Iterative Set Expansion of Named Entities using the Web

Richard C. Wang and William W. Cohen
Language Technologies Institute
Carnegie Mellon University
5000 Forbes Avenue, Pittsburgh PA 15213
{rcwang,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 a previous study, SEAL showed good set expansion performance using three seed entities; however, when given a larger set of seeds (e.g., ten), SEAL's expansion method performs poorly. In this paper, we present Iterative SEAL (iSEAL), which allows a user to provide many seeds. Briefly, iSEAL makes several calls to SEAL, each call using a small number of seeds. We also show that iSEAL can be used in a "bootstrapping" manner, where each call to SEAL uses a mixture of user-provided and self-generated seeds. We show that the bootstrapping version of iSEAL obtains better results than SEAL using fewer user-provided seeds. In addition, we compare the performance of various ranking algorithms used in iSEAL, and show that the choice of ranking method has a small effect on performance when all seeds are user-provided, but a large effect when iSEAL is bootstrapped. In particular, we show that Random Walk with Restart is nearly as good as Bayesian Sets with user-provided seeds, and performs best with bootstrapped seeds.

1. Introduction

Have you ever wanted to find out the names of reality TV shows similar to the ones you regularly watch? A *set expansion system* takes as inputs a few example seeds (e.g., Survivor) of a user-desired class (e.g., reality TV shows) and outputs more examples of that class (e.g., The Apprentice). More specifically, a user issues a query consisting of a small number of seeds $x_1, x_2, ..., x_k$ where each x_i is a member of some target set S_t. The answer to the query is a listing of other probable entities of S_t.

	English	Chinese	Japanese
input seed	Amazing Race	豬血糕	ドラえもん
	Survivor	臭豆腐	ハローキティ
output	Big Brother	蘿蔔糕	アンパンマン
	The Mole	蚵仔煎	シナモロール
	The Apprentice	肉圓	ウルトラマン
	Project Runway	滷肉飯	スヌーピー
	The Bachelor	春捲	くまのプーさん

Figure 1: Real examples of SEAL's input and output. English items are reality TV shows, Chinese are popular Taiwanese food, and Japanese are famous cartoon characters.

A well-known example of a web-based set expansion system is Google Sets[1]. It has been used for numerous purposes, including deriving features for named entity recognition [6] and evaluation of question answering systems [5]. However, it is a proprietary method that may be changed at any time, so research results based on Google Sets cannot be reliably replicated.

Another web-based set expansion system is Set Expander for Any Language[2] (SEAL) [9]. As its name implies, SEAL is independent of document languages: both the written language (e.g., English) and the markup language (e.g., HTML). SEAL is a research system that has shown good performance in previously published results. By using only three seeds and the top one hundred documents returned by Google, SEAL achieved 93% in mean average precision (MAP), averaged over 36 datasets from three languages: English, Chinese, and Japanese. Unlike other published research work [1], SEAL focuses on finding small closed sets of entities (e.g., Disney movies) rather than large and more open sets (e.g., scientists). Figure 1 shows examples of SEAL's input and output. In more detail, SEAL contains three major components: Fetcher, Extractor, and Ranker. The Fetcher fetches one hundred web pages, each containing all seeds, by querying Google. The

[1] http://labs.google.com/sets
[2] http://rcwang.com/seal

Ranker \ # Seeds	2	3	4	5	6
Random Walk	77.1	83.9	84.5	83.7	78.9
Page Rank	74.1	82.6	83.4	83.0	78.5
Bayesian Sets	77.0	84.1	84.8	84.0	79.3
Wrapper Length	77.5	83.2	83.3	82.2	78.0
Average	76.4	83.5	84.0	83.2	78.7

Table 1: MAP of set expansion using various rankers and various number of seeds. Note that four seeds maximize the performance.

Extractor constructs extraction rules or *wrappers* for each web page. The Ranker builds a graph that models all relations between pages, wrappers, and extracted mentions. Nodes in the graph are then given weights, using one of several *ranking methods*, which we will discuss later.

Although SEAL works well given three or four seeds, it has a limitation on the number of seeds it can handle. Table 1 shows the performance of SEAL (as the MAP score averaged across 36 datasets) for four different ranking methods when provided with two to six *supervised seeds* (i.e., correct seeds randomly selected from our development set, as a proxy for user-provided seeds). When SEAL is given more than five supervised seeds, its performance drops substantially. SEAL's first step is to retrieve web pages containing all seeds, but few web pages contain more than five seeds. To overcome this limitation, we will propose the iSEAL method, which is an (supervised) iterative process that performs *supervised expansion* multiple times. In each iteration, iSEAL invokes the SEAL method on a few supervised seeds, and statistics are accumulated from iteration to iteration to obtain a final ranking.

The ability to use many seeds enables *bootstrapping* - an (unsupervised) iterative process in which a system continuously consumes its own outputs to improve its own performance [1, 3]. We propose a bootstrapping technique that requires only two supervised seeds, which are used to trigger the first expansion of the iterative process above. In each iteration after the first, iSEAL expands a few *unsupervised seeds* (i.e., highly ranked items obtained in the previous iteration of iSEAL), and again statistics are accumulated.

Bootstrapping introduces a potential problem, as the self-provided seeds used in bootstrapping may be incorrect, and prior results do not indicate how SEAL performs with "noisy" seeds. We show that iSEAL, when used in bootstrap mode, is indeed much more sensitive to the choice of ranking method and number of seeds. We compare several ranking methods, including Random Walk with Restart [7], which is similar to that used in the previously published version of SEAL; PageRank [4], which was designed to rank hyperlinked documents; Bayesian Sets [2], which formulates the set expansion problem as a Bayesian inference problem; and a fast but *ad hoc* ranking heuristic we call Wrapper Length.

Below, Section 2 presents Iterative SEAL. Section 3 presents the ranking methods. Section 4 describes the experimental design, and Section 5 presents the experimental results. The paper concludes in Section 6.

2. Iterative SEAL

In this section, we present the Iterative SEAL (iSEAL) system and consider two different iterative processes: supervised expansion and bootstrapping. In our experiments, both processes start their first iteration with two supervised seeds, which is the smallest number of seeds required by the wrapper induction technique used in SEAL. In every successive iteration, there are several ways to select seeds. For each process, we propose two seeding (i.e., seed selection) strategies: Fixed Seed Size (FSS) and Increasing Seed Size (ISS).

2.1 Iterative Supervised Expansion

The iterative supervised expansion improves SEAL's performance by allowing it to handle unlimited number of supervised seeds. In each iteration, it expands a couple of randomly selected seeds while accumulating statistics from one iteration to another. This allows the expansion of seeds in the current iteration to have access to all statistics computed in the past iterations.

We present two seeding strategies for this process below. The strategy *Fixed Seed Size* (FSS) requires two seeds in every iteration. Below is the pseudo-code for this strategy:

$stats \leftarrow \emptyset$
for i = 1 to M **do**
  $seeds \leftarrow select_2(E)$
  $stats \leftarrow expand_{stats}(seeds)$
  $ranked\_list \leftarrow rank_r(stats)$
end for

where M is the total number of iterations (inclusively), $select_n(E)$ randomly selects n different seeds from the set E, E is a set containing supervised seeds, $expand_{stats}(seeds)$ expands the selected $seeds$ using $stats$ and outputs accumulated statistics, and $rank_r(stats)$ applies the ranker r on the accumulated $stats$ to produce a $ranked\_list$ of entities.

Increasing Seed Size (ISS) is a strategy that starts the iterative process with two supervised seeds, then increments the number of seeds by one for every successive expansion, until a maximum size of n is reached. After then, it continues to expand using n seeds. We set n to be four based on results in Table 1, which shows that four seeds maximize the performance. This number has also been reported by Etzioni et al. [1] and Nadeau et al. [3]. The pseudo-code for this strategy is presented in the following page:

$stats \leftarrow \emptyset, used \leftarrow \emptyset$
for i = 1 to M **do**
  if $i = 1$ **then**
    $seeds \leftarrow select_2(E)$
  else
    $m = min(3, |used|)$
    $seeds \leftarrow select_m(used) \cup \underline{select_1(E)}$
  end if
  $used \leftarrow used \cup seeds$
  $stats \leftarrow expand_{stats}(seeds)$
  $ranked\_list \leftarrow rank_r(stats)$
end for

where $used$ is a set that contains previously expanded seeds and $min(x, y)$ returns the minimum of x and y. This strategy starts by expanding two supervised seeds. For the second iteration, it expands three seeds: two *used* plus one *new* supervised seed. For every successive iteration, it expands four seeds: three randomly selected used seeds plus one new supervised seed.

2.2 Bootstrapping

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. Carefully designed seeding strategies can minimize the propagated errors. We present the two seeding strategies in the unsupervised mode.

As mentioned earlier, FSS is a strategy that requires two seeds in every iteration. Unlike supervised expansion, bootstrapping expands (unsupervised) seeds that are the most confident new pair of entities (i.e., an entity pair that has never been used as seeds) extracted from the last iteration. The pseudo-code is presented below:

$stats \leftarrow \emptyset$
for i = 1 to M **do**
  if $i = 1$ **then**
    $seeds \leftarrow select_2(E)$
  else
    $seeds \leftarrow top\_pair(ranked\_list)$
  end if
  $stats \leftarrow expand_{stats}(seeds)$
  $ranked\_list \leftarrow rank_r(stats)$
end for

where $top\_pair(ranked\_list)$ returns a new pair of entities which has the highest joint probabilistic weights according to their confidence scores in $ranked\_list$. More specifically, for every successive i^{th} iteration after the first expansion, this strategy selects from the results of $(i-1)^{th}$ iteration a new pair of entities to be used as seeds for the i^{th} iteration. Regardless of the number of iterations, the supervised seeds required are only those two initial seeds.

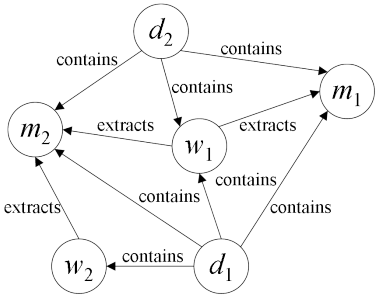


Figure 2: Example graph constructed by Random Walk. Every edge from node x to y has an inverse relation edge from node y to x that is not shown here (i.e. m_i is extracted by w_i).

In the unsupervised mode of ISS, the strategy is exactly the same as in the supervised mode, except that after the first iteration, the new seed (i.e., the entity never used as seed) at every i^{th} iteration is the highest-ranked new entity in $(i-1)^{th}$ iteration. More precisely, the pseudo-code of this strategy is that of supervised ISS with the underlined part replaced with $top\_one(ranked\_list)$, which returns a new entity that has the highest weight in $ranked\_list$. Again, the two initial seeds are the only supervised seeds required.

3. Ranking Methods

In the last section, we presented two iterative processes where each has two seeding strategies. In all four pseudo-codes presented above, there is a $rank_r(stats)$ that ranks entities based on the accumulated $stats$ using ranker r. In this section, we present the rankers used in our experiments.

3.1. Random Walk with Restart

Wang and Cohen [9] presented a graph-walk based model that is effective for solving the set expansion problem. This model encapsulates the relations between documents, wrappers, and extracted 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. Every edge in the graph also has an inverse relation edge (i.e. m_i is contained by d_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 similar-

ity between two nodes by random walk with restart [7]. 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,

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

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,

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

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,

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

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:

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

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 that shows good performance in our development set. New statistics can be accumulated easily by adding additional nodes and edges to the existing graph.

3.2. PageRank

Page et al. [4] proposed the PageRank algorithm that is being used extensively at Google to score web pages. Although it was designed to rank hyperlinked set of documents (i.e., web pages), it can also be used to rank other elements [8, 10]. The graph that we use for PageRank is identical to the one shown in Figure 2, except that edges are undirected and they do not have relations. New statistics can be easily accumulated by attaching new nodes and edges to the graph. Page et al. [4] uses a teleporting probability λ of 0.15, which we also use in our experiments. We iterate the graph until all node weights converge, and rank the extracted mentions based on their final node weights.

3.3. Bayesian Sets

Ghahramani and Heller [2] proposed the Bayesian Sets algorithm that formulates the set expansion problem as a Bayesian inference problem. It uses a model-based concept of a class and ranks items using a score which evaluates the marginal probability that each item belongs to the class containing the seed items. We implemented Bayesian Sets, by constructing one large two-dimensional feature table where each column represents a feature, each row an extracted item, and each entry (j, k) indicates item x_k's possession of the feature f_j. We incorporate two features: document containment and wrapper extraction. For example, if an item is contained by a document d_j or was extracted by a wrapper w_j, then entry (j, k) would be 1, otherwise 0. We tried using either one of the features alone on our development set but the results are worse. New statistics can be accumulated by appending new rows and columns to the feature table.

3.4. Wrapper Length

SEAL defines a wrapper as a pair of maximally-long strings (l, r) that bracket at least one occurrence of every seed on a web page. We have observed that longer strings are generally better. Therefore, we propose a simple, fast but ad hoc ranking algorithm called Wrapper Length as detailed below:

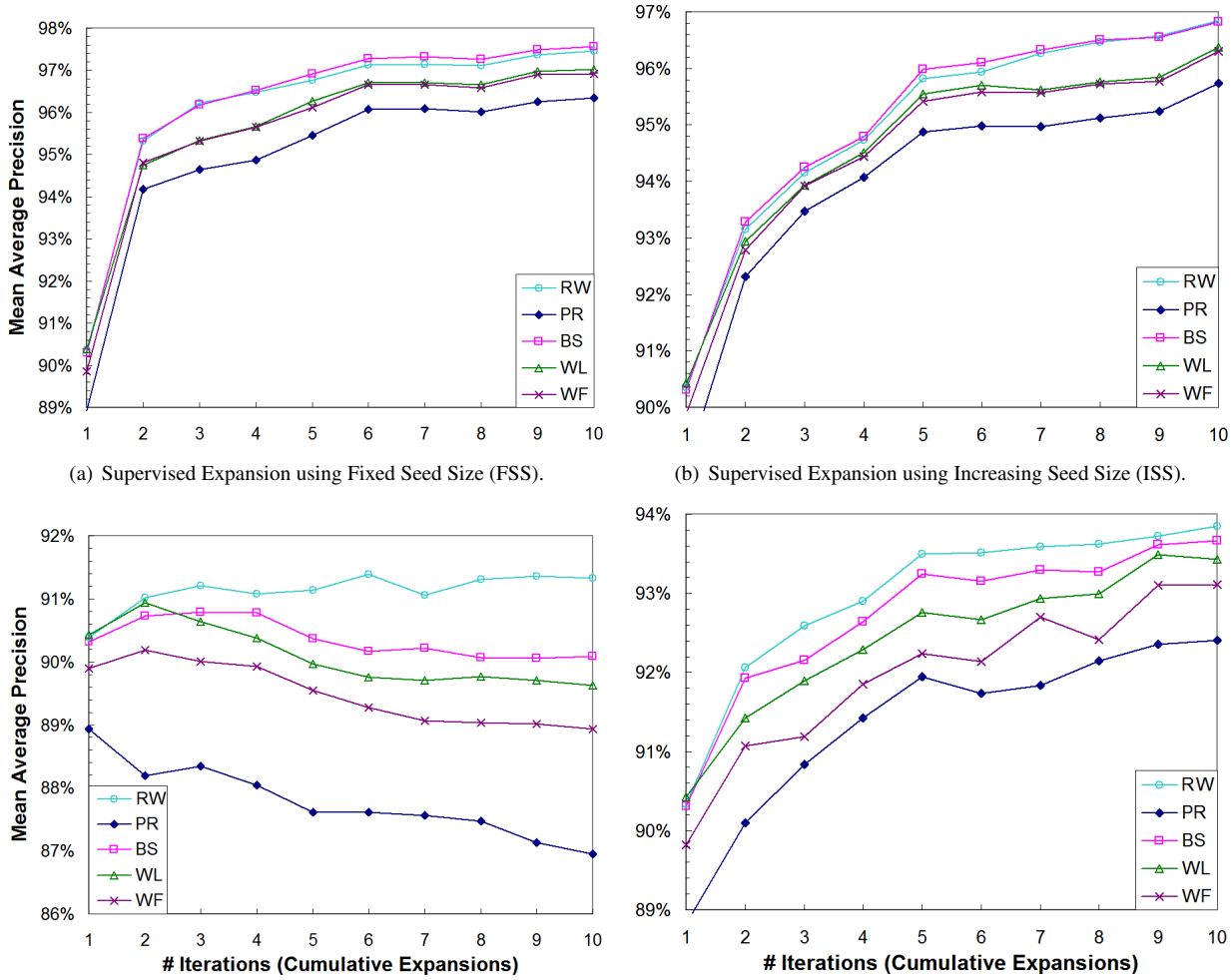
$$\log score(x) = \sum_{j \text{ extracts } x} \log(length(w_j))$$

where w_j is the j^{th} wrapper composed of a pair of left and right contextual strings, and the function *length* returns the sum of the character lengths of those pair of strings in w_j. This heuristic is based on the assumption that an item should have a high score if it is extracted by many long wrappers.

4. Experimental Setting

We evaluate the iterative processes using the datasets[3] presented in Wang and Cohen [9], which consists of 36 manually constructed lists across three different languages: English, Chinese, and Japanese (12 lists per language). In the lists, each entity is represented by a set of synonyms or "mentions" (e.g., USA, America). There are a total of 2515

[3]Available at `http://rcwang.com/papers/dataset/seal-data.zip`



(a) Supervised Expansion using Fixed Seed Size (FSS).

(b) Supervised Expansion using Increasing Seed Size (ISS).

(c) Bootstrapping using Fixed Seed Size (FSS).

(d) Bootstrapping using Increasing Seed Size (ISS).

Figure 3: MAP of using various iterative methods and rankers: Random Walk (RW), PageRank (PR), Bayesian Sets (BS), Wrapper Length (WL), and Wrapper Frequency (WF).

entities and 4312 mentions in this dataset, with an average of 70 entities per list and 1.7 mentions per entity.

Since the output of iSEAL is a ranked list of extracted mentions, we choose mean average precision (MAP) as the evaluation metric because it is sensitive to the entire ranking and it contains both recall and precision-oriented aspects. The MAP for multiple ranked lists is the mean value of average precisions calculated separately for each ranked list. We define the average precision of a single ranked list as:

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

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

As a baseline, we introduce in the experimental results a fifth ranker called Wrapper Frequency, which is simply the number of wrappers that extract a particular entity. Each experiment evaluates a particular combination of iterative process, seeding strategy, and ranker; we evaluated all 20 possible combinations. For each combination, we performed ten iterative expansions on each of the 36 evaluation datasets independently three times; thus, at each of the ten iterations, there are 108 (3×36) ranked lists. We then report the MAP of those ranked lists in the next section.

5. Experimental Results

We first examine the effect of supervised expansion. Figure 3(a) and 3(b) illustrate the performance of various rankers using supervised expansion with FSS and ISS respectively. Although both strategies improve the performance of the rankers, FSS improves faster than ISS. The reason is that FSS requires two new supervised seeds at every iteration whereas ISS requires only one. From the two graphs, we observe that Bayesian Sets (BS) performs the best, Random Walk with Restart (RW) is nearly as good as Bayesian Sets, Wrapper Frequency (WF) is almost as good as Wrapper Length (WL), and PageRank (PR) is the worst among the five rankers.

We then examine the effect of bootstrapping. Figure 3(c) and 3(d) illustrate the performance of various rankers in bootstrap mode using FSS and ISS respectively. As illustrated, ISS reliably improves with more seeds, but FSS failed to improve any ranker other than Random Walk at the 10^{th} iteration, and even this improvement was modest. This result shows that bootstrapping set expansion is not a trivial task. The ISS strategy was carefully designed to circumvent this performance problem. While ISS uses only two supervised seeds, it is much more conservative about using self-generated seeds: at every expansion, FSS boldly introduces two new seeds taken from the results of the last iteration, whereas ISS conservatively introduces only one. Therefore, the chance of FSS selecting an incorrect entity as seed is higher than that of ISS. Furthermore, in every iteration, ISS has three prior seeds to support the newly chosen one, which minimizes the chance of expanding seeds that are all incorrect.

Figure 3(c) shows that Random Walk is the most robust of the five rankers, followed by Bayesian Sets and Wrapper Length. While all rankers performed poorly with noisy seeds, only Random Walk improved (slightly at the 10^{th} iteration). Figure 3(d) shows that the performance of Random Walk increases monotonically when bootstrapping using ISS. It also shows that Random Walk has the best performance, followed by Bayesian Sets and Wrapper Length.

The graphs show that Wrapper Length is also an effective ranking algorithm. It is comparable to the baseline Wrapper Frequency in supervised mode and is always better in the bootstrap mode. We want to emphasize that Wrapper Length is a very simple and light-weight algorithm; the memory space needed is proportional only to the number of extracted items. Furthermore, the results suggest that if many supervised seeds are available, then supervised expansion using FSS should be considered. If only a few are available, then bootstrapping using ISS should be considered. In terms of rankers, both Bayesian Sets and Random Walk are good choices for supervised expansion, and Random Walk is the best choice for bootstrapping.

6. Conclusion

In this paper, we have presented a system called Iterative SEAL (iSEAL) to examine various iterative processes and seeding strategies using different rankers for the problem of set expansion. We have shown that the performance of SEAL can improve monotonically if we bootstrap the results using ISS and rank the results using Random Walk with Restart. By using this method and only two seeds, the final result (94%) is even better than that of using three supervised seeds (with same amount of web pages) as published in previous results of SEAL (93%). We have also shown that in supervised mode, Random Walk is comparable to the best ranker (Bayesian Sets), but in bootstrap mode, Random Walk is the best due to its robustness to noisy seeds. We have also presented a simple and light-weight ranker, Wrapper Length, that shows good performance in most experiments.

Acknowledgements

This work was supported in part by the Google Research Awards program and the Defense Advanced Research Projects Agency (DARPA) under Contract No. NBCHD030010.

References

[1] O. Etzioni, M. J. 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. *Artif. Intell.*, 165(1):91–134, 2005.

[2] Z. Ghahramani and K. A. Heller. Bayesian sets. In *NIPS*, 2005.

[3] D. Nadeau, P. D. Turney, and S. Matwin. Unsupervised named-entity recognition: Generating gazetteers and resolving ambiguity. In *Canadian Conference on AI*, 2006.

[4] L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank citation ranking: Bringing order to the web. Technical report, Stanford Digital Library Tech. Project, 1998.

[5] J. M. Prager, J. Chu-Carroll, and K. Czuba. Question answering using constraint satisfaction: Qa-by-dossier-with-contraints. In *ACL*, pages 574–581, 2004.

[6] B. Settles. Biomedical named entity recognition using conditional random fields and rich feature sets. In *COLING 2004 International Joint workshop on NLPBA/BioNLP*.

[7] H. Tong, C. Faloutsos, and J.-Y. Pan. Fast random walk with restart and its applications. In *ICDM*, pages 613–622. IEEE Computer Society, 2006.

[8] J. Wang, J. Liu, and C. Wang. Keyword extraction based on pagerank. In *PAKDD*, pages 857–864, 2007.

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

[10] C. C. Yang and K. Y. Chan. Retrieving multimedia web objects based on pagerank algorithm. In *WWW*, 2005.