necessary to be able to tokenize the target language.

Most of the wrappers that SEAL discovers will be "noisy" – i.e., they will extract some entities not in the user's target set S_t. Thus, it is important to rank entities, so that the entities most likely to be in the target set are ranked higher. To rank entities, SEAL uses another novel approach: a graph is built containing all seeds, all constructed wrappers, and all extracted candidate entities. Candidates are then ranked by "similarity" to the seed entities, according to a certain measure of similarity in the graph. The similarity metric is defined by aggregating the results of many randomly-selected walks through the graph, where each walk is defined by a particular random process.

The paper is organized as follows. Section 2 illustrates the architecture of SEAL system. Section 3 describes how wrappers are constructed. Section 4 explains our ranking scheme based on graph walk. Section 5 presents our evaluation dataset. Section 6 illustrates our experimental results. The last section summarizes this paper and describes our future work.

2. System Architecture

SEAL is comprised of three major components: the Fetcher, the Extractor, and the Ranker. The Fetcher focuses on fetching web pages from the World Wide Web. The URLs of the web pages come from top results provided by Google, and the query is simply the concatenation of all seeds (each seed is quoted, to require that it occur as an exact phrase). The Extractor then learns one or more wrappers for each page, and then executes the wrappers, to extract additional candidate entities (see Section 3). Finally, the Ranker builds a graph, and then ranks the extracted mentions globally based on the weights computed in the graph walk (see Section 4).

3. The Extractor

The extractor must learn wrappers instantly and automatically from only a few training examples (the seeds). In this section, we explain the semi-structured characteristics of web documents that SEAL requires, and describe an unsupervised approach for automatic construction of wrappers. The wrappers that are constructed are page-dependent – i.e., they are intended to be applied only to a single web page. However, the approach that we use to learn wrappers is both domain- and language- independent.

3.1. Semi-Structured Documents

The information in semi-structured web documents will be formatted quite differently on different pages, but fairly consistently within a single page. For example, each movie name in a list of classic Disney movies might be be embedded with "`<tr><td>`" (to the left) and "`</td></tr>`" (to the right) in one page, and "`<ul>Disney: `" and "`</ul>`" in another. This observation suggests that entities belonging to the same class (i.e. movies) will be linked by appearing in similar contexts (formatting structures) on the same page.

This characteristic of semi-structured web documents can be exploited for expanding some set of given seeds. Suppose initially, a couple of seeds are provided from a reliable source (i.e. a human), and web documents that contain all of these seeds are retrieved. Then it is very likely that each of these documents will contain other entities that are embedded in the same contexts as the seeds, and also belong to the same semantic class as the seeds. The next section explains in detail the algorithm for constructing wrappers utilizing the semi-structured characteristics of web documents.

Definition:

1. $s_{i,j} = j^{\text{th}}$ occurrence of i^{th} seed, $s_i \in \{s_1,\ldots,s_k\}$, in a document $d \in D$
2. $l_{i,j}$ and $r_{i,j}$ (left and right context respectively) so $d = l_{i,j} \cdot s_{i,j} \cdot r_{i,j}$
3. $L_i = \{l_{i,1},\ldots,l_{i,n_i}\}$ and $R_i = \{r_{i,1},\ldots,r_{i,n_i}\}$
4. $\text{FullSuffix}(L_1,\ldots,L_k) = \{x : \forall i\ x \text{ is a suffix of some } l_{i,j} \in L_i\}$
5. $\text{LongestFullSuffix}(L_1,\ldots,L_k) = \{y : y \in \text{FullSuffix}(L_1,\ldots,L_k) \wedge \neg\exists y' \in \text{FullSuffix}(L_1,\ldots,L_k) : y \text{ is a suffix of } y'\}$
6. $\text{LongestFullPrefix}(R_1,\ldots,R_k)$ is analogous to 5 above

Pseudo code:

```
Let $X = \text{LongestFullSuffix}(L_1,\ldots,L_k)$
$\forall x \in X$
    $\forall i : 1,\ldots,k$
        Let $R'_i = \{r' \in R_i : d = z \cdot x \cdot s_i \cdot r' \text{ for some } z\}$  (i.e. $R'_i$ is a subset of $R_i$ with $s_i$ preceded by $x$)
    Let $\widetilde{R}_x = \text{LongestFullPrefix}(R'_1,\ldots,R'_k)$
    $\forall y \in \widetilde{R}_x$ create pattern "$x \cdot * \cdot y$" $= \{a : x \cdot a \cdot y \text{ is a substring of } d \wedge x, y \text{ are not a substring of } a\}$
Let $Y = \text{LongestFullPrefix}(R_1,\ldots,R_k)$
$\forall y \in Y$
    $\forall i : 1,\ldots,k$
        Let $L'_i = \{l' \in L_i : d = l' \cdot s_i \cdot y \cdot z \text{ for some } z\}$  (i.e. $L'_i$ is a subset of $L_i$ with $s_i$ followed by $y$)
    Let $\widetilde{L}_y = \text{LongestFullSuffix}(L'_1,\ldots,L'_k)$
    $\forall x \in \widetilde{L}_y$ create pattern "$x \cdot * \cdot y$" $= \{a : x \cdot a \cdot y \text{ is a substring of } d \wedge x, y \text{ are not a substring of } a\}$
```

Figure 2. Pseudo-code for automatic construction of wrappers.

3.2. Algorithm

First, the top *n* URLs returned by Google, using the seeds as the query, are downloaded from the web. All instances of seeds are identified from web documents *D* by simple string matching. For each document $d \in D$, let $s_{i,j}$ be the j^{th} occurrence of i^{th} seed. Let the *left context* $l_{i,j} \in L$ be the part of *d* preceding $s_{i,j}$ and the *right context* $r_{i,j} \in R$ be the part of *d* following $s_{i,j}$. For each $d \in D$, all possible suffixes of some left context from *L* and all prefixes of some right context from *R* that embed *at least one instance of every seed* are (conceptually) extracted; these are referred to as *full* suffixes and *full* prefixes respectively. Within these *full* suffixes, ones that are suffixes of other *full* suffixes are removed, keeping only the *longest full* suffixes. To find the *longest full* suffixes, we build a trie of all the suffixes, and each node is marked with the number of unique seeds that the suffix precedes. For each of those longest suffixes, it is easy to find its corresponding longest right context (and vice versa). An extraction pattern, or wrapper, is then constructed for each of those pairs. When extracting candidate entities using left and right contexts *L* and *R,* we consider only substrings between *L* and *R* which do not contain both *L* and *R*.

Since web documents are usually structured consistently within the same page but not across multiple pages, the wrappers derived from a particular document *d* are used to extract from *d* only. We will call each such extracted string a (candidate) *entity mention.* The complete pseudo-code for building these wrappers is described in Figure 2.

Note that our approach is completely character-based and does not assume any language or domain. Also, unlike prior approaches [4, 8], we do not impose any limit on the length of the contextual strings in *L* and *R* nor do we require any parser (i.e. HTML). This also implies that our algorithm will apply not only on HTML pages, but also on other documents semi-structured by any kind of mark-up language (i.e. XML, SGML, TeX, Wiki Markup Language, etc.).

```
...<li class="ford"><a href="http://www.curryford.com/">
<img src="/common/logos/ford/logo-horiz-rgb-lg-dkbg.gif" alt="3"></a>
    <ul><li class="last"><a href="http://www.curryford.com/">
        <span class="dName">Curry Ford</span>...</li></ul>
</li>
<li class="honda"><a href="http://www.curryhonda.com">
<img src="/common/logos/honda/logo-horiz-rgb-lg-dkbg.gif" alt="4"></a>
    <ul><li><a href="http://www.curryhonda-ga.com/">
        <span class="dName">Curry Honda Atlanta</span>...</li>
        <li><a href="http://www.curryhondamass.com/">
            <span class="dName">Curry Honda</span>...</li>
        <li class="last"><a href="http://www.curryhondany.com/">
            <span class="dName">Curry Honda Yorktown</span>...</li></ul>
</li>
<li class="acura"><a href="http://www.curryacura.com/">
<img src="/curryautogroup/images/logo-horiz-rgb-lg-dkbg.gif" alt="5"></a>
    <ul><li class="last"><a href="http://www.curryacura.com/">
        <span class="dName">Curry Acura</span>...</li></ul>
</li>
<li class="nissan"><a href="http://www.geisauto.com/nissan/">
<img src="/common/logos/nissan/logo-horiz-rgb-lg-dkbg.gif" alt="6"></a>
    <ul><li class="last"><a href="http://www.geisauto.com/nissan/">
        <span class="dName">Curry Nissan</span>...</li></ul>
</li>
<li class="toyota"><a href="http://www.geisauto.com/toyota/">
<img src="/common/logos/toyota/logo-horiz-rgb-lg-dkbg.gif" alt="7"></a>
    <ul><li class="last"><a href="http://www.geisauto.com/toyota/">
        <span class="dName">Curry Toyota</span>...</li></ul>
</li>...
```

Figure 3. HTML source text from 'curryauto.com' ("…" is omitted text).

Table 1. Wrappers induced from 'curryauto.com' and their extracted entity mentions.

Wrapper #1:	`\n<li class="[...]"><a href="http://www.`
Extractions:	**ford**, honda, acura, kia, **toyota**, scion, **nissan**, buick, pontiac
Wrapper #2:	`/">\n<img src="/common/logos/[...]/logo-horiz-rgb-lg-dkbg.gif" alt="`
Extractions:	chevrolet, **ford**, kia, **toyota**, scion, **nissan**, pontiac, cadillac, hyundai
Wrapper #3:	`<span class="dName">Curry [...]</span>`
Extractions:	chevrolet, **ford**, honda atlanta, honda, honda yorktown, acura, subaru chicopee, subaru, kia, **toyota**, scion, **nissan**, buick, pontiac, cadillac

3.3. Examples

Suppose car makers "Ford", "Nissan", and "Toyota" were provided as seeds, wrappers can be automatically constructed for each document by using the proposed wrapper construction algorithm. Figure 3 shows an example source page (of 'curryauto.com') and Table 1 shows the contexts that the algorithm selected for constructing wrappers from 'curryauto.com', with the symbol "[…]" representing the placeholder for an extracted entity. The entities extracted by the wrappers are also shown in Table 1. Here the boldfaced mentions are the seeds themselves.

4. The Ranker

The entity mentions extracted by wrappers may contain noisy entities, or entities that are rarely associated with the seeds by popular consensus on the web. For example, "honda atlanta" and "honda yorktown" extracted by Wrapper #3 in Table 1 are such entities; these are unlikely to be members of the user's target set. Since it is extremely difficult for machines to perfectly understand the information needs of users, we choose to rank the extracted entity mentions in the set presented to the users. In this section, we first analyze the problem of finding similarity between seeds and extracted mentions. We then propose a graph walk for ranking extracted mentions.

Table 2. Node and relation types

Source Type	Edge Relation	Target Type
seeds	find	document
document	derive	wrapper
	find^{-1}	seeds
wrapper	extract	mention
	derive^{-1}	document
mention	extract^{-1}	wrapper

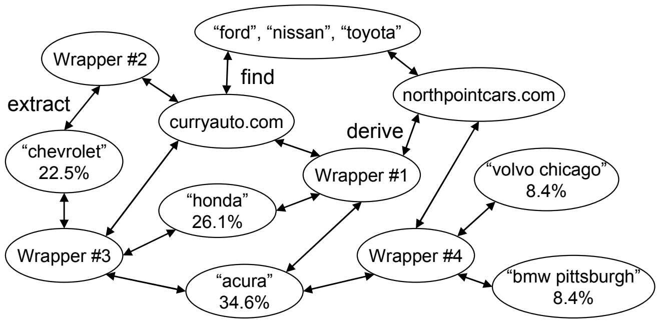

Figure 4. Example of a constructed graph

4.1. Problem Analysis

In order to determine the similarity between extracted mentions and seeds (or the likelihood that they all belong to the same class based on contextual information), we need to first understand how they are related globally. We know that seeds were used as a query to find documents online. We also know that the same wrapper may be derived from more than one document, and the same entity can be extracted by more than one wrapper. Also, we have observed that noisy entities are usually extracted less frequently than non-noisy entities. Intuitively, the more non-noisy entities extracted by a wrapper, the better quality the wrapper (and vise versa), and the more high-quality wrappers derived from a document, the better quality the document (and vise versa).

In order to model these complex relations, we will use a graph which contains all the objects of interest – seeds, web pages, wrappers, and extracted entity mentions. Similarity in the graph will then be used to rank entity mentions.

4.2. Building a Graph

A graph G consists of a set of nodes, and a set of labeled directed edges. We will use letters such as x, y, and z to denote nodes, and we will denote an edge from x to y with labeled relation r as $x \xrightarrow{r} y$. Every node x has a type and we will assume that there is a fixed set of possible node types. We will also assume, for convenience, that there are no edges from a node to itself; however, this assumption can be easily relaxed.

Each node represents an object, and each edge $x \xrightarrow{r} y$ asserts that that a binary relation $r(x, y)$ holds. The graph edges are directed. We also create an inverse relation $r^{-1}(x, y)$ for each edge; thus the graph will certainly be cyclic. The first column of Table 2 shows all possible source entity types, and the middle column shows each of their possible relations with some target entity types in the last column.

4.3. Graph Walk

We define the similarity between two nodes by a *lazy walk process*, similar to PageRank with decay. To walk away from a source node x, one first picks an edge relation r; then given r, one picks a target node y such that $x \xrightarrow{r} y$. We assume that, given a source node x, 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,

$$P(r \mid x) = \frac{1}{\left| r : \exists y \; x \xrightarrow{r} y \right|} \quad (1)$$

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

$$P(y \mid r, x) = \frac{1}{\left| y : x \xrightarrow{r} y \right|} \quad (2)$$

At each step in a lazy graph walk, there is also some probability λ of staying at x. Putting everything together, the probability of reaching any node z from x is computed recursively as follows:

$$P(z \mid x) = \lambda \cdot \mathrm{I}(x = z) + (1-\lambda)\sum_r \left[P(r \mid x) \sum_y P(y \mid r, x) P(z \mid y) \right] \quad (3)$$

where $\mathrm{I}(x = z)$ is a binary function that returns 1 if node x and node z are the same, and 0 otherwise.

In our experiments, we use a constant λ of 0.5, and we sample the graph by taking 10000 walks randomly,

Table 3. Explanation for each dataset (* are incomplete sets).

	Dataset	Eng	Chi	Jap	Class Description
1	classic-disney	✓	✓	✓	Classic Disney movie names
2	constellations	✓	✓	✓	Constellation names
3	countries	✓	✓	✓	Country names
4	mlb-teams	✓	✓	✓	Major League Baseball team names
5	nba-teams	✓	✓	✓	National Basketball Association team names
6	nfl-teams	✓	✓	✓	National Football League team names
7	popular-car-makers	✓	✓	✓	*Popular car manufacturer names
8	us-presidents	✓	✓	✓	United States president names
9	us-states	✓	✓	✓	United States state names
10	cmu-buildings	✓			Carnegie Mellon's building names
11	common-diseases	✓			*Common disease names
12	periodic-comets	✓			Periodic comet names
13	china-dynasties		✓		Chinese dynasty names
14	china-provinces		✓		Chinese province names
15	taiwan-cities		✓		Taiwanese city names
16	japan-emperors			✓	Japanese emperor names
17	japan-priministers			✓	Japanese priminister names
18	japan-provinces			✓	Japanese province names

Table 4. Statistics of datasets for English, Chinese, and Japanese.

Language	# Class (C)	# Entity (E)	# Mention (M)	Avg. E/C	Avg. M/E
English	12	1017	1461	85	1.4
Chinese	12	694	1677	58	2.4
Japanese	12	804	1174	67	1.5

each walk consists of up to 10 steps starting from the node with type 'seeds'. At the end of the graph walk, each node will have a probability, or weight, assigned, and we rank all nodes of type 'mention' by their assigned weights.

4.4. Example

An example of such a graph is illustrated in Figure 4. A graph walk has been performed on this graph where each node is assigned a weight due to the walk. As expected, a walk on the graph in Figure 4 would weigh "bmw pittsburgh" and "volvo chicago" the least among the extracted mentions because these nodes have fewer incoming edges; thus they are harder to reach. The weights assigned to these mentions are shown on the example graph as well.

5. Datasets

There are a total of 36 evaluation datasets, constructed evenly over three languages: English, Chinese, and Japanese; thus there are 12 datasets per language. The datasets consist of 18 classes, where half were constructed in all three languages and the other half in one language only, as illustrated in Table 3. The intention is to diversify the datasets such that some are culture-specific while some are not. Each dataset is a plain text file that represents a particular class C, and each entity $e \in C$ is represented by a list of true mentions, or synonyms, of that particular e. The statistics of classes, entities, and entity mentions for each language are shown in Table 4.

6. Experiments

In this section, we describe our baseline system, alternative methods we attempted to use, evaluation metric and procedure, experimental results, and finally, comparisons of our results with those published by other researchers.

6.1. Baseline System

We choose Google Sets as our baseline system, mainly because it is well-known and publicly available. However, since Google Sets does not handle languages other than English, it is only directly comparable to SEAL on the English evaluation dataset. To our knowledge there is no set expansion system that can handle Chinese and/or Japanese, with which we could compare our evaluation results.

6.2. Alternative Methods

We conducted ablation studies using alternative methods for set expansion. The extraction approach described in section 3.2 was simplified. In the definition of *full* suffix of the wrapper construction algorithm, instead of finding all possible common suffixes of left context L and prefixes of right context R that embed *at least one instance of every seed*, it finds common suffixes of L and prefixes of R that embed *all seed instances*. More specifically, we let:

$$\text{FullSuffix}(L_1,\ldots,L_k) = \{x : \forall i\ x \text{ is a suffix of all } l_{i,j} \in L_i\}$$

This simple extractor, referred to as E1, is compared to the proposed extractor, referred to as E2, in the experimental results section.

The ranking approach described in section 4.2 was also simplified: instead of a graph walk, it ranks entity mentions by their frequency counts of being extracted by any wrapper from any document. This simple ranking scheme is referred to as EF (extracted frequency) and the proposed method is referred to as GW (graph walk) in the experimental results section.

6.3. Evaluation

Since the output of SEAL is a ranked list of extracted mentions, we choose mean average precision (MAP) as the evaluation metric. MAP has been commonly used for evaluating ranked lists in the field of Information Retrieval. It contains both recall and precision-oriented aspects, and it is sensitive to the entire ranking. For evaluating a system that produces multiple ranked lists, MAP is simply the mean value of average precisions computed for each ranked list separately. Average precision of a single ranked list is defined as:

$$\text{AvgPrec}(L) = \frac{\sum_{r=1}^{|L|} \text{Prec}(r) \times \text{NewEntity}(r)}{\#\,\text{True Entities}} \tag{4}$$

where L is a ranked list of extracted mentions, r is the rank, Prec(r) is the precision at rank r. NewEntity(r) is a binary function, which returns 1 if a) the extracted mention at r matches any true mention, and b) there exist no other extracted mention at rank less than r that is of the same entity as the one at r. It returns 0 otherwise.

The procedure for evaluating SEAL is that, for each dataset:

1. Randomly select three true entities and use their first listed mentions as seeds.
2. Expand the three seeds obtained from step 1.
3. Repeat steps 1 and 2 five times.
4. Compute MAP for the five resulting ranked lists.

6.4. Experimental Results

Table 5 shows our experimental results. The baseline Google Sets (G.Sets) performed the worst. Even our simplest approach, E1+EF, beats Google Sets by a substantial amount. In our first set of experiments, we requested only top 100 URLs per expansion from Google. Our simplest approach, E1+EF, achieved an overall average of around 82%. After improving E1 to E2, we observed a 6.34% improvement on the overall average. We then enhanced EF to GW, and observed a 6.30% improvement. We decided to increase our corpus size to see if any improvements can still be made. Rather than requesting only top 100 URLs, we increase the number to 200 and 300, and we observed a slight improvement of 0.97% and 0.16% respectively.

6.5. Set Comparisons

We present side-by-side comparison of set expansion results published by other researchers and obtained by SEAL. Table 6 illustrates the top 42 set expansion results on watch brand names using 17 seeds as presented in Talukdar et al [8]. As comparison, we present our top 42 results in the right column using only the first three of their 17 seeds (namely Corum, Longines, and Lorus). SEAL achieved a precision of 100% on those results; whereas Talukdar's system returned noisy entities (boldfaced mentions) towards the bottom of their list and achieved a precision of 85.7%. We also tried the three seeds on Google Sets but obtained no results other than the seeds themselves. Table 7 shows top 10 set expansion results on children's movies from Bayesian Sets, as presented in Ghahramani et al [9]. We provide results from Google Sets and SEAL as comparisons. Note that Bayesian Sets used a movie-specific dataset: EachMovie. As illustrated, both Bayesian Sets and SEAL systems perform well on finding children movies. Similar to Table 7, Table 8 shows top 10 results on NIPS authors from Bayesian Sets as presented in Ghahramani et al [9]. We provide results from SEAL as comparisons but not from Google Sets since it failed to return any result. Note that Bayesian Sets, again, used a domain-specific dataset: the NIPS dataset.

Table 5. Experimental results

		Max. 100 Results			Max. 200	Max. 300
English	**G.Sets**	**E1+EF**	**E2+EF**	**E2+GW**	**E2+GW**	**E2+GW**
classic-disney	37.62%	79.36%	74.45%	84.42%	88.20%	89.39%
cmu-buildings	0.00%	87.85%	87.75%	87.83%	87.83%	87.83%
common-diseases	1.12%	17.94%	52.84%	57.46%	75.79%	76.87%
constellations	10.45%	89.61%	99.97%	100.00%	100.00%	100.00%
countries	14.24%	95.95%	97.86%	98.17%	98.67%	98.53%
mlb-teams	70.06%	98.61%	99.50%	99.80%	99.84%	99.81%
nba-teams	90.73%	100.00%	100.00%	100.00%	100.00%	100.00%
nfl-teams	94.26%	99.22%	99.98%	100.00%	100.00%	100.00%
periodic-comets	0.22%	69.24%	79.04%	84.78%	84.77%	84.77%
popular-car-makers	73.61%	79.18%	88.23%	95.16%	96.23%	96.95%
us-presidents	56.77%	91.64%	97.07%	99.99%	100.00%	100.00%
us-states	76.00%	99.96%	93.55%	100.00%	100.00%	100.00%
Average	**43.76%**	**84.05%**	**89.19%**	**92.30%**	**94.28%**	**94.51%**
Trad. Chinese	**G.Sets**	**E1+EF**	**E2+EF**	**E2+GW**	**E2+GW**	**E2+GW**
china-dynasties	-	25.45%	33.86%	65.20%	64.62%	65.22%
china-provinces	-	94.97%	99.19%	99.21%	99.34%	99.35%
classic-disney	-	80.73%	91.17%	91.68%	91.68%	91.68%
constellations	-	92.00%	96.25%	99.99%	99.99%	99.99%
countries	-	94.79%	95.39%	96.94%	97.76%	97.72%
mlb-teams	-	94.42%	84.05%	99.98%	99.96%	99.96%
nba-teams	-	90.29%	95.04%	99.90%	100.00%	100.00%
nfl-teams	-	68.08%	88.43%	95.75%	95.75%	95.75%
popular-car-makers	-	71.44%	83.29%	94.36%	94.47%	94.55%
taiwan-cities	-	95.26%	98.04%	100.00%	100.00%	100.00%
us-presidents	-	62.84%	82.61%	93.03%	94.24%	94.24%
us-states	-	98.47%	97.08%	99.48%	99.48%	99.49%
Average	**-**	**80.73%**	**87.03%**	**94.63%**	**94.77%**	**94.83%**
Japanese	**G.Sets**	**E1+EF**	**E2+EF**	**E2+GW**	**E2+GW**	**E2+GW**
classic-disney	-	72.83%	75.00%	81.65%	83.00%	82.95%
constellations	-	96.87%	95.18%	99.99%	100.00%	100.00%
countries	-	97.33%	90.47%	98.69%	99.18%	99.16%
japan-emperors	-	95.93%	99.10%	99.24%	99.24%	99.24%
japan-priministers	-	71.25%	91.75%	93.12%	93.00%	93.03%
japan-provinces	-	99.40%	100.00%	100.00%	100.00%	100.00%
mlb-teams	-	80.00%	85.60%	98.89%	98.91%	98.91%
nba-teams	-	95.28%	99.44%	99.96%	99.98%	99.98%
nfl-teams	-	92.83%	93.86%	99.05%	99.05%	99.06%
popular-car-makers	-	53.64%	76.13%	79.57%	84.81%	86.47%
us-presidents	-	36.44%	34.61%	59.46%	59.46%	59.46%
us-states	-	96.95%	98.02%	99.92%	99.94%	99.98%
Average	**-**	**82.40%**	**86.60%**	**92.46%**	**93.05%**	**93.19%**
Overall Average	**43.76%**	**82.39%**	**87.61%**	**93.13%**	**94.03%**	**94.18%**

Table 6. Comparison: Watch Brands

Talukdar et al	SEAL
Rolex	Omega
Fossil	Cartier
Swatch	Seiko
Cartier	Tag Heuer
Tag Heuer	Ebel
Super Bowl	Rado
Swiss	Gucci
Chanel	Bulova
SPOT	Raymond Weil
Movado	Movado
Tiffany	Citizen
Sekonda	Breitling
Seiko	Tissot
TechnoMarine	Pulsar
Rolexes	Fossil
Gucci	Hamilton
Franck Muller	Rolex
Harry Winston	Casio
Patek Philippe	Swatch
Versace	Concord
Hampton Spirit	Swiss Army
Piaget	Wittnauer
Raymond Weil	Nike
Girard Perregaux	Oris
Omega	Chopard
Guess	IWC
Frank Mueller	DKNY
Citizen	Wenger
Croton	Piaget
David Yurman	Timex
Armani	ESQ
Audemars Piguet	Guess
Chopard	Patek Philippe
DVD	Croton
DVDs	Tommy Hilfiger
Chinese	Sector
Breitling	Invicta
Montres Rolex	Oakley
Armitron	Skagen
Tourneau	Anne Klein
CD	Armitron
NFL	Zodiac

7. Related Work

Google Sets uses a proprietary method that has not been published. The KnowItAll system [4] contains a List Extractor (LE) component that is functionally similar to Google Sets. It uses an HTML parser for identifying sub-trees of a parsed web page. For each selected sub-tree, it finds one contextual pattern that maximally matches all of the seeds. However, SEAL does not require any parsing, and it finds all contextual patterns in the whole document that maximally match at least one instance of every seed.

Etzioni et al [4] describe a number of possible variants of the LE component, but it is not clear which variant was used in their experiment. The KnowItAll system achieved precisions of 23~79% on four sample problems.

Several researchers have studied set expansion using free text rather than semi-structured Web documents; for instance, Talukdar et al [8] present a method for automatically selecting *trigger words* to

Table 7. Comparison: Children's Movies

Seeds: Mary Poppins, Toy Story		
Google Sets	**Bayesian Sets**	**SEAL**
Toy Story	Mary Poppins	Mary Poppins
Mary Poppins	Toy Story	Toy Story
Mulan	Winnie the Pooh	Cinderella
Toy Story 2	Cinderella	Hercules
Moulin Rouge	The Love Bug	The Lion King
Monsters Inc	Bedknobs and Broomsticks	Pocahontas
Man on the Moon	Davy Crockett	Pinocchio
Mummy The	The Parent Trap	Beauty and the Beast
Matrix The	Dumbo	The Jungle Book
Mod Squad The	The Sound of Music	Song of the South

Table 8. Comparison: NIPS Authors

Seeds: L. Saul, T. Jaakkola	
Bayesian Sets	**SEAL**
L. Saul	T. Jaakkola
T. Jaakkola	L. Saul
M. Rahim	B. Frey
M. Jordan	P. Niyogi
N. Lawrence	M. J. Wainwright
T. Jebara	C. Bishop
W. Wiegerinck	M. I. Jordan
M. Meila	Z. Ghahramani
S. Ikeda	A. Smola
D. Haussler	Y. Weiss

mark the beginning of a pattern, which is then used for bootstrapping from free text.

Ghahramani et al [9] illustrates a Bayesian Sets algorithm that solves a particular sub-problem of set expansion, in which candidate sets are given, rather than a corpus of web documents. We intend to compare their ranking method with graph-walks in future experiments.

8. Conclusion and Future Work

In this paper, we have presented a novel and effective approach for expanding sets of named entities in an unsupervised, domain and language independent fashion. We have shown that our system, SEAL, performs better than Google Sets in terms of mean average precision for the dataset tested. We have also shown that our system is capable of handling various languages such as English, Chinese, and Japanese which Google Sets does not.

There are several future topics of research that we are currently considering. First, we will investigate bootstrapping of named entities by performing several rounds of expansions, where each round uses the top extracted mentions from the previous round as seeds. Second, we will explore re-ranking of web documents based on their contained set of mentions extracted from previous round of expansion. Third, we will look into automatic identification of possible class names for expanded sets. Lastly, we will study the possibility of hierarchical clustering on the expanded sets and graph learning for ranking candidate entities.

Acknowledgements

We want to thank Robert E. Frederking and Eric Nyberg for proof-reading this paper. This material is based upon work supported by the Defense Advanced Research Projects Agency (DARPA) under Contract No. NBCHD030010. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Defense Advanced Research Projects Agency (DARPA).

References

[1] B. Settles, "Biomedical Named Entity Recognition Using Conditional Random Fields and Rich Feature Sets," in *NLPBA/BioNLP*, 2004.

[2] J. Prager, J. Chu-Carroll, and K. Czuba, "Question Answering using Constraint Satisfaction: QA-by-Dossier-with-Constraints," in *ACL*, 2004.

[3] M. J. Cafarella, D. Downey, S. Soderland, and O. Etzioni, "KnowItNow: Fast, Scalable Information Extraction from the Web," in *EMNLP*, 2005.

[4] O. Etzioni, M. Cafarella, D. Downey, A.-M. Popescu, T. Shaked, S. Soderland, D. S. Weld, and A. Yates, "Unsupervised Named-Entity Extraction from the Web: An Experimental Study," *Artificial Intelligence*, vol. 165, pp. 91-134, 2005.

[5] W. W. Cohen, "Automatically Extracting Features for Concept Learning from the Web," in *ICML*, 2000.

[6] W. W. Cohen and W. Fan, "Learning Page-Independent Heuristics for Extracting Data from Web Pages," in *WWW*, 1999.

[7] S. Kambhampati, G. Wolf, Y. Chen, H. Khatri, B. Chokshi, J. Fan, and U. Nambiar, "QUIC: Handling Query Imprecision & Data Incompleteness in Autonomous Databases," in *CIDR*, 2007.

[8] P. P. Talukdar, T. Brants, M. Liberman, and F. Pereira, "A Context Pattern Induction Method for Named Entity Extraction," in *Computational Natural Language Learning (CoNLL-X)*, 2006.

[9] Z. Ghahramani and K. A. Heller, "Bayesian Sets," in *Advances in Neural Information Processing Systems*, 2005.