Iterative Set Expansion of Named Entities using the Web

Abstract

Set expansion refers to expanding a given 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 an 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, an 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.

A well-known example of a web-based set expansion system is Google Sets[1]. Google Sets has been used for numerous purposes, including deriving features for named entity recognition [7] and evaluation of question answering systems [6]. However, it is a proprietary method that may be changed at any time, so research results based on Google Sets cannot be reliably replicated.

		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.

Another web-based set expansion system is Set Expander for Any Language[2] (SEAL) [10]. As its name implies, SEAL is independent of document language: both written language (e.g., English) and 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 over 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 some real examples of SEAL's input and output in various languages.

In more detail, SEAL contains three major components: Fetcher, Extractor, and Ranker. The Fetcher is responsible for fetching top one hundred web pages from Google by issuing a seed query, which is simply the concatenation of all seeds. This ensures that every fetched web page contain all seeds; however, it creates a problem that we will discuss later. The Extractor takes those fetched web pages and auto-

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

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

matically constructs page-specific extraction rules, or *wrappers*, for each page. Every wrapper comprises two character strings that specify the left and right contexts necessary for extracting entity mentions. These contextual strings are maximally-long contexts that bracket at least one occurrence of every seed string on a page. All entity mentions bracketed by these contextual strings derived from a page are extracted from that page.

After the entity mentions are extracted, the Ranker builds a graph that models all the relations between documents, wrappers, and extracted mentions. Figure 3 shows an example graph, which is explained more fully in Section 3.1. Nodes in the graph are then given weights, using one of several *ranking methods*, which we will discuss later; the general idea, however, is that nodes should be weighted higher if they are connected to many other highly weighted (important) nodes. The final expanded set contains all extracted mentions in the graph, ranked globally by their weights in the graph.

Although SEAL works well given three or four seeds, SEAL has a limitation on the number of seeds it can handle. Figure 2 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. The problem is that 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.

Bootstrapping is an (unsupervised) iterative process in which a system continuously consumes its own outputs to improve its own performance [1, 3, 4]. The advantage of bootstrapping is that, if successful, the performance of a system can be greatly improved with minimal supervision, so it is natural to explore bootstrapping using iSEAL. 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 from iteration to iteration.

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 [8], which is similar to that used in the previously published version of SEAL; PageRank [5] 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.

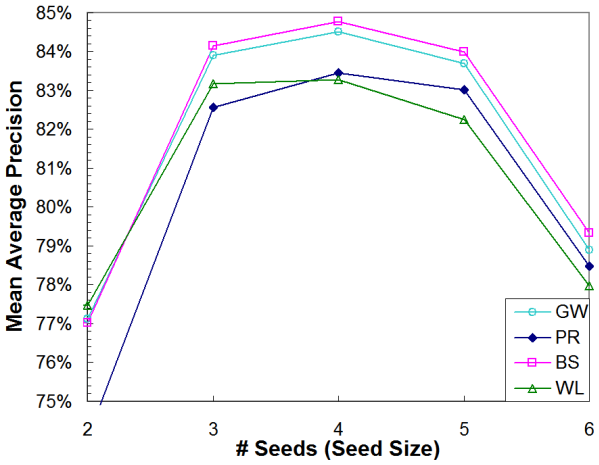


Figure 2. MAP of set expansion using various rankers and various number of seeds.

Below, Section 2 presents Iterative SEAL. Section 3 presents the ranking methods for iterative set expansion. Section 4 describes the details of the experimental design, and Section 5 presents the experimental results. The paper concludes in Section 6 with a discussion of planned future work.

2. Iterative SEAL

In this section, we present the Iterative SEAL (iSEAL) system to examine two iterative processes: supervised expansion and bootstrapping. In our experiments, both processes start their first iteration with two supervised seeds, which is the least 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 cumulating statistics from one iteration to another; thus no information is ever tossed

away. This allows the expansion of seeds in current iteration to have access to all statistics computed in the past iterations. We present two seeding strategies for this process below.

2.1.1 Fixed Seed Size

Fixed Seed Size (FSS) is a strategy that requires two seeds in every iteration. In supervised expansion, the two seeds are randomly selected from a collection of seeds provided by some reliable source (i.e., users). The pseudo-code for this strategy is presented below:

$stats \leftarrow \emptyset$
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 cumulated statistics, $rank_r(stats)$ applies the ranker r on the cumulated $stats$ to produce a $ranked\_list$ of entities.

2.1.2 Increasing Seed Size

Increasing Seed Size (ISS) is a strategy that starts the iterative process with two supervised seeds, then it 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. In an ideal case, n should be the number that maximizes the set expansion performance. By examining the results of the preliminary study conducted on the development set shown in Figure 2, we observed that four seeds maximizes the performance; thus we set n to four. This number is also suggested by Etzioni et al. [1] and Nadeau et al. [4]. As a result, the pseudo-code for this strategy is as follows:

$stats \leftarrow \emptyset, used \leftarrow \emptyset$
for i = 1 to M **do**
  if $i = 1$ **then**
    $seeds \leftarrow select_2(E)$
  else
    $m = min(3, |used|)$
    $seeds \leftarrow select_m(used) \cup select_1(E)$
  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. All other notations are the same as those defined in last section. This strategy starts by expanding two supervised seeds. For the second iteration, it expands the two *used* seeds plus an additional *new* supervised seed (three seeds in total). For every successive iteration, it expands three randomly selected used seeds plus a new supervised seed (four seeds in total). Therefore, every iteration consumes exactly only one new supervised seed, and every iteration after the second one consumes four seeds (three used plus one new).

2.2 Bootstrapping

Bootstrapping refers to iterative unsupervised set expansion. Unlike supervised expansion, bootstrapping improves SEAL's performance with minimal supervision. It is an iterative process for a system to continuously improve its own performance by utilizing its own outputs. This technique is very sensitive to the system's performance because errors can easily propagate from one iteration to another. For example, if the top ranked entities in the last iteration are incorrect, then when they are chosen as seeds, the expansion results of the next iteration could be worse. A carefully designed seeding strategy can minimize the propagated errors. We present the two seeding strategies in the unsupervised mode.

2.2.1 Fixed Seed Size

As mentioned earlier, FSS is a strategy that requires two seeds in every iteration. Unlike supervised expansion where the two seeds of every iteration are supervised, bootstrapping uses unsupervised seeds by selecting 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. In unsupervised mode, regardless of the number of iterations, the supervised seeds required are only those two initial seeds.

2.2.2 Increasing Seed Size

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 is presented below with the difference between this strategy in unsupervised and supervised mode underlined:

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

where $top\_one(ranked\_list)$ returns a new entity that has the highest weight in $ranked\_list$. Again, the total number of supervised seeds required are only the two initial seeds.

3. Ranking Methods

In the last section, we presented two iterative processes where each has two seeding strategies. In all four methods presented, there is a $rank_r(stats)$ function that ranks entities based on the cumulated $stats$ using ranker r. In this section, we present those rankers used in our experiments: Random Walk with Restart, PageRank, Bayesian Sets, and Wrapper Length.

3.1. Random Walk with Restart

Wang and Cohen [10] 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 3 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. Table 1 shows all possible source node types and their relations with some target node types.

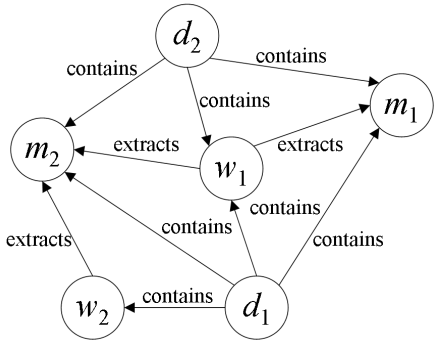


Figure 3. Example graph constructed by Random Walk. Every edge from node x to y has an inverse relation edge from node y to x that is not shown here (i.e. m_i is extracted by w_i).

Source Type	Edge Relation	Target Type
document	contains	wrapper
document	contains	mention
wrapper	extracts	mention
wrapper	is_contained_by	document
mention	is_contained_by	document
mention	is_extracted_by	wrapper

Table 1. All possible source node types and their relations with some target node types.

In order to explain the random walk equations in detail, 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 [8]. 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,

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

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,

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

In order to perform the 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,

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

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:

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

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 for evaluation. Throughout our experiment, we use a constant λ of 0.01 which shows good performance in our development set. New statistics can be cumulated easily by adding additional nodes and edges to the existing graph.

3.2. PageRank

Page et al. [5] 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), many research have shown that it can also be used to rank other things [9, 11]. PageRank measures relative importance of each node within a graph; it interprets a link from node A to node B as a vote, by node A, for node B. However, the weights of the votes are not the same. Votes cast by nodes that are themselves *important* weigh more heavily and help to make other nodes *important*.

The graph that we use for PageRank is identical to the one shown in Figure 3, except that edges are undirected and they do not have relations. In order to compute PageRank scores, we will also build a transition matrix M where each entry at (x, y) is the probability of traveling one step from a source node x to a target node y, or more precisely,

$$M_{xy} = \frac{1}{|y : x \longrightarrow y|} \quad (5)$$

We also define a state vector $\vec{v_t}$, and the state vector at $t+1$ iteration is defined as:

$$\vec{v}_{t+1} = \lambda\vec{u} + (1-\lambda)M\vec{v}_t \quad (6)$$

where $\vec{u}$ is a teleport vector with probabilities uniformly distributed over all nodes in the graph. In PageRank, each node has an uniform probability λ of teleporting to an arbitrary node in the graph. Page et al. [5] uses a λ of 0.15, which is also what we use in the experiments. We iterate the graph until the state vector converges, and rank the extracted mentions based on their final node weights. New statistics can be easily cumulated by attaching new nodes and edges to the existing graph.

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. They evaluated their algorithm on three datasets: EachMovie, NIPS dataset, and Grolier encyclopedia, and have achieved good results.

We implemented Bayesian Sets, by constructing one large two-dimensional feature table where each column represents a particular feature, each row represents a particular extracted item, and each entry (j, k) in the table indicates whether or not item x_k possesses the feature f_j. We incorporate two important 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. For an extracted item x_k, we calculate its score using the equation:

$$\log score(x_k) = c + \sum_{j:(j,k)=1} \Big[\log(\alpha_j + n_j) - \log(km_j) - \log(\beta_j + N - n_j) + \log(k(1-m_j)) \Big] \quad (7)$$

where c is some constant, m_j is the mean value of column j, n_j is the number of seeds contained by d_j or extracted by w_j, N is the total number of seeds, and k is set to two as also done by Ghahramani and Heller [2]. The basic concept is that an item will have a higher score if it is a) extracted by many wrappers that extract few items and b) extracted from many documents that contain few extracted items. New statistics can be cumulated with the existing ones simply by appending new rows and columns to the feature table.

3.4. Wrapper Length

SEAL defines a wrapper, which extracts item x, as maximally-long contextual strings that x have in common with the input seeds in a document. We have observed that the longer the common left and right contextual strings, the more indicative that x is correlated with the seeds in that document. Therefore, we propose a simple, fast but ad hoc ranking algorithm called Wrapper Length as detailed below:

$$score(x) = \prod_{j \text{ extracts } x} length(w_j) \tag{8}$$

or equivalently, in log space:

$$\log score(x) = \sum_{j \text{ extracts } x} \log(length(w_j)) \tag{9}$$

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. In this approach, new statistics can be cumulated simply by summing up the log of wrapper length for every new item, and the memory size needed to store these statistics is only proportional to the number of items.

4. Experimental Setting

4.1. Datasets

We evaluate the iterative processes using the datasets[3] presented in Wang and Cohen [10], which consists of 36 manually constructed lists across three different languages: English, Chinese, and Japanese (12 lists per language). Each list represents a class for a certain language (e.g., English country names). Each class contains all of its entities (e.g., our country) and each entity includes all of its mentions (e.g., USA or America). There are a total of 2515 entities and 4312 mentions in this dataset, with an average of 70 entities per class and 1.7 mentions per entity.

4.2. Evaluation Metric

Since the output of iSEAL is a ranked list of extracted mentions, 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:

$$AvgPrec(L) = \frac{\sum_{r=1}^{|L|} \text{Prec}(r) \times \text{FirstSeenEntity}(r)}{\text{Total \# of Correct Entities}} \tag{10}$$

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.

4.3. Evaluation Procedure

As a baseline, we introduce in the experimental results a fifth ranker called Wrapper Frequency (WF), which is simply the number of wrappers that extract a particular entity. Each experiment evaluates a particular combination of the following: iterative process, seeding strategy, and ranker; we evaluated all possible combinations (20 combinations in total).

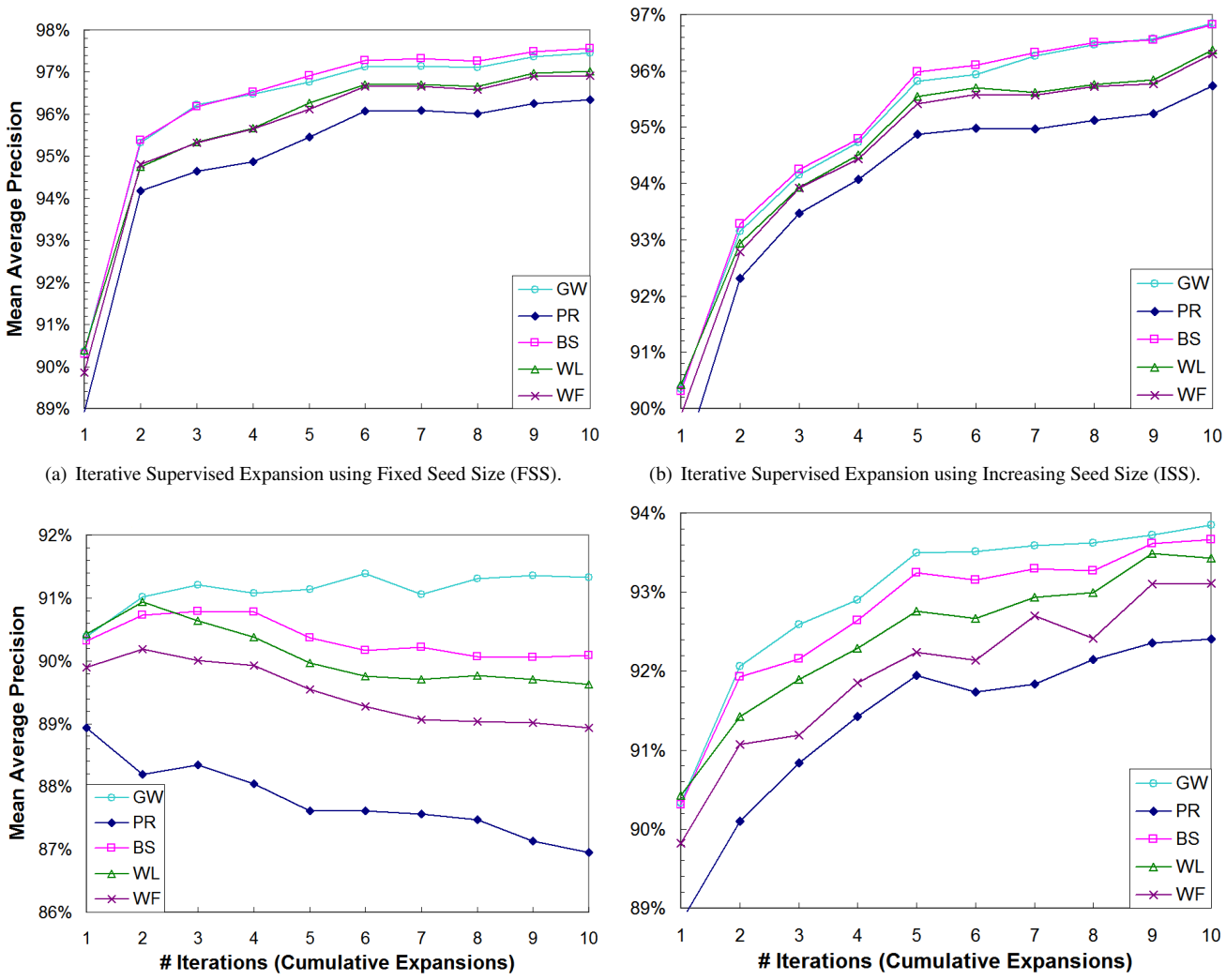
For each combination, we performed ten iterative expansions on each of the 36 evaluation datasets independently three times; thus, at each of the ten iterations, there are 108 (3×36) ranked lists. We then report and plot the MAP of those ranked lists in the graphs shown in the next section.

5. Experimental Results

We first examine the effect of supervised expansion. Figure 4(a) and 4(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 (GW) 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 4(c) and 4(d) illustrate the performance of various rankers in bootstrap mode using FSS and ISS respectively. As illustrated, FSS failed to improve any ranker other than Random Walk at the 10^{th} iteration, and even this improvement

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



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

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

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

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

Figure 4. MAP of using various iterative methods and rankers: Random Walk (GW), PageRank (PR), Bayesian Sets (BS), Wrapper Length (WL), and Wrapper Frequency (WF).

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 used seeds to support the newly chosen one, which minimizes the chance of expanding seeds that are all incorrect.

Figure 4(c) shows that Random Walk is the most robust ranker among the five rankers, followed by Bayesian Sets and Wrapper Length. While all rankers performed poorly due to noisy seeds, only Random Walk improved slightly at the 10^{th} iteration. This shows that Random Walk is robust to noisy seeds. Figure 4(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, again, Bayesian Sets and Wrapper Length.

Table 2 summarizes the experimental results by averaging the performance of all rankers for each iterative process and seeding strategy. Since supervised seeds are difficult to obtain, we analyze the amount of relative improvement (in MAP) per supervised seed for each method. The table shows that supervised expansion using FSS gives the most improvement (7.9%) and produces the highest score (97.1%); however, it consumes 20 supervised seeds, re-

	S. Expansion		Bootstrapping	
	FSS	ISS	FSS	ISS
Avg. MAP @ 1^{st}	89.9%	89.9%	89.9%	89.9%
Avg. MAP @ 10^{th}	97.1%	96.4%	89.4%	93.3%
Relative Δ MAP	7.9%	7.2%	-0.6%	3.7%
Total # S. Seeds	20	11	2	2
Δ MAP / S. Seed	0.4%	0.7%	-0.3%	1.9%

Table 2. MAP averaged over all rankers for each iterative method.

	MAP @ 1^{st}	MAP @ 10^{th}	Δ MAP
GW	90.3%	93.9%	3.9%
PR	88.8%	92.4%	4.0%
BS	90.3%	93.7%	3.7%
WL	90.4%	93.4%	3.3%
WF	89.8%	93.1%	3.7%
Average	89.9%	93.3%	3.7%

Table 3. MAP of relative improvements (1^{st} to 10^{th} iteration) using bootstrapping with ISS.

sulting in a low improvement-per-seed ratio (0.4%). The one that gives the highest ratio is bootstrapping using ISS (1.9%), which shows that it is an effective bootstrapping method. Looking more closely at this particular method, Table 3 shows that each ranker's performance using this method gives an relative improvement of about 3.7% in MAP. This shows that this method is effective for every ranker, even for the ranker PageRank which has the worst performance in every experiment. The table also indicates that Random Walk is able to reach a MAP of nearly 94% at 10^{th} iteration using only two supervised seeds. This performance is even better than the MAP of 93% achieved by the previously published version of SEAL using *three* supervised seeds and the same amount of web pages (i.e., 100).

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. Unlike Random Walk, PageRank, and Bayesian Sets, the amount of memory it requires to store cumulated statistics is only proportional to the number of extracted entities.

The results suggest that if many supervised seeds are available, then supervised expansion using FSS should be considered. If only a few is 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.

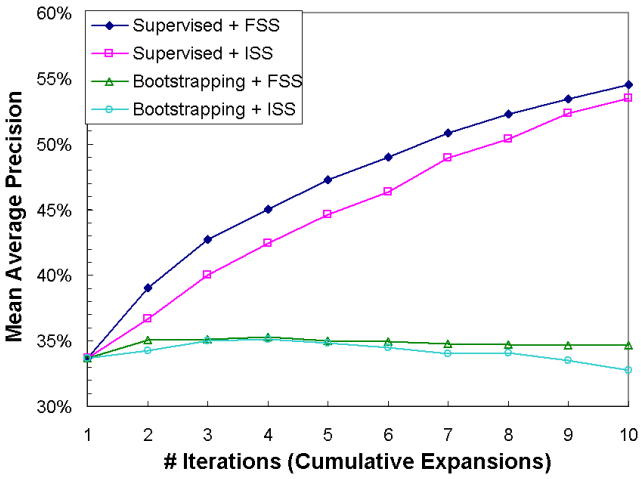


Figure 5. Iterative expansion using G. Sets.

Since Google Sets is publicly available online, it is natural for us to examine the performance of Google Sets using the iterative set expansion methods presented in this paper. Google Sets returns a list of items without their confidence scores; thus, we assume each item has a weight of $1 - \frac{r-1}{|L|}$, where r is the rank and L is the ranked list. This gives the first item a weight of one and the last item a weight close to zero. When we cumulate statistics, we simply sum up the weights for every item. Figure 5 shows the performance of iterative set expansion using Google Sets. For the supervised expansion, Google Sets shows a monotonic increase of performance for both FSS and ISS; however, it shows no improvement for both bootstrapping methods. There could be two reasons. First, our assumption of their item weights is ineffective or incorrect. Second, as mentioned earlier, bootstrapping is sensitive to a system's performance. Since Google Sets has a base performance (the performance at the first iteration) of only about 34% in MAP (SEAL has 90%), its top ranked items are often incorrect, causing errors to propagate easily and worsening the results.

6. Conclusion and Future Work

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 is even better than that of using three supervised seeds (with same amount of web pages) as published in previous results of SEAL. 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 fast, simple, and light-weight ranker called Wrapper Length that shows good performance in both supervised and unsupervised modes. If memory is an issue, then Wrapper Length is the best ranker to use. From the experimental results, we have learned that if there are many supervised seeds, then the best way to improve performance is by supervised expansion using FSS. However, if there are only a few supervised seeds, then the best way is by bootstrapping using ISS.

In the future, we would like to investigate whether or not bootstrapping with ISS will continue to improve the performance of Random Walk monotonically after ten iterations. In addition, we would like to examine in detail the reason why Random Walk performs well while PageRank performs poorly compared to other rankers. Lastly, due to the results of Google Sets, we would like to analyze the minimum base performance required for a set expansion system to bootstrap successfully.

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] Z. Kozareva. Bootstrapping named entity recognition with automatically generated gazetteer lists. In *EACL*. The Association for Computer Linguistics, 2006.

[4] D. Nadeau, P. D. Turney, and S. Matwin. Unsupervised named-entity recognition: Generating gazetteers and resolving ambiguity. In L. Lamontagne and M. Marchand, editors, *Canadian Conference on AI*, volume 4013 of *Lecture Notes in Computer Science*, pages 266–277. Springer, 2006.

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

[6] 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.

[7] B. Settles. Biomedical named entity recognition using conditional random fields and rich feature sets. In N. Collier, P. Ruch, and A. Nazarenko, editors, *COLING 2004 International Joint workshop on NLPBA/BioNLP*, pages 107–110, Geneva, Switzerland, August 28th and 29th 2004. COLING.

[8] 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.

[9] J. Wang, J. Liu, and C. Wang. Keyword extraction based on pagerank. In Z.-H. Zhou, H. Li, and Q. Yang, editors, *PAKDD*, volume 4426 of *Lecture Notes in Computer Science*, pages 857–864. Springer, 2007.

[10] 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.

[11] C. C. Yang and K. Y. Chan. Retrieving multimedia web objects based on pagerank algorithm. In A. Ellis and T. Hagino, editors, *WWW (Special interest tracks and posters)*, pages 906–907. ACM, 2005.