SEISA: Set Expansion by Iterative Similarity Aggregation

Yeye He*
University of Wisconsin-Madison
Madison, WI 53706
heyeye@cs.wisc.edu

Dong Xin†
Microsoft Research
Redmond, WA 98052
dongxin@microsoft.com

ABSTRACT

In this paper, we study the problem of expanding a set of given seed entities into a more complete set by discovering other entities that also belong to the same concept set. A typical example is to use "Canon" and "Nikon" as seed entities, and derive other entities (e.g., "Olympus") in the same concept set of camera brands. In order to discover such relevant entities, we exploit several web data sources, including lists extracted from web pages and user queries from a web search engine. While these web data are highly diverse with rich information that usually cover a wide range of the domains of interest, they tend to be very noisy. We observe that previously proposed random walk based approaches do not perform very well on these noisy data sources. Accordingly, we propose a new general framework based on iterative similarity aggregation, and present detailed experimental results to show that, when using general-purpose web data for set expansion, our approach outperforms previous techniques in terms of both precision and recall.

Categories and Subject Descriptors: H.2.8 Database Application: Data Mining

General Terms: Algorithms.

Keywords: Set Expansion, Named Entity Recognition, Similarity Measure.

1. INTRODUCTION

Set expansion refers to the practical problem of expanding a small set of "seed" entities, into a more complete set by discovering other entities that also belong to the same "concept set". Here a "concept set" can be any collection of entities that conceptually form a set that people have in mind, and "seeds" are the instances of entities in the set. As an example, a person wanting to discover all camera brand names may give a small number of well-known brand names like "Canon" and "Nikon" as seeds, the set expansion techniques would leverage the given data sources to discover other camera brands, such as "Leica", "Pentax" and "Olympus" that are also camera brands.

Set expansion systems are of practical importance and can be used in various applications. For instance, web search engines may use the set expansion tools to create a comprehensive entity repository (for, say, brand names of each product category), in order to deliver better results to entity-oriented queries. As another example, the task of named entity recognition can also leverage the results generated by set expansion tools [13].

Considerable progresses have been made in developing high-quality set expansion systems. The most relevant efforts include Google Sets [1], which employs proprietary algorithms to do set expansions. However, due to its proprietary nature, algorithms and data sources behind Google Sets are not publicly available for future research endeavors.

Another prominent line of work is the SEAL system [16, 17, 18, 19]. They adopted a two-phase strategy, where they first build customized text wrappers based on the input seeds in order to extract candidate entities from web pages in a precise manner. They then use a graph-based random walk to rank candidates entities based on their closeness to the seeds on the graph. While they have demonstrated that this customized data extraction/ranking process can produce results with high quality, the necessary online data extraction can be costly and time-consuming.

Here we study the problem of conducting set expansion using general web data sources without resorting to online data extractions specific to the given seeds. In particular, we look at two common types of web data: the HTML lists extracted from web pages by web crawls (henceforth referred to as the Web Lists) and the web search query logs (the Query Logs). We model both types of data using bipartite graphs to provide a unified computation model.

Such general-purpose web data can be highly useful for set expansion tasks: they are very diverse in nature, with rich information that covers most domains of interest. In addition, since these general data are not domain/seed specific, they can be pre-processed and optimized for efficiency purposes.

However, these general web data can be inherently noisy. Random walk or other similarity measures along may not be sufficient to distinguish true results from the noises, especially when the number of seeds are limited. As we observe in our experimental evaluations, that random walk based ranking techniques used in previous work perform poorly on the general-purpose Web Lists or Query Logs and produce results with low precision/recall. Partly because of that, previous approaches [16, 17, 18, 19] use seed-specific and page-specific wrappers to reduce the candidate set to a smaller and much cleaner subset over which the random walk based ranking techniques work reasonably well. However, we note that this additional data extraction process is at the cost of overall architectural complexity and system responsiveness.

Unlike previous approaches, we in this work propose a general framework that only uses general-purpose web data without resorting to on-line data extractions specific to the given seeds. In particular, while previous random walk based approaches leverage the intuition that candidates close to the given seeds in the graph struc-

*Work done at Microsoft Research

†Work done at Microsoft Research. Now with Google (dongxin@google.com).

WWW 2011, March 28–April 1, 2011, Hyderabad, India.
ACM 978-1-4503-0632-4/11/03.

ture are more likely to belong to the same concept set as the seeds; we take an alternative tack and propose to measure the quality of an expanded set of entities relative to the given set of seeds in a more straightforward and comprehensible way. Intuitively, a set of expanded results is "good" if it has two key properties: (1) the set of produced entities are similar to the given seeds; (2) the set of produced entities are coherent in the sense that they represent a consistent concept. We abstract the intuitions and define quality of the result set as the sum of two component scores: the *Relevance* of a set of entities that measures their similarity with the given seeds, and *Coherence* of the set of entities produced which is how closely the entities in this set are related to each other. Based on this quality measure, we further develop a class of iterative set expansion algorithms for which we call SEISA (Set Expansion by Iterative Similarity Aggregation). We show that SEISA is robust and effective in producing expanded sets over noisy web data sources.

The rest of the paper is organized as follows. We summarize the related work in the space of set expansion in Section 2. We then formally define the problem of set expansion using the new quality metric we propose in Section 3. In Section 4 we detail our iterative set expansion algorithms. Finally we present our experimental results in Section 5 and conclude this paper in Section 6.

2. RELATED WORK

There is a significant body of related work in the broad space of information extraction and named entity extraction. We will only summarize work most relevant to set expansion due the limit of space.

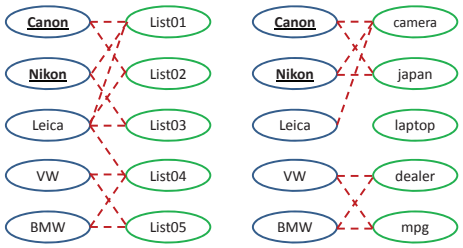
Wang and Cohen [16, 17, 18, 19] developed the SEAL system for set expansion, using a two-phase extraction/ranking architecture. In the first extraction phase, they build for each web page customized wrappers using maximal left/right context that would enclose all given seeds, which are in turn applied on the web page from which they are constructed to extract candidate terms in addition to the given seeds. In the second ranking phase, web pages, wrappers and candidate terms are modeled as nodes in the graph, and random walk techniques are used to rank candidates based on their structural proximity to the seeds in the graph. In comparison, our approach ranks a set of candidates as a whole based on its relevance and coherence, and does not require page-specific and seed-specific data extraction process.

Agichtein et al. [4] introduce the Snowball system that bootstraps from a small number of input tuples, by first obtaining typical contextual patterns of the seed tuples from the web pages, which are used in turn to extract more tuples. While Snowball is well suited for extracting certain types of structured data like binary relations, it may not work well for set expansion due to its reliance on textual context patterns (sets can be viewed as unary relations of tuples, whose context can be much more dynamic and less predictable than that of binary relations).

Etzioni et al. [6, 7] develop the KnowItAll system that automatically extracts facts from the web using textual patterns like "cities such as Paris, London and New York" to extract candidate entities. Candidates are then ranked in a bootstrapping manner using statistical information gathered from the search engine such as PMI over hit counts.

Talukdar et al. [14] study the problem of set expansion from open text. They propose to automatically identify trigger-words which indicate patterns in a bootstrapping manner.

Ghahramani et al. [8] uses Bayesian inference to solve the problem of set expansion. It has been shown in [18] that this candidate ranking mechanism is comparable to random walk in quality of the results of the expanded set.



(a) Web List Data (b) Query Log Data

Figure 1: Bipartite graph data model

Set expansion is also somewhat related to the problem of class label acquisition [15, 20] where the goal is to propage a set of class labels to data instances using labeled traning examples. While the set expansion problem can be modeled as propagating class labels associated with seeds to candidate entities, we observe that a large number of training examples is necessary in order for these techniques to be effective.

Finally Google Sets [1] does set expansions using propriety algorithms which are not publicly available.

3. PROBLEM DEFINITION

3.1 Data Model

In this work we target general web data sources. Specifically, we look at lists extracted from the HTML web pages (the Web List data), and the web search query logs (the Query Log data). In each case, we model the data as bipartite graphs as in Figure 1, with candidate terms being nodes on one side (henceforth referred to as term-nodes) and their contexts on the other side. Since we use textual terms in the web data as candidate entities for the expanded set, from this point on we will be using the word "term" interchangeably with "entity".

Web List Data. For Web List as in Figure 1a, each unique web list crawled from the web, ("`List01`", "`List02`", etc), is modeled as a node on the right-hand-side, while each term that appears in those web lists is modeled as a term-node on the left hand side. In this example, the underlined nodes "`Canon`" and "`Nikon`" on the left are the seed terms, while the remaining terms, including "`Leica`", "`VW`" and "`BMW`", are possible candidate terms. There is an edge connecting a term-node with a list-node if that term is a members of the list. For example, the list-node `List01` connects to "`Canon`", "`Nikon`" and "`Leica`", indicating that all three terms are members of `List01`, which is probably a web list on some web page that enumerates a list of camera brands.

While it is possible to resort to additional information of the web data to assign different weights to each edge (using the quality of the page from which the list is extracted, for example), we in this work adopt a simpler approach that only assigns a uniform weight of 1 to each edge in the Web List graph. We in our experiments find this approach to work quite well.

Query Log Data. Query log data is modeled as in Figure 1b. Here for each keyword query, we break up the query into two parts, the term and the context. The context is a prefix or suffix of the query up to 2 tokens, and the term is the remainder of the query. Each term is again modeled as a graph node on the left, the context is modeled as a node on the right.

There are various ways in which we can model edges in the graph for query log data. In this work we assign weight of the edge be-

tween each pair of nodes using the *Mutual Information* between the query term and query context, which is defined in Definition 1.

DEFINITION 1. *Let $Prob(t)$ be the probability that term t occur in the query log, $Prob(c)$ be the probability that context c occur in the query log, let $Prob(t, c)$ be the probability that the term t and context c co-occur in the query log. The* ***Mutual Information*** *H(t, c) is defined as: $H(t, c) = \frac{Prob(t,c)}{Prob(t)*Prob(c)}$*

Furthermore, we only keep the edge between a pair of nodes if the Mutual Information between the term and the context (or the weight on the edge) is positive, and additionally, the co-occurrences of the term and the context is frequent enough to be above certain threshold. Figure 1b is an example of the resulting bipartite graph after this simple processing.

In general, we feel that this bipartite graph model is straightforward and general enough to be applied and extended to other types of data sources.

3.2 Similarity Metric

With this bipartite data model, intuitively the overall task of doing set expansion given a set of seeds can to an extent be viewed as the problem of finding term-nodes that are similar to the given seed-nodes, using the right hand side nodes as the features. In order to measure similarities between the term-nodes, common similarity metrics, like Jaccard Similarity [11] and Cosine Similarity [11] as defined below, can all be used.

DEFINITION 2. *[11] Let x, y be two term-nodes on the left hand side. Let L_x and L_y be the two sets of right side nodes that connect to node x and y, respectively. The* ***Jaccard Similarity*** *of x and y, denoted as $Sim_{Jac}(x, y)$, is defined as $Sim_{Jac}(x, y) = \frac{|L_x \cap L_y|}{|L_x \cup L_y|}$.*

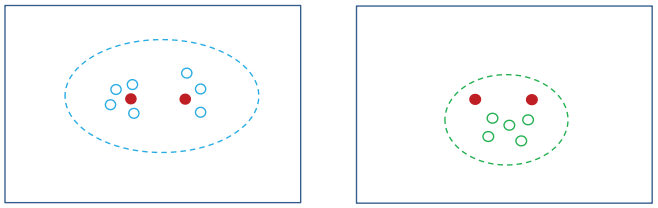
DEFINITION 3. *[11] Let x, y be two term-nodes on the left hand side. Let V_x and V_y be the weight vectors that indicate the weights of the edges that connect web lists to node x and y, respectively. The* ***Cosine Similarity*** *of x and y, denoted as $Sim_{Cos}(x, y)$, is defined as $Sim_{Cos}(x, y) = \frac{V_x \cdot V_y}{\|V_x\| \|V_y\|}$.*

We use the following two examples as simple illustrations of the Jaccard similarity and Cosine similarity.

EXAMPLE 1. *We first illustrate the computation of Jaccard similarity of two term-nodes in our bipartite graph model. In Figure 1a, the term-node "Canon" connects to list nodes $L_{\text{"Canon"}}$ = {"List01", "List02"}; while the term-node "Leica" connects to nodes $L_{\text{"Leica"}} = \{$"List01", "List02", "List03", "List04"$\}$. By Definition 2, the Jaccard similarity between the seed-nodes "Leica" and "Canon" is $\frac{|L_{\text{"Canon"}} \cap L_{\text{"Leica"}}|}{|L_{\text{"Canon"}} \cup L_{\text{"Leica"}}|} = \frac{2}{4} = 0.5$. Similarly, the similarity between "Leica" and the other seed-node "Nikon" is also $\frac{2}{4} = 0.5$.*

On the other hand, the Jaccard similarities between "VW" and both of the seed-nodes "Canon" and "Nikon" are $\frac{0}{6} = 0$. Therefore, using the Jaccard Similarity definition, the term "Leica" is more similar to both seeds than the term "VM".

Next we show how Cosine similarity between term-nodes is computed. In Figure 1a, the term-node "Canon" connects to $L_{\text{"Canon"}}$ ={"List01", "List02"}, its edge weight vector $V_{\text{"Canon"}}$ is thus $(1, 1, 0, 0, 0)$. By the same token, the edge weight vector for "Leica" is $V_{\text{"Leica"}} = (1, 1, 1, 1, 0)$. According to Definition 2, the Cosine similarity between nodes "Leica" and "Canon" is $\frac{V_{\text{"Canon"}} \cdot V_{\text{"Leica"}}}{\|V_{\text{"Canon"}}\| \|V_{\text{"Leica"}}\|} = \frac{2}{2.83} = 0.71$. Similarly, the similarity between "Leica" and the other seed-node "Nikon" is also $\frac{2}{2.83} = 0.71$.



(a) A set with high relevance (b) A set with high coherence

Figure 2: Quality of an expanded set

The Cosine similarities between "VW" and both of the seed-nodes "Canon" and "Nikon" are 0 due to the lack of overlap in the right side list-nodes. We again have the term "Leica" to be more similar to seeds than the term "VM".

While in this section we only discuss two most commonly used similarity metrics, the Jaccard Similarity and the Cosine Similarity, we emphasize that the set expansion framework to be introduce in detail in Section 4 is general and extensible enough that any other similarity metrics can be easily plugged in to be used. Furthermore, in our experimental evaluations, we find that the performances of set expansion using both similarity metrics are reasonably good, underlining the generality of the framework we propose.

3.3 Quality Measurement

While the previous work uses techniques like random walk to rank individual terms based on their graph structure similarity to the given seeds, we see the expanded set of entities as a whole and propose a simple and intuitive metric to measure the quality of the expanded set, as will be detailed in this section.

The first observation we have is that, the more similar the expanded entities are to the given seed entities, the better quality the expanded set. This is intuitive because after all the task of set expansion is to find entities that are in the same "concept set" as the seeds, which by definition should be somewhat similar to the seeds. We formalize this observation with the following definition of *relevance* to capture the similarity between the expanded set and the seed set.

DEFINITION 4. *Let U be the universe of entities, $R \subseteq U$ be the expanded set, and $S \subseteq U$ be the seed set. Let $Sim : U \times U \rightarrow [0, 1]$ be the function that measures the similarity of any two entities. The* ***relevance*** *of R with respect to S is defined as:*

$$S_{rel}(R, S) = \frac{1}{|R| * |S|} * \sum_{r \in R} \sum_{s \in S} Sim(s, r)$$

To better illustrate this definition of *relevance*, we use Figure 2 to graphically demonstrate the quality of the expanded set. In both Figure 2a and Figure 2b, the two solid dots in the middle represent the given seed set S, while the circles surrounding these two dots are the derived entities that constitute the expanded set R. The similarity of any two entities is then represented as the distance of these two dots in the graph.

It is clear that in both of these two figures, the expanded set of entities as circled by the dashed oval are very similar (or graphically speaking, close) to the two given seeds. So in terms of our *relevance* metric, both of the two sets in Figure 2a and Figure 2b have high *relevance* to the given seeds.

However, we observe that this definition of *relevance* alone does not fully capture the quality of the expanded set. The reason for

this is that while the overarching goal of set expansion is to find a consistent "concept set" that are very similar to the given seeds, there could be cases where a set of entities are similar to the seeds but not consistent enough to be a *coherent* concept set. As an example, in Figure 2a, while the expanded entities as denoted by the circles are close to the given seeds, they are relative dispersed in the space and may not form a consistent "concept set" as required by set expansion. On the other hand, the the expanded entities in Figure 2b are not only equally close to the given seeds as in Figure 2a, they are also much closer to each other to form a consistent "concept set". Thus, the expanded entities in Figure 2b may be a better candidate for the expanded set than the entities in Figure 2a.

To capture the intuition in Figure 2b that the closer the entities in the expanded set are to each other, the more coherent and thus better the set as a whole is, we formally define the notion of *coherence* in the following.

DEFINITION 5. *Let U be the universe of entities, $R \subseteq U$ be the expanded set, $Sim : U \times U \rightarrow [0, 1]$ be the function that measures the similarity of any two entities. The* ***coherence*** *of R is defined as:*

$$S_{coh}(R) = \frac{1}{|R| \cdot |R|} * \sum_{i=1}^{|R|} \sum_{j>i}^{|R|} Sim(r_i, r_j)$$

where $r_i, r_j \in R$.

Based on the observation that both *relevance* and *coherence* contribute to the quality of an expanded set, we define the quality of the expanded set as the weighted sum of *relevance* and *coherence* as follows.

DEFINITION 6. *Let U be the universe of entities, Let $R \subseteq U$ be the expanded set, $S \subseteq U$ be the seed set. Let $0 \leq \alpha \leq 1$ be the constant weight factor. The* ***quality*** *of the expanded set R with respect to the seed set S, $Q(R, S)$, is defined as:*

$$Q(R, S) = \alpha * S_{rel}(R, S) + (1 - \alpha) * S_{coh}(R)$$

Here, α is a constant weight that balances the emphasis between *relevance* and *coherence*. In our experiments that we will detail in Section 5, we find $\alpha = 0.5$ to be a good value in practice, and this intuitive yet simple definition of quality works well in the extensive experiments that are conducted.

3.4 Problem Statement

With the definition of quality metric, we formally state the problem as follows. Given the universe of candidate terms U and some seeds $S \subseteq U$; given a similarity function $Sim : U \times U \rightarrow [0, 1]$ that measures the similarity of any two terms, identify the ***expanded seed set*** R, $R \subseteq U$ and is of size K, such that the objective function $Q(R, S)$ is maximized, where

$$Q(R, S) = \alpha * S_{rel}(R, S) + (1 - \alpha) * S_{coh}(R)$$

Intuitively, the *expanded seed set*, or the ESS, is the core component of the concept set that we want to expand, and consists of entities that we know with high confidence that belong to the desired concept set. We say an ESS is good if its quality score is high. Once a good ESS (denoted as R) is derived, individual terms t can then be ranked based on R and the seed set S using the ranking function $g(t, R, S)$, which is again a straightforward combination of *relevance* score and *coherence* score as follows.

$$g(t, R, S) = \frac{\alpha}{|S|} \sum_{i=1}^{|S|} Sim(t, s_i) + \frac{(1 - \alpha)}{|R|} \sum_{i=1}^{|R|} Sim(t, r_i) \quad (1)$$

where $r_i \in R$ and $s_i \in S$.

However, we show that the problem of finding the optimal R of size K with maximum quality score is NP-hard.

THEOREM 1. *Given the seed set S, the problem of finding an entity set R of size K that maximizes the objective function $Q(R, S)$ is NP-Hard.*

The hardness of this problem can be proved by reduction from the maximum clique problem [9]. Details of the proof can be found in Appendix 8.1.

4. ALGORITHMS FOR SET EXPANSION

Given Theorem 1 which states that it is NP-Hard to find the optimal *expanded seed set* (ESS), we in this section propose two greedy algorithms, the *static thresholding algorithm* and the *dynamic thresholding algorithm*, that iteratively refine a candidate ESS R of size K to maximize $Q(R, S)$ (Definition 6). Both algorithms are built on top of an automatic score thresholding technique. In this section, we first outline two algorithms, then describe the automatic score thresholding method, and finally, we discuss the connection of our proposed algorithms and the standard random walk based approaches.

4.1 Iterative Similarity Aggregation

On the high level, the *static thresholding algorithm* fixes the size of ESS R at the beginning, and then iteratively searches for terms in R to maximize $Q(R, S)$; while the *dynamic thresholding algorithm* refines both the size of R and contents of R at the same time in each iteration.

4.1.1 Static Thresholding Algorithm

The static thresholding algorithm starts with a good guess of ESS, then iteratively improve the quality metric as defined in Definition 6 by replacing one entity in the ESS of the previous iteration, until the computation of ESS converges and a local maximum of the quality score is reached. The pseudo-code of the algorithm is described in Algorithm 1.

The *static thresholding algorithm* takes two parameters, the set of seed entities, $seeds$, and the $graph$ with all candidate $terms$ as left side nodes. We start by computing the relevance score of each term with the seeds $S_{rel}(term_i, seeds)$, as defined in Definition 4, in the first for loop. We then rank the terms according to their relevance scores, and pick top K ranked terms as the initial estimate of the ESS, R_0, where the threshold value K is determined by a thresholding analysis of the score distribution that will be detailed in Section 4.2.

In the subsequent iterations in the while loop, we iteratively compute the new candidate ESS R_{iter} based on R_{iter-1} of the previous iteration and progressively improve the overall quality score of the ESS until a local maximum is reached. Specifically, in each iteration $iter$, we compute the relevance score of each candidate term $term_i$ with the previous ESS R_{iter-1}, $S_{rel}(term_i, R_{iter-1})$, and the corresponding ranking function $g(term_i)$ which is a weighted combination of the relevance score with the $seeds$, and the relevance score with R_{iter-1}. We then sort the candidate terms by $g(term_i)$. Let the top ranked K terms be R'_{iter}, if $R'_{iter} \neq R_{iter-1}$, we replace the lowest ranked term in R_{iter-1} with the top ranked term $r \in R'_{iter}$ that is not in R_{iter-1}, and continue the iteration; otherwise we have converged and will stop and return R_{iter-1} as the result of ESS.

We use the following running example to demonstrate how the static thresholding algorithm works to compute ESS, which can then be used to rank candidate terms for set expansion.

Algorithm 1 Static Thresholding Algorithm

```
Static_Thresholding (seeds, graph)
for each term_i in graph.terms do
    Rel_Score[i] ← S_rel(term_i, seeds)
end for
sort term_i by Rel_Score[i] desc
K ← Pick_Threshold(Rel_Score[i])
R_0 ← the top K ranked terms by Rel_Score[i]
iter ← 1
while true do
    for each term_i in graph.terms do
        Sim_Score[i] ← S_rel(term_i, R_{iter-1})
        g(term_i) ← α * Rel_Score[i] + (1 − α) * Sim_Score[i]
    end for
    sort term_i by g(term_i) desc
    R'_iter ← the top K terms by g(term_i)
    if R'_iter ≠ R_{iter-1} then
        let r ∈ R'_iter be the top ranked term not in R_{iter-1}
        let q ∈ R_iter be the last ranked term in R_{iter-1}
        R_iter ← (R_{iter-1} ∪ {r}) − {q}
    else
        R_iter ← R_{iter-1}
        break
    end if
    iter++
end while
return R_iter
```

EXAMPLE 2. *Let $U = \{A, B, C, D, E, F\}$ be the set of 6 terms that we consider in this example, in which $S = \{A, B\}$ is the input seed set. Let the pair-wise similarity matrix M be*

$$M = \begin{array}{c|cccccc} & A & B & C & D & E & F \\ \hline A & 1 & 0.5 & 0.8 & 0.7 & 0.6 & 0.7 \\ B & 0.5 & 1 & 0.6 & 0.7 & 0.7 & 0.5 \\ C & 0.8 & 0.6 & 1 & 0 & 0.9 & 0.3 \\ D & 0.7 & 0.7 & 0 & 1 & 0.8 & 0.5 \\ E & 0.6 & 0.7 & 0.9 & 0.8 & 1 & 0.4 \\ F & 0.3 & 0.5 & 0.3 & 0.5 & 0.4 & 1 \end{array}$$

where each entry stands for the similarity score of the two corresponding nodes using some similarity metric $Sim : U \times U \rightarrow [0, 1]$.

For each term $term_i \in U$, we first compute its relevance score with the seed set, $S_{rel}(term_i, S)$. For example, $S_{rel}(A, S) = \frac{1}{2}(M(A, A) + M(A, B)) = 0.75$, $S_{rel}(B, S) = \frac{1}{2}(M(B, B) + M(B, A)) = 0.75$, etc. This gives rise to the $Rel\_Score[] = \{0.75, 0.75, 0.7, 0, 7, 0.65, 0.6\}$.

Next we invoke the thresholding algorithm (to be detailed in Section 4.2), which analyzes the score distribution to find a natural threshold point that separates the high scoring entities from the remaining background entities. We use the number of entities above the threshold as the estimate of the size of ESS. In this particular case, suppose the thresholding algorithm returns $K = 4$, which gives us $R_0 = \{A, B, C, D\}$.

In the first iteration ($iter = 1$), we compute for each term $term_i \in U$ its similarity score with the previous estimate of ESS, R_0, to derive $S_{rel}(term_i, R_0)$. For example, $S_{rel}(A, R_0) = \frac{1}{4}(M(A, A) + M(A, B) + M(A, C) + M(A, D)) = 0.75$, while $S_{rel}(B, R_0) = \frac{1}{4}(M(B, A) + M(B, B) + M(B, C) + M(B, D)) = 0.7$, so on and so forth. This leads to $Sim\_Score[] = \{0.75, 0.7, 0.6, 0.6, 0.75, 0.5\}$.

Given the weight constant $\alpha = 0.5$ as used in the quality metric, the ranking scores $g[] = \frac{1}{2} * Sim\_Score[] + \frac{1}{2} * Rel\_Score[] = \{0.75, 0.725, 0.65, 0.65, 0.7, 0.55\}$. Sorting the terms in U again, we get a different ordering (A, B, E, C, D, F). Let $R'_1 = \{A, B, E, C\}$ be the top ranked $K = 4$ terms in the new ordering. Given that $R'_1 \neq R_0$, D is the last ranked term in R_0, and E the top ranked term in R'_1 but not in R_0, we get $R_1 = R_0 - \{D\} + \{E\} = \{A, B, C, E\}$.

In the second iteration ($iter = 2$), we re-compute the similarity score $Sim\_Score[] = \{0.725, 0.7, 0.825, 0.55, 0.8, 0.375\}$. Combining that with $Rel\_Score[] = \{0.75, 0.75, 0.7, 0.7, 0.65, 0.6\}$ we have the new ranking scores $g[] = \{0.7375, 0.725, 0.7625, 0.625, 0.725, 0.4875\}$. The new top ranked $K = 4$ terms in U is $R'_2 = \{C, A, B, E\}$. Observe that $R'_2 == R_1$, we can now stop the iteration and use the ranking scores $g[]$ to generate a ranked list of terms, (C, A, B, E, D, F).

THEOREM 2. *The computation of R_{iter} in Algorithm 1 is guaranteed to converge, thus the while loop in Algorithm 1 is bound to terminate.*

Theorem 2 states the nice property that after a fixed number of iterations, the computation of R_{iter} will converge and stops changing for subsequent iterations. Here to outline the intuition of the proof of the convergence of our algorithm, we note that in our computation of ESS, we are implicitly maximizing the quality score of ESS. We show that this quality function will monotonically increase in each iteration, until reaching a local maximum, at which point it will converge and stop. Details of the proof can be found in Appendix 8.2.

While the algorithm is bound to converge, we don't have an upper bound on the number of iterations it may take before it stops. However in our experiments, we observe that it converges quickly and typically takes only a small number of iterations (less than 10).

The reason we term this algorithm *static thresholding* is due to the way the estimated size of ESS, K, is determined. In this static thresholding algorithm, once threshold K is computed in the first iteration, it will stay the same in subsequent iterations. In the following section, we will present a different variant of the algorithm in which K changes from iteration to iteration.

4.1.2 Dynamic Thresholding Algorithm

While the *static thresholding algorithm* described in Section 4.1.1 is proven to converge, its use of the static threshold (the parameter K in Algorithm 1) as computed in the first iteration may not accurately reflect the actual size of the ESS. It can be the case that in subsequent iterations with iterative score computation it becomes clear that based on the new score distribution, the new threshold value – which is interpreted as the size of the ESS – is significantly different from the initial estimate derived from the score distribution for the first iteration. To overcome this issue, we in this section propose a *dynamic thresholding algorithm* that iteratively use the new threshold value of the current score distribution to adjusts the estimated size of ESS. The algorithm is described in detail in Algorithm 2.

The structure of this algorithm is similar to Algorithm 1. We first compute the relevance score between each candidate term $term_i$ and the seeds. We then again invoke the thresholding procedure to find a good threshold value K_0, and use the top ranked K_0 terms as the initial ESS, R_0.

In each subsequent iteration, we again rank each term $term_i$, using the ranking function $g(term_i)$. Based on the new score distribution computed using $g(term_i)$, we re-invoke the automatic thresholding procedure to determine a new estimate of size of ESS, K_{iter}. Observe that instead of using the initial threshold K_0 computed in the first iteration as in the static thresholding algorithm,

Algorithm 2 Dynamic Thresholding Algorithm

Dynamic_Thresholding $(seeds, graph)$
for each $term_i$ in $graph.terms$ **do**
  $Rel\_Score[i] \leftarrow S_{rel}(term_i, seeds)$
end for
$K_0 \leftarrow$ **Pick_Threshold**$(Rel\_Score[i])$
sort $term_i$ by $Rel\_Score[i]$ desc
$R_0 \leftarrow$ the top ranked K_0 terms by $Rel\_Score[i]$
$iter \leftarrow 1$
while $iter \leq$ *MAX_ITER* **do**
  for each $term_i$ in $graph.terms$ **do**
    $Sim\_Score[i] \leftarrow S_{rel}(term_i, R_{iter-1})$
    $g(term_i) \leftarrow \alpha * Sim\_Score[i] + (1-\alpha) * Rel\_Score[i]$
  end for
  $K_{iter} \leftarrow$ **Pick_Threshold**$(g(term_i))$
  sort $term_i$ by $g(term_i)$ desc
  $R'_i \leftarrow$ the top ranked K_{iter} terms by $g(term_i)$
  $R_{iter} \leftarrow R'_{iter}$
  $iter++$
end while
return R_{iter}

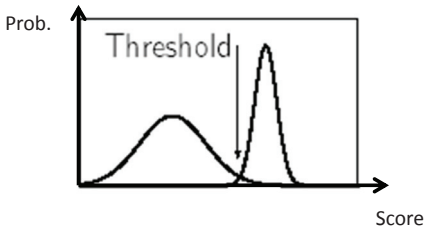


Figure 3: Thresholding to separate two score distributions

we recompute the threshold based on the new score distribution. This dynamic thresholding technique adapts to the changes in the score distribution and may be able to reflect the size of the ESS more accurately. In practice we observe that this algorithm slightly outperforms the static thresholding algorithm.

However, since we are dynamically changing the thresholding value in the dynamic thresholding algorithm, we cannot guarantee the convergence property as in the static thresholding algorithm. Therefore we place a loop-termination condition which is the maximum number of iterations to execute for efficiency consideration. In practice we use small number of iterations (*e.g.*, 5) and observe reasonable performance.

4.2 Automatic Score Thresholding

The subproblem we look at in this section, is to automatically determine the natural threshold that best separates two underlying score distributions from one score distribution. This problem arises when we have a set of scores, each of which represents an estimation of the likelihood of the term being a member of the "concept set" we are trying to uncover. The assumption here is that those terms that really belong to the "concept set" will have higher scores, and follows some kind of score distribution as in the right curve in Figure 3; while those that do not belong to the set will have lower scores, but also follow a score distribution as the left curve in Figure 3. Under this assumption, the problem becomes the classical score thresholding problem, and we tap into existing literature to solve this thresholding problem.

In particular, the same problem arises in image segmentation in the computer graphics, where the goal is to separate the foreground image from the background images. For example, in Figure 4, the

Figure 4: Automatic thresholding to segment images

image on the left has a hand in the foreground and a dark background. The task of image segmentation is to separate the foreground hand from the background to get the image on the right hand side. Each pixel in the image has a gray scale value that again is assumed to follow two distributions: those belong to the foreground and those belong to the background.

In the computer graphics literature, a number of thresholding algorithms have been proposed and shown to be effective, including the Iterative Threshold Selection [12] and Otsu's thresholding [10]. We in this work adopt the Otsu's thresholding to find the good threshold point that naturally separates those high scoring terms from those background terms. Formally, Otsu's threshold is defined as follows.

DEFINITION 7. *Let ω_1, ω_2 be the probabilities of the two classes separated by a threshold t, and σ_1^2, σ_2^2 be the variances of these two classes. The weighted sum of the variances of the two classes is $f(t) = \sigma_w^2(t) = \omega_1(t)\sigma_1^2(t) + \omega_2(t)\sigma_2^2(t)$. The **Otsu's threshold** T is defined as the one that minimizes $f(t)$, or equivalently, $T = \arg\min_x(f(x))$.*

Briefly, Otsu's technique sees the two sets of scores separated by the threshold as two clusters. It is based on the observation that the threshold that best separates the two clusters is the point with the least intra-cluster variances. Therefore, Otsu's thresholding uses the sum of the two intra-cluster variances as the objective function, and searches for the point that minimizes the sum of the intra-cluster variances as the threshold. We observe that in our experiments, this thresholding technique outperforms alternative thresholding techniques we experimented with. Given a sorted list of scores, the Otsu's threshold can be computed linearly.

4.3 Discussions

Having described the details of our algorithm, here we discuss some major differences compared with existing random walk based approaches.

Our algorithms in its essence take an iterative score computation process, which is very similar to, say random walk based ranking algorithms which also compute scores iteratively. How is our iterative score computation algorithm different from the general random walk? More importantly, as we have noted, we observe in our experiments that our algorithm outperforms the random walk based approaches. It is thus interesting to explore the differences between our approach and the random walk based algorithms that lead to the divergence in performance.

First, we would like to point out the key similarities between our iterative score computation and random walk algorithms. If we were to build a $N \times N$ similarity matrix M where each entry (x, y) denotes the similarity of terms x and y on the left-side of the bipartite graph, this matrix essentially represents a graph between terms (instead of graph between terms and contexts). In addition, in each iteration where we aggregate score computation, we are essentially doing a matrix multiplication $\alpha M \times V + (1-\alpha) M \times V'$ in which

V is the $(0, 1)$ vector of size N that represents the given seeds, and V' is a $(0, 1)$ vector that represents the ESS in the previous iteration. The computation framework bears some similarity with random walk based approach.

However, we note that there are at least three important aspects that differentiate our algorithm from the random walk based approaches. First, we use a score thresholding mechanism as discussed in Section 4.2 to find a good ESS (equivalently, the multiplication vector V), which tends to be very small in size (comparing to the set of universal terms), as opposed to normal random walk that does not have such score thresholding. As the result, we only propagate probability (or, scores) through confident nodes. Secondly, we normalize scores in each iteration differently. The aggregate score we compute is a weighted sum of the score computed against the given seeds and the score computed against the ESS in each iteration (catching both relevance and coherence), which is different from typical random walks that only take care of relevance to the seeds. Lastly, we treat each entity in the ESS equally and reset their weights to be "1" after each iteration, which is again different from random walks. Given the specific set-expansion application we are targeting at, where each entity in the set should be conceptually treated equally, the reset of weight is reasonable given a robust automatic score thresholding mechanism. It turns out in our experimental evaluation that these aspects are critical to achieving reasonable set expansion performances.

5. EXPERIMENTS

5.1 Experimental Setup

5.1.1 Data Set Preparation

As we have alluded before, we experiment with two types of data, the web lists and the query log. Both of them are modeled as bipartite graph as described in Section 3.1. In particular, for the web list data, we use crawlers to extract around 6 million lists in HTML pages from the web, and the resulting bipartite graph has $58,550,245$ edges. For the query log data, we obtain a sample of user queries from Bing, and the resulting bipartite graph has $94,859,646$ edges.

To evaluate the effectiveness of our algorithms, we select four sets of concepts over which set expansion experiments are conducted, namely, country names, colors, camera brands and mattress brands. The "ground truth", or the set of entities that is considered to belong to each concept set are determined as follows. For country names and colors, we resort to Wikipedia and use the list of countries in [2] and the list of atomic web browser colors in [3], respectively. For camera brands and mattress brands, we obtained the manually created lists from domain experts. We selected these four categories because (1) they are across different domains; and (2) they have different degree of difficulty for set expansion.

In order to compare the performance of the our algorithms and the existing techniques, we randomly pick 6 entities from the ground truth data for each concept set as the input starting seeds. Since each algorithm returns a ranked list of results, we evaluate the performance of these competing algorithms by measuring the precision/recall values of each algorithm at different rank positions.

5.1.2 Algorithms/Systems Compared

We conduct three broad groups of experiments to thoroughly understand the effectiveness of our set expansion algorithms. In the first set of experiments we compare the algorithms proposed in this work with the state-of-art random-walk based ranking algorithms used for set expansion, over the same web list/query log data. We implemented two random-walk based ranking algorithm, the regular random-walk with fixed teleport probability as used in [17, 18, 19], and the Adsorption random walk algorithm [5, 15, 20], which essentially is a variant of the regular random-walk, which penalizes popular nodes by customizing the teleport probability based on the edge-degree of each node. In both cases the random walk is performed on the bipartite graph model as described in Section 3.1.

In addition, we conduct a second group of experiments that compares our algorithms with the two existing software set expansion systems, namely the SEAL set expansion system [16] and the Google Sets system [1] through their respective public web portals. Since SEAL interface only accepts up to 3 seeds, we pick 3 entities as seeds in this set of experiments.

Lastly, we drill down to the dynamic thresholding algorithm (the behavior of the static thresholding algorithm is similar) proposed in this work, and present a set of experiments in which we vary various parameters. Specifically, we vary the parameter α, the number of seeds used, and the similarity metric used, etc. This allows us to analyze and to better understand the performance characteristics of our algorithm in response to the changes in parameter values.

5.2 Experimental Results

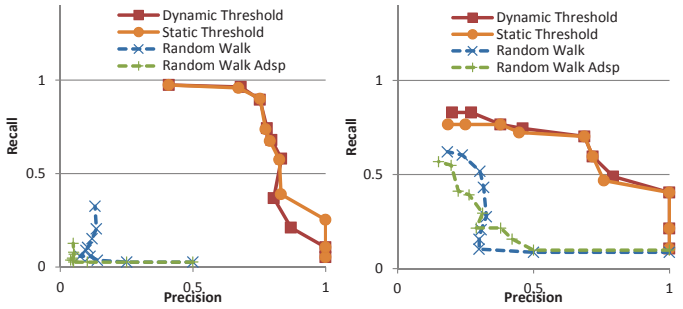
5.2.1 Example Output

Country	Color	Camera	Mattress
germany	**blue**	**olympus**	**serta**
japan	**red**	**nikon**	**sealy**
belgium	**white**	**kodak**	**simmons**
denmark	**green**	pentax	beautyrest
canada	**yellow**	**canon**	sealy posturepedic
spain	brown	**casio**	stearns foster
netherlands	black	**fuji**	**bassett**
hungary	purple	panasonic	tempur pedic
italy	pink	leica	**sears**
czech republic	gray	fujifilm	spring air
poland	orange	sony	gold bond
sweden	**violet**	ricoh	jobri
norway	silver	vivitar	universal furniture
austria	grey	sigma	champlain
peru	gold	konica minolta	fbg
croatia	greyish green	samsung	coapt systems
switzerland	bronze	minolta	riverside furniture
slovakia	beige	polaroid	hyla vacuum
china	navy	vistaquest	broyhill
slovenia	tan	rollei	lifestyle solutions

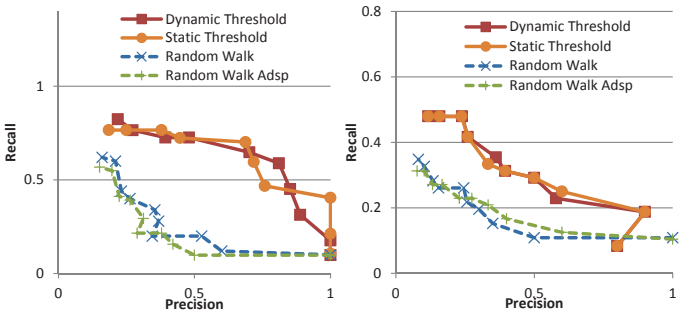
Table 1: Top-20 results by Dynamic Thresholding algorithm

Table 1 lists the top 20 ranked results produced by dynamic thresholding algorithm for the four domains that we experiment with. The static thresholding algorithm reports similar results, as we will see later in the precision/recall curves. In each domain, those terms in boldface are the input seeds. The underlined terms are the results that do not belong to the ground truth set and thus counted as incorrect results; while the remaining terms are correct results expanded from the input seeds.

From Table 1, we can see that in the top-20 ranked results, the "Country" and "Camera" domains have perfect precision. "Color" domain has only one incorrect result "greyish green" (which although being a real composite color, is nonetheless not included in our ground truth set which only includes atomic colors). The top-20 results for "Mattress", however, includes many noisy entires that are incorrect, including product line names ("sealy posturepedic" and "beautyrest"), and furniture retailers ("universal furniture" and "riverside furniture").

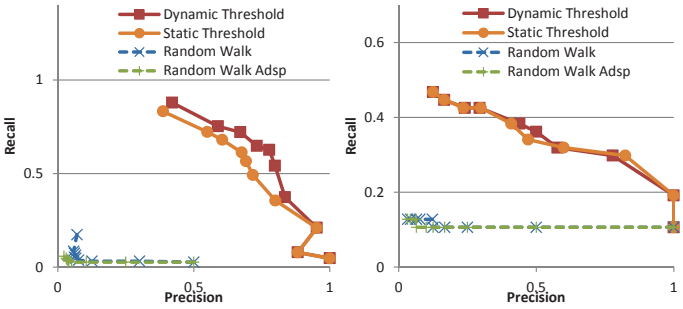


(a) Country (b) Color

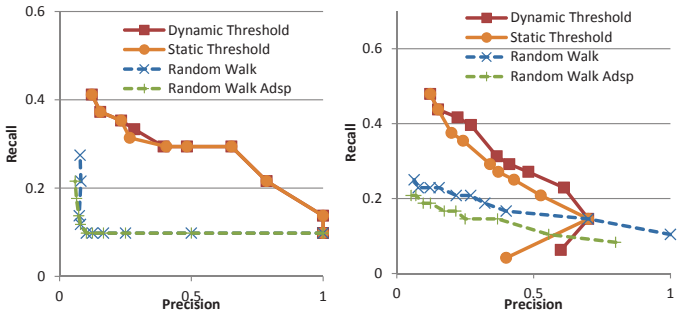


(c) Camera (d) Mattress

Figure 5: Comparison with Random-Walk based approach using Web List data



(a) Country (b) Color



(c) Camera (d) Mattress

Figure 6: Comparison with Random-Walk based approach using Query Log data

Observing the patterns of the incorrect results in the ranked list, we further adopt a token-based subset/superset-filtering heuristics to remove results that are likely to be incorrect. Specifically, we remove a result in the ranked list if there is another result that ranks higher in the list whose token set is a subset/superset of the current result. The intuition here is that for a pair of results whose token sets are subset-superset, the concepts that they represent also tend to constitute a semantic superclass-subclass hierarchy. Given that we are expanding the given seeds to produce a coherent concept set, one and at most one of the two entities in the superclass-subclass hierarchy can be correct. Therefore, we only pick the result that ranks higher (a more confident prediction) and ignore subset/superset results that rank lower. As an example, since "sealy" ranks higher in the list for "Mattress", we will not consider the superset result "sealy posturepedic" that ranks lower, which is only a product line, or a subclass of the desired superclass concept, the manufacturer/brand name "sealy". Similarly "greyish green" will not be considered since the subset "green" is also a result that ranks higher. This simple heuristic turns out to be useful and boosts the precision/recall results. We apply this token-based subset/superset-filtering for all algorithms/systems in our experiments.

5.2.2 *Comparison With Random Walk*

In this section we present the performance comparison between the iterative thresholding algorithm proposed in this work, and the random walk based ranking algorithms.

For the random walk based approaches, we report experimental results for both the regular random walk [17, 18, 19], and the Adsorption random walk [5, 15, 20], on the bipartite graph as modeled in Section 3.1. They are reported as the "Random Walk" and "Random Walk Adsp" curves. We set teleport probability as 0.2.

Figure 5 shows the performance results on the Web List data for each of the four domains that we experimented with. Note that while Dynamic Thresholding is slightly better than Static Thresholding, both approaches significantly outperform random walk based approaches. This confirms that simple random walk based approach is not able to handle noisy data well, as we analyzed in Section 4.3.

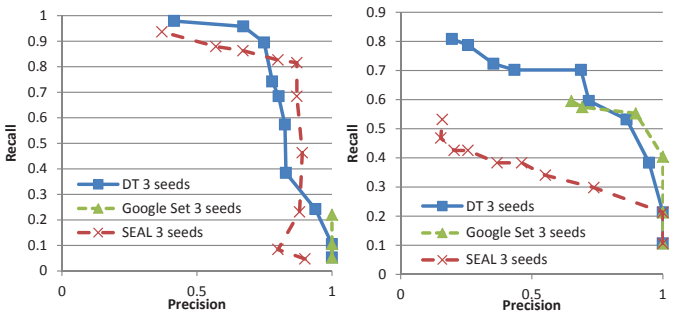
Figure 6 presents the same experiments run on the Query Log data. The general trends observed in Figure 5 is confirmed: Dynamic Thresholding and Static Thresholding outperforms the random walk based approaches quite significantly. We note, however, that the precision/recall results observed on Query Log data are not as good as the those on Web List data. This is not entirely surprising, as the query log tends to be much noisier than the HTML web lists.

5.2.3 *Comparison with Google Sets and SEAL*

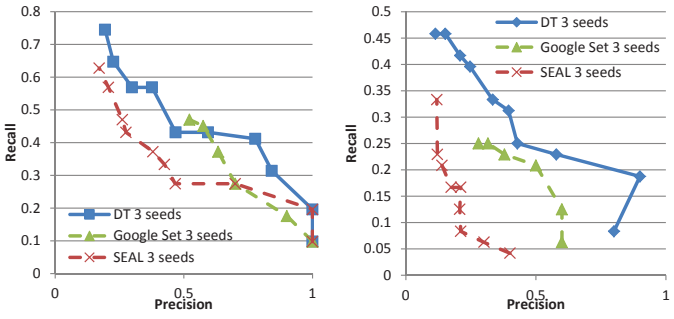
We additionally conduct a second group of experiments that compares our algorithms against two existing systems, SEAL [16] and Google Sets [1] [1]. As we stated previously, since we do not have complete details of the implementation of the algorithms or the back-end data set used by either of the SEAL/Google Sets systems, the performance numbers do not directly compare. However, we still feel that reporting these results is useful in that it puts the performance of our algorithm in the context of start-of-art existing systems, and helps us to understand the usefulness of the algorithms.

Figure 7 summarizes the performance comparison between Dynamic Thresholding that we propose, and SEAL/Google Sets. Since web interface for Google Sets can take up to 5 seeds, while SEAL can only allow for 3 seeds as input, in this set of experiments we only use 3 seeds as input to all three algorithms. In addition, as Google Sets only returns 50 results at most, its performance curve is incomplete (especially for the "Country" domain, where we report precision/recall up to the top-ranked 500 terms given that the

[1] performance numbers for SEAL and Google Sets were obtained on 10/01/2010

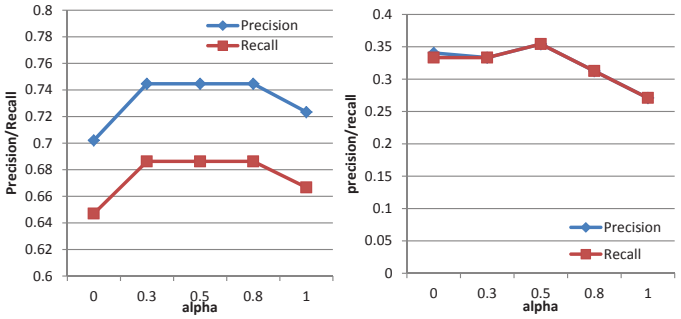


(a) Country (b) Color



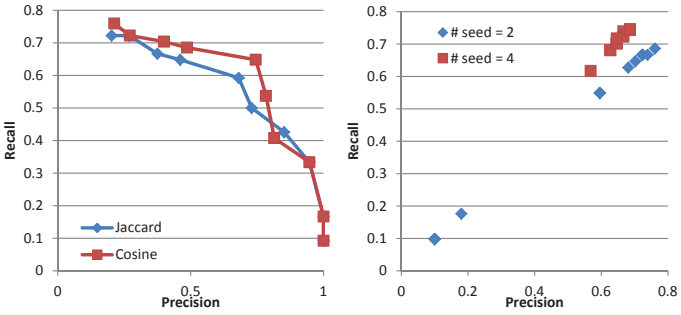
(c) Camera (d) Mattress

Figure 7: Comparison with Google Sets and SEAL



(a) Camera (b) Mattress

Figure 8: Sensitivity analysis to α



(a) Vary similarity metric (b) Vary # of seeds

Figure 9: Sensitivity analysis

ground-truth set is much larger for "Country"). The general observation is that while Dynamic Thresholding and Google Sets perform roughly the same in "Country" and "Color", Dynamic Thresholding has a slightly better precision/recall curve for "Camera" and "Mattress". Furthermore, in each of the four domains Dynamic Thresholding seems to outperforms the SEAL system. Nevertheless, we once again emphasize that this is by no means an implication of the relative performance of these algorithms. The difference in performance may simply because of the different data sets each system uses. It does suggest, however, that the algorithm we develop is competitive against existing set-expansion systems.

5.2.4 *Sensitivity Analysis to Parameters*

To better understand the performance characteristics of our proposed approaches, we in this section conduct sensitivity analysis to understand the impact of various parameters to our algorithm. Again, we use the dynamic thresholding algorithm as example.

Figure 8 depicts the performance of the Dynamic Thresholding algorithm with varied parameter α in domain "Camera" and "Mattress". Recall that in Definition 4, α is the essentially the weight parameter used to balance the quality metric of the expanded seed set (ESS) between *relevance* and *coherence*. $\alpha = 0$ means that we only consider the *coherence* of ESS, while $\alpha = 1$ indicates that only *relevance* is taken into account. Any value of α in between suggests a combination of both of these two metrics. In both Figure 8a and Figure 8b we can see that when α gets extreme values (0 or 1, meaning only one of the *relevance* and *coherence* metrics is considered), precision/recall performance numbers suffer. On the other hand, α values in between boosts the performance of our algorithm. While we only report performance for domain "Camera" and "Mattress" here, similar trends are observed in other domains. In general, we observe that $\alpha = 0.5$ usually gives the best performance in most cases, and adopt $\alpha = 0.5$ in all remaining experiments.

Next, in Figure 9a we vary the similarity metric used in our Dynamic Thresholding algorithm for the "Camera" domain. Recall that our algorithm is a general framework that can work with any similarity measurements. In this experiment we report performance of our algorithm using both Jaccard similarity and the Cosine similarity metric. As can be seen in Figure 9a, while both approaches perform reasonably well, the Cosine similarity is slightly better. Similar trends are also observed in experiments over other domains, confirming the performance advantage of using Cosine similarity. Therefore, we only report performance using Cosine similarity metric in all other experiments.

Finally, in Figure 9b, we vary the number of input seeds and report the corresponding set expansion performance. Specifically, given the 6 seeds we used for each of the four domains in the previous experiments, we pick all possible 2-seed combination out of these 6 seeds, which gives us a total of 15 such combinations. Similarly we pick all possible 4-seed combination out of the 6 seeds which again gives us 15 possibilities. We then use all 15 combinations of 2/4 seeds as input, to test the performance of our Dynamic Thresholding algorithm. The results are reported in Figure 9b. The overall trend that stands out in this figure is that the performance of our algorithm with 4 seeds is in general much better and more stable than the case where only 2 seeds are used as input. Observe that to the lower left corner, there are two instances of 2-seed combinations that lead to extremely low precision/recall. This suggests that our algorithm is more robust when a reasonable number of seeds are given, and the performance may fluctuate with very few number of seeds, largely depending on the quality of the seeds given. However, we believe that it is not really hard to find 4 input seeds in virtually any domain, thus not a stringent requirement in general, ensuring the usefulness of our algorithm.

6. CONCLUSION

In this paper we studied the problem of using general-purpose web data (web lists and query logs) to expand a set of seed entities. We proposed a simple yet effective quality metric to measure the expanded set, and designed two iterative thresholding algorithms to rank candidate entities. We validated our approach using experiments conducted on multiple domains, and concluded that our algorithm outperforms existing techniques for set expansion on noisy web data.

7. REFERENCES

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

[2] List of United Nations member states. http://en.wikipedia.org/wiki/united_nations_member_states.

[3] Web colors. http://en.wikipedia.org/wiki/web_colors.

[4] E. Agichtein and L. Gravano. Snowball: extracting relations from large plain-text collection. In *JCDL*, 2000.

[5] S. Baluja, R. Seth, D. Sivakumar, Y. Jing, J. Yagnik, S. Kumar, D. Ravichandran, and M. Aly. Video suggestion and discovery for youtube: Taking random walks through the view graph. In *WWW*, 2008.

[6] O. Etzioni, M. Cafarella, D. Downey, S. Kok, A. Popescu, T. Shaked, S. Soderland, D. S. Weld, and A. Yates. Web-scale information extraction in KnowItAll. In *WWW*, 2004.

[7] 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. In *Artifical Intelligence*, 2005.

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

[9] R. M. Karp. Reducibility among combinatorial problems. *Complexity of Computer Computations*, 1972.

[10] N. Otsu. A threshold selection method from gray-level histograms. *IEEE Transactions on Systems, Man and Cybernetics*, 1979.

[11] M. S. Pang-Ning Tan and V. Kumar. *Introduction to Data Mining*. 2005.

[12] T. Ridler and S. Calvard. Picture thresholding using an iterative selection method. *IEEE Transactions on Systems, Man and Cybernetics*, 1978.

[13] B. Settles. Biomedical named entity recognition using conditional random fields and rich feature sets. In *CoLING*, 2004.

[14] P. P. Talukdar, T. Brants, M. Liberman, and F. Pereira. A context pattern induction method for named entity extraction. In *CoNLP*, 2006.

[15] P. P. Talukdar, J. Reisinger, M. Pasca, D. Ravichandran, R. Bhagat, and F. Pereira. Weakly-supervised acquisition of labeled class instances using graph random walks. In *EMNLP*, 2008.

[16] R. Wang and W. Cohen. SEAL: http://rcwang.com/seal.

[17] R. Wang and W. Cohen. Language-independent set expansion of named entities using the web. In *ICDM*, 2007.

[18] R. Wang and W. Cohen. Iterative set expansion of named entity using the web. In *ICDM*, 2008.

[19] R. Wang and W. Cohen. Character-level analysis of semi-structured documents for set expansion. In *EMNLP*, 2009.

[20] Y.-Y. Wang, R. Hoffmann, X. Li, and J. Szymanski. Semi-supervised learning of semantic classes for query understanding. In *CIKM*, 2009.

8. APPENDIX

8.1 Proof of Theorem 1

We show that there exists a polynomial-time reduction from the Maximum-Clique problem to the problem we stated in Section 3.4.

For any given graph $G = \{V, E\}$ for which the Maximum-Clique needs to be computed, we can build a corresponding similarity matrix M of size $|V| \times |V|$, where each row r_i corresponds to vertex $v_i \in V$, and each column c_j corresponds to vertex $v_j \in V$. The matrix entry $M(r_i, c_j)$ is 1 if there is the edge $(v_i, v_j) \in E$, otherwise $M(r_i, c_j) = 0$.

Given this construction of this similarity matrix M, we prove the claim by contradiction. Suppose there is an algorithm A that efficiently finds the optimal R of size K with maximum quality $Q(R, S)$. If we assign α to be 0, the decision problem of finding Maximum-Clique of any graph G can be solved using the corresponding similarity matrix M and algorithm A. This is because the optimal set R computed by A with maximum quality score ($\frac{|R|-1}{2|R|}$) must corresponds to a clique in the original graph. This contradicts with the hardness of the Maximum-Clique problem, hence the hardness of the problem we stated in Section 3.4.

8.2 Proof of Theorem 2

To prove that the Algorithm 1 will terminate and the computation of R_{iter} converges, we show that the quality metric $Q(R_{iter}, S)$ is monotonically increasing with the number of iteration $iter$.

Let $R = \{r_1, r_2, ..., r_{K-1}, r_K\}$ and $R' = \{r'_1, r'_2, ..., r'_{K-1}, r'_K\}$ be the ESS of two subsequent iterations. Without loss of generality, let $r_i = r'_i$ for $1 \leq i \leq K-1$, and denote $\tilde{R} = \{r_1, r_2, ..., r_{K-1}\} = \{r'_1, r'_2, ..., r'_{K-1}\}$, such that we have $R = \tilde{R} \cup \{r_K\}$ and $R' = \tilde{R} \cup \{r'_K\}$. Let $g(r_j, R, S)$, and $g(r_j, R', S)$ be the ranking function of r_j against R and R' respectively. Observe that the quality metric

$$Q(R,S) = \frac{\alpha}{|R| \cdot |S|} \sum_{i=1}^{|S|} \sum_{j=1}^{|R|} Sim(s_i, r_j) + \frac{(1-\alpha)}{|R| \cdot |R|} \sum_{i=1}^{|R|} \sum_{j>i}^{|R|} Sim(r_i, r_j)$$

and similarly

$$Q(R',S) = \frac{\alpha}{|R'| \cdot |S|} \sum_{i=1}^{|S|} \sum_{j=1}^{|R'|} Sim(s_i, r'_j) + \frac{(1-\alpha)}{|R'| \cdot |R'|} \sum_{i=1}^{|R'|} \sum_{j>i}^{|R'|} Sim(r'_i, r'_j)$$

.

The difference of the quality function in two subsequent iterations are thus

$$\begin{aligned}
&Q(R',S) - Q(R,S) \\
=&\frac{\alpha}{|R| \cdot |S|}\Big(\sum_{i=1}^{|S|} Sim(s_i, r'_K) - \sum_{i=1}^{|S|} Sim(s_i, r_K)\Big) \\
&+ \frac{(1-\alpha)}{|R| \cdot |R|}\Big(\sum_{i=1}^{K-1} Sim(r'_K, r_i) - \sum_{i=1}^{K-1} Sim(r_K, r_i)\Big) \\
\geq&\frac{\alpha}{|R| \cdot |S|}\Big(\sum_{i=1}^{|S|} Sim(s_i, r'_K) - \sum_{i=1}^{|S|} Sim(s_i, r_K)\Big) \\
&+ \frac{(1-\alpha)}{|R| \cdot |R|}\Big(\sum_{i=1}^{K} Sim(r'_K, r_i) - \sum_{i=1}^{K} Sim(r_K, r_i)\Big) \\
\geq&\frac{1}{|R|}(g(r'_K, R, S) - g(r_K, R, S)) \\
>&0.
\end{aligned}$$

In other words, after each iteration the quality of the new ESS $Q(R', S)$ is strictly greater than that of the previous iteration $Q(R, S)$. Since for each iteration $iter$, $R_{iter} \subseteq U$, the number of different R_{iter} is finite. Thus we know the algorithm will terminate and the computation of R_{iter} converges.