Using Wikipedia to Identify Relational Knowledge for Textual Inference

ABSTRACT

Many data and knowledge management problems require some sort of textual inference. Some fundamental forms of inference—determining whether a piece of text implies or contradicts another piece of text—depend on the ability to recognize basic relations between concepts. For example, identifying that *Toyota* is a kind of *Car*, or that *trout* and *bass* are both kinds of *fish*. While there has been a large body of work on the problem of identifying relations between concepts that occur in a given sentence or corpus, this is not useful in identifying such "common sense" relations between two given concepts—many pairs of related concepts are unlikely to ever appear in close proximity in any document.

In this paper, we present a novel and robust approach for the problem of identifying relations between pairs of concepts. We focus on identifying relations that are essential to supporting textual inference: determining whether two concepts hold the ancestor relation or whether they are siblings. Our method makes use of Wikipedia as a main source for background knowledge, but we also propose an effective approach to improve the coverage of our method and support inference between concepts that are not mentioned in Wikipedia. Our experimental analysis shows that our approach outperforms systems that use other existing resources as background knowledge.

Categories and Subject Descriptors

H.4 [**Knowledge Management**]: Information Extraction

General Terms

Relation identification

Keywords

Concept, relation, classification, Wikipedia

1. INTRODUCTION

Many data and knowledge management problems require some sort of textual inference. These range from expansion and query interpretation in information retrieval to query schema matching to question answering. Textual Inference, in turn, requires the use of large amounts of background knowledge. For example, it may be important to know that a *blue Toyota* is not a *red Toyota* nor a *blue Honda* but that all are cars, and even Japanese made cars.

The fundamental problem expressed here is that of recognizing basic relations between two given concepts. This view of recognizing relations is different from the Open Information Extraction effort [1] and On-Demand Information Extraction [20], which aim to extract large databases of open-ended facts and build the two concepts occurring together in some local context. It is also different from the effort on supervised relation extraction[18] which requires additional supervised data to learn new relations.

Moreover, a textual inference system that needs to know if concepts A and B are related, and how, cannot benefit from the large body of literature on knowledge acquisition that has extracted all *easy to find* facts in a given corpus [2, 5, 15], since it will know of A and B only if they have occurred in an explicit way and in close proximity in a sentence. Indeed, we know of no successful application of the large scale existential knowledge acquisition efforts to textual inference.

In the context of Textual Inference [4, 9, 3], it has been argued, e.g., in [12] that many inferences are largely compositional and depend on the ability to recognize specific relations between concepts such as entities, noun phrases, etc. For example, it is often necessary to know of an *ancestor* relation and its directionality, in order to deduce that a statement with respect to the *child* (e.g., *cannabis*) holds for an *ancestor* (e.g.,*drugs*). This is illustrated in the following example, taken from the RTE4 test data:

> **T**: Nigeria's National Drug Law Enforcement Agency (NDLEA) has seized 80 metric tonnes of *cannabis* in one of its largest ever hauls, officials say.
> **H**: Nigeria seizes 80 tonnes of *drugs*.

Similarly, it is often important to know of a *sibling* relation to infer that a statement about *Taiwan* may *contradict* an identical statement with respect to *Japan* (at least, without additional information) since these are *different* countries. This is illustrated in the following example from RTE4 test data.

> **T**: A strong earthquake struck off the southern tip of *Taiwan* at 12:26 UTC, triggering a warning from Japan's Meteorological Agency that a 3.3 foot tsunami could be heading towards Basco, in the Philippines.
> **H**: An earthquake strikes *Japan*.

This paper proposes to address the problem of relation identification and classification in a form that is directly applicable to textual inference. Specifically, our system accepts two input concepts as arguments (these could be entities or noun phrases) and identifies the relation between them . For

WWW2010, April 26-30, 2010, Raleigh, North Carolina.
WOODSTOCK '97 El Paso, Texas USA
.

example, we identify that *global warming* and *food crisis* are in a *sibling* relation, and the concept of *economic problems* is in an *ancestor* relation with both of them.

We focus here on the *ancestor* relation and the *sibling* relation, that were identified as key relations also in [12] (the *sibling* relation is called an *alternation* there, and our *ancestor* relation is called *forward entailment* and *backward entailment*). Following the logic developed there, we expect that the resource we develop in this work can be used compositionally to support robust textual inference.

We develop an approach that makes use of Wikipedia that given a pair of concepts, can, on the fly, recognize and classify the relation between these concepts. While Wikipedia is vast resource that is rapidly being updated, our approach needs to take into account that it is developed by a large number of people, and is thus very noisy and non-uniform. Our algorithmic approach therefore treats Wikipedia and its category structure as an open resource and uses statistical text mining techniques to get robust information from this noisy resource. For example, the concept *Ford* appears many times in Wikipedia, and is part of a large number of categories. As a *president*, mentions of *Ford* are inconsistent with mentioned of other presidents. However, it appears also in other senses, related to the *car* industry, for example. We need to disambiguate it and determine which category it belongs to and, within this category, which specific concept is intended. In order to disambiguate it, we make use of the context provided by the concept pair—*Ford* in (*Ford*, *Nixon*) is probably different than the one in (*Ford*, *Chevrolet*) and even from the one in (*Ford*, *Iacocca*).

Textual inference is driven by background knowledge. Therefore, the notion of *prominence* is an essential notion in supporting textual inference. Most people *know* that *Michael Jordan* is a former NBA player, and do not know the *Michael Jordan* who goes to school with the authors. Consequently, unless additional knowledge is given, a textual inferences system should assume that *Michael Jordan* ia a basketball player. This is the reason we use Wikipedia as our background knowledge source. Moreover, under the assumption that in textual infernce applications we are in search of some notion of "common sense" knowledge we make use of a notion of prominence with respect to a given text collection (in this case, with respect to Wikipedia itself). Clearly, while Wikipedia has broad coverage that is sufficient for many application, one may want to go beyond it in some cases. We suggest a simple technique to makes use of the web to do that, and show its effectiveness when at least one of the target concepts is not mentioned in wikipedia.

We measure the performance of our system over a large number of pairs chosen from over 40 semantic classes. We compare it with other large scale efforts to identify relations between concepts. For example we show that, even when all concepts are covered by the *extended WordNet* [22], our system still significantly outperforms that system.

The key contributions of this paper are (i) the definition of a relation identification problem that is directly relevant to supporting textual inference and (ii) the development of a robust and accurate machine learning based approach to address this problem. We show that this approach has significantly better coverage and accuracy than other approaches that can only identify relations between concepts that occur in an explicit way and in close proximity or other Wikipedia-based approaches. Notably, our algorithmic approach is trained with a small number of annotated examples and generalizes well across semantic classes.

2. PREVIOUS WORK

Several lines of work share some similarities to the work presented here. One line of work makes use of the web, and exploits the fact that many relations are expresses explicitly and locally, typically within a single sentence. It uses this to discover terms which have hypernym or coordinate relation [10, 22] or to expand or refine members in a semantic class (given seeds or the semantic class itself) [17, 25, 11, 26, 2]. A second line of work attempts to classify pre-specified relations [28, 20, 18] and typically makes use of supervised or semi-supervised learning techniques.

The later line of work is very different from ours in terms of goals and techniques. It consider relations such as "SpouseOf" of "BornIn" and relies on the relation being expressed in close proximity, typically within a sentence, and on training supervised classifiers. The work in the first line of work is more similar to ours in terms of the final task, but is very different in the assumptions and the techniques—mostly using the fact that due to redundancy of information in the web, a lot of related terms with appear often enough in some simple explicit form in close proximity. In [21], the authors construct a hypernym-only classifier that builds on dependency path patterns discovered in sentences that contain noun pairs in hypernym and hyponym relations. Furthermore, they show that using coordinate terms can help improve the recall of the hypernym classifier. The best system in their work is a hypernym-only classifier additionally trained with hundred thousands of dependency paths extracted from Wikipedia corpus. The system outperforms the best WordNet classifier in identifying coordinated terms by relatively improving F-score by over 54%. More recently, [22] proposed a method to train their (m,n)-cousin relationship classifier (defined similarly to our *sibling* relation) to significantly increase the coverage and improve the F-score by 23% over the WordNet-2.1 hypernym classifier.

Using similar techniques to the aforementioned line or work, there has been a large body of work on automatically expanding a set of entities given some seeds that belong to a semantic class or the class itself. A typical application of this research is that of expanding, correcting or refining Google Sets[1]. The common approach is to search the web for patterns around the seeds and bootstrap to other semantically similar members of the class [11, 24]. On the other hand, [27, 26] use surrounding contexts of seeds in semi-structured text to extract other members of the set. Similarly, [14, 15] automatically acquirew open-domain classes of entities and attributes by using very little supervised seed information.

The key difference between this line of work and the approach we develop here is that we are given two concepts and are required to determine the relation between them. The coverage of the aforemention methods is lacking since they rely on concepts appearing in close proximity and with specific patterns that reveal their relation. It is likely, however, that our input concepts never occur together in a given sentence yet, it is well known that they are related. As we show, we can robustly and accurately determine these relations, significantly outperforming state of the art acquisition methods such as [22]. [16] has a related goal of generating

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

a semantic taxonomy using wikipedia as knowledge source, but it covers only around 20% of the relations in wikipedia resulting, as we show, in a rather poor coverage of relations between common concepts.

2.1 The differences with YAGO ontology

Below are main points for the differences with YAGO ontology. - - -

3. IDENTIFY RELATIONAL KNOWLEDGE

Our approach is a machine learning based algorithm. We first present an algorithm to disambiguate given concepts. We then extract features of disambiguated concepts and train a supervised multi-class classifer using Wikipedia as the background knowledge to identify relations between concepts. Given a pair of concepts (X, Y), we focus on addressing the following relation types:

1. X is an ancestor of Y, denoted by $X \leftarrow Y$.
2. Y is an ancestor of X, denoted by $X \rightarrow Y$.
3. X and Y are siblings, denoted by $X \leftrightarrow Y$.
4. X and Y have no relation, denoted by $X \nleftrightarrow Y$.

We also propose a simple and effective method to improve the coverage of our system by going beyond Wikipedia to identify relations between concepts that are not mentioned in Wikipedia.

3.1 Concept Disambiguation

By posing the problem as identifying relations, we have to deal with the problem of disambiguating given concepts. The more accurate the disambiguation is, the more precise the relation recognition. For example, the concept *ford* in the concept pair *(bush, ford)* may refer to several concepts, such as *Ford Motor Company*, president *Gerald Ford*, or *Ford* city in Wisconsin - USA. Our approach is motivated by observing that the concept *bush* can be used to disambiguate *ford* and vice versa. Eventually, we can put two concepts *George W. Bush* and *Gerald Ford* in the top retrieved lists of *bush* and *ford*.

We use Explicit Semantic Analysis (ESA) [8] to generate a list of top semantically relevant articles in Wikipedia for both given concepts. For each article in the top list, we extract and keep its categories. All categories are then tokenized to create a bag of tokens relevant to the two given concepts. We use TF-IDF score to weigh the importance of each token. Only the top important tokens, which are different from the two input entities, are kept to use in the next step of our algorithm. Each given concept is then concatenated with the top weighed tokens to create a new query. We use the new queries to search titles and texts of Wikipedia articles to get the top relevant articles. These lists of articles later are used to provide features for our relation classifier.

Figure 1 shows the pseudo codes of our concept disambiguation algorithm. In this algorithm, function $ESA(query)$ retrieves the most semantically relevant articles in Wikipedia to *query*. All categories of a *list* of articles are extracted by function $getCategories(list)$. Function $search(query)$ retrieves relevant articles in Wikipedia by searching *query* in the articles' titles and texts.

```
CONCEPT DISAMBIGUATION ALGORITHM
    INPUT: Two concepts X and Y
    OUTPUT: A list of relevant articles in Wikipedia.
            for each concept.

    Query q ← Concatenating X and Y;
    L = ESA(q); // return a list of relevant articles to X and Y

    C = getCategories(L);
    T = tokenize(C);
    T_top = top^K_tfidf(T); // pick top K tokens

    Query q_x ← Concatenating x and tokens from T_top;
    Query q_y ← Concatenating y and tokens from T_top;
    A_x = search(q_x);
    A_y = search(q_y);

    RETURN: A_x and A_y;
```

Figure 1: Pseudo code of our concept disambiguation algorithm. The algorithm takes as input two concepts. The algorithm disambiguates two given concepts and return a list of relevant articles in Wikipedia for each concept.

3.2 Learning Concept Relations

Our goal is to design a supervised learning approach that identifies relational knowledge between concepts. We want our learning algorithm to be as general as possible so that by training on examples having concepts in some semantic classes, we are still able to classify examples in other semantic classes with high accuracy. To do this, we first select expressive features that will be associated with given concept pairs. These features are necessarily independent of the semantic class of the given concepts. Recall that, after the concept disambiguation step, each concept of a given pair is associated with a list of relevant articles in Wikipedia. These articles give lists of titles, texts, and categories for the corresponding concepts. From now on, we use *the titles of concept* X to refer to the titles of the articles associated with concept X; similarly for *the texts of concept* X, and *the categories of concept* X. As learning algorithm, we use a regularized averaged Perceptron [7]. The features selected for our learning problem include the word usage of the associated articles, the association information between two concepts, and the overlap ratios of important information between the concepts and the articles. We describe these features below.

3.2.1 Word Usage

One of the most important features in recognizing relations between concepts is the word usage in associated articles. An article in Wikipedia typically contains three main components: title, text, and categories. The title distinguishes a concept from other concepts in Wikipedia. The text describes the concept. The categories classify the concept into one or more concept groups which can be further categorized. To collect the categories for a concept, we take the categories of its associated articles go up to K level in the Wikipedia category system. In our experiments, we use abstracts of Wikipedia articles instead of whole texts.

By analyzing Wikipedia articles, one could observe that the word usage of the texts describing a concept often overlaps with the word usage of the categories of its inferior concepts. For example, the texts describing the concept

Presidents of the United States overlap with the categories of *George W. Bush*, and *Gerald Ford* in the tokens *president, united, states* and so on (see Table 1.)

On the other hand, two sibling concepts often have overlap not only in their texts, but also in their categories. For example, concepts *George W. Bush* and *Gerald Ford* both use *presidents, united, states* in their texts, as well as *presidents, united, states, republican* in their categories.

We define four word usage features associated with any two given concepts X and Y: the degree of similarity in word usage between the texts of concept X and the categories of concept Y, the similarity between the categories of X and the texts of Y, the similarity between the text of X and the text of Y, and the similarity in word usage between the categories of X and the categories of Y.

We now have to measure the degree of similarity in word usage for the features. There are several ways to calculate degree of similarity between two texts [13]. In this paper, we use the cosine similarity to measure the degree of similarity of word usage of two text fragments. Equation (1) shows the formula of the cosine similarity metric applying on the term vectors T_1 and T_2 of two text fragments.

$$Sim(T_1, T_2) = cos\theta(T_1, T_2) = \frac{T_1 \cdot T_2}{\|T_1\| \, \|T_2\|} = \frac{\sum_{i=1}^{N} x_i y_i}{\sqrt{\sum_{j=1}^{N} x_j} \sqrt{\sum_{k=1}^{N} y_k}} \quad (1)$$

where N is the vocabulary size, and x_i, y_i are two indicator values in T_1 and T_2, respectively.

3.2.2 Association Information

Another useful feature that can help to recognize the relation between two concepts is the association information between them. We capture the association information between two concepts X and Y by measuring the pointwise mutual information, defined in equation (2). In this paper, the association information is measured in document level.

$$PMI(X, Y) = log \frac{p(X, Y)}{p(X)p(Y)} = log \frac{\frac{f(X,Y)}{N}}{\frac{f(X)}{N} \frac{f(Y)}{N}} = log \frac{N f(X, Y)}{f(X) f(Y)} \quad (2)$$

where X and Y are two input entities, N is the total number of document in Wikipedia, and $f(.)$ is a function returning document frequency.

3.2.3 Overlap Ratio

Wikipedia articles are not only good in giving definition and explanation about concepts, they are also very helpful in categorizing concepts into groups and topics. We want to capture the fact that the titles of a concept often has an overlap with the categories of its descendants. For examples, the title (and also the concept itself) *Presidents of the United States* overlap with one of the categories of the concepts *George W. Bush* and *Gerald Ford* (see Table 1.) This phenomenon is a good feature to recognize the ancestor relation. We measure this overlap as the ratio of *common phrases* used in the titles of concept X and the categories of concept Y. In our context, a phrase is considered as a *common phrase* if it appears in the titles of X and the categories of Y and it is one of the following things:

- the *whole string* of a category of Y, or
- the *head* in its root form of a category of Y, or
- the *post-modifier* of a category of Y.

In our work, we use the Noun Group Parser from [23] to extract the *head* and *post-modifer* from a category. For example, one of the categories of an article about *Chicago* is *Cities in Illinois*. This category can be parsed into a head in its root form *City*, and a post-modifer *Illinois*. Given two entity pairs *(Illinois, Chicago)* and *(City, Chicago)*, we can recognize that *Illinois* and *City* match the category *Cities in Illinois* of concept *Chicago*. This is a good indication that *Chicago* is a descendant of both *Illinois* and *City*.

To collect the categories for a concept, we collect the categories of its associated articles within K level up in the Wikipedia category system. We measure the overlap ratio between the titles of concept X and the categories of concept Y by using the Jaccard similarity coefficient as shown in equation (3).

$$\sigma^K_{ttl}(X, Y) = \frac{|L_X \cap \mathcal{P}^K_Y|}{|L_X \cup \mathcal{P}^K_Y|} \quad (3)$$

where L_x is the set of the titles of X, and $\mathcal{P}_y$ is the union of the category strings, the heads of the categories, and the post-modifiers of the categories of Y.

Similarly, we also use the ratio $\sigma^K_{ttl}(Y, X)$ as one of our features.

We also observe that there are many cases where two concepts are siblings or have no relation but they still have an overlap between their titles and their categories. To handle this, we add one more feature which is the overlap ratio of *common phrases* between the categories of two given entities. However, to capture the sibling relation, we do not use the *post-modifier* of the categories, because *post-modifier* does not help us to recognize sibling relation (e.g. *Cities in Illinois* and *Mountains In Illinois* have the same *post-modifier*, which is *Illinois*, but a city and a mountain cannot be sibling.)

The Jaccard similarity coefficient for overlap ratio between the categories of two concepts X and Y is showed in equation (4).

$$\sigma^K_{cat}(X, Y) = \frac{|\mathcal{Q}^K_X \cap \mathcal{Q}^K_Y|}{|\mathcal{Q}^K_X \cup \mathcal{Q}^K_Y|} \quad (4)$$

where $\mathcal{Q}^K_{(.)}$ is the union of the category strings and the heads of the categories. The categories are collected by going up to K level in the Wikipedia category system.

All of the features described above are used in training our multi-class classifier to recognize relation between concepts.

3.3 Improving System Coverage

In our work, Wikipedia is the main source of background knowledge used to recognize concept relations. Although most commonly used concepts are mentioned in Wikipedia as we have argued above, there is still a need to identify relations between other concepts that do not have relevant articles in Wikipedia. We call these concepts *non-Wikipedia*

Concept	Text	Categories
President of the United States	*The President of the United States is the head of state and head of government of the United States and is the highest political official in the United States by influence and recognition. The President leads the executive branch of the federal government and is one of only two elected members of the executive branch...*	*Presidents of the United States, Presidency of the United States*
George W. Bush	*George Walker Bush; born July 6, 1946) served as the 43rd President of the United States from 2001 to 2009. He was the 46th Governor of Texas from 1995 to 2000 before being sworn in as President on January 20, 2001...*	*Children of Presidents of the United States, Governors of Texas, Presidents of the United States, Texas Republicans...*
Gerald Ford	*Gerald Rudolff Ford (born Leslie Lynch King, Jr.) (July 14, 1913 December 26, 2006) was the 38th President of the United States, serving from 1974 to 1977, and the 40th Vice President of the United States serving from 1973 to 1974.*	*Presidents of the United States, Vice Presidents of the United States, Republican Party (United States) presidential nominees...*

Table 1: Examples of texts and categories of the concepts from Wikipedia: *President of the United States*, *George W. Bush* and *Gerald Ford*.

concepts. In this section, we show how to improve the coverage of our approach to concepts outside Wikipedia. The idea is that we find, for each such concept, a replacement that is in Wikipedia. Our method was motivated by [19]. In their work, they address the set expansion problem. In order to identify concept pairs that belong to lists, they look for structures like *"... ne_a, ne_b and ne_c ..."* where ne_a, ne_b, etc. are named entities. E.g. *"I've lived in NY, Paris, and Amsterdam."* . Another possibility would be *"... ne_a, ne_b or ne_c ..."* . However, in their work, the authors use the texts from Wikipedia to look for entities in the desired structures. We are trying to cover concepts beyond Wikipedia. Therefore, we use a modified version of their structures and use Web search engines to search for concepts in textual lists. In our work, we use Yahoo! Web Search API[2]. We use Web search enignes to search for the conjunction of two given concepts (e.g. *"bass"* AND *"trout"* and look for the list structures *"... ¡DELIMITER¿ c_a ¡DELIMITER¿ c_b ¡DELIMITER¿ c_c ¡DELIMITER¿ ..."* in the top snippets returned. The delimiters used in our experiments are comma(,), puntuation(.), and asterisk(*). For sentences that contain our structures, we extract c_a, c_b, etc. as candidate concepts which are in the same semantic class with our original given concepts. To reduce noise from concepts extracted from the snippets, we constrain the list structure to contain at least 4 concepts and each concept cannot be longer than 20 characters. Once a list of concept candidates was extracted, the concepts are ranked based on their frequency of occurrence. The highest ranked concepts are used to replace the non-Wikipedia concepts in the original given pair. The highest ranked candidates are tried with the concept disambiguation algorithm until we find the concepts mentioned in Wikipedia.

Note that unlike other pattern-based methods discussed in sec. 2 that search for concepts in close proximity, we use concepts in Wikipedia as anchors for the search, thus increasing the likelihood of covering less commonly used concepts.

4. INFERENCE WITH RELATIONAL CONSTRAINTS

In this session, we describe a further approach that helps our system achieve a better performance on relation identification. From our real case observation, we notice that there are some constraints that can be used to enforce the relation between two input concepts identified by the system. For instance, concept *George W. Bush* cannot be an ancestor or sibling of concept *president* if we know that concept *president* is an ancestor of concept *Bill Clinton*, and *Bill Clinton* is a sibling of concept *George W. Bush* (see Fig. 2.) Another example would be that concepts *red* and *green* are known to be siblings, and concept *blue* is also known as a sibling of *red*, the prediction identifying that *green* is an ancestor of *blue* should be invalid. If, by some way, we can obtain related concepts of the two input concepts, we can enforce such constraints as prior knowledge to do inference that guides the final decision of the relation identifier. In literature, there are several models that take advantage of prior knowledge to achieve significant improvement in performance (CITATIONS XXX YYY). There are two main approaches to inject prior knowledge. The first approach incorporates prior knowledge *indrectly*; by adding more features (CITATIONS XXX YYY). In the other hand, the other approach incorporate prior knowledge directly under the form of hard or soft constraints (CITATIONS XXX YYY). In our work, we want our model to incorporate prior knowledge directly, therefore our model is closely related to the latter approach. We first present our inference model that incorporate prior knowledge. After that, we propose our approach that allow us to obtain related concepts of two input concepts.

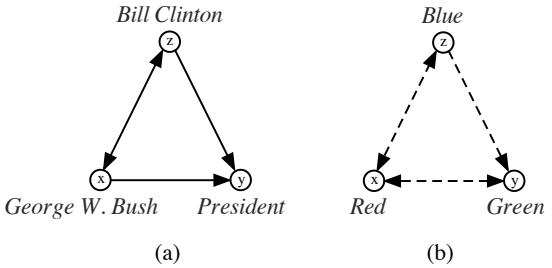

Figure 2: Examples of triangles formed by two input concepts x, y, and a related concept z. The relation between concepts is predicted by a local classifier (see Sec. 3.2). The left triangle (a) shows a valid combination of three edges. The right triangle (b) demonstrate an invalid combination of the edges. The right triangle forms a relational constraint that we do not allow to happen.

[2]http://developer.yahoo.com/search/web/

4.1 Constraints as Prior Knowledge

Let two input concepts (x, y), and their related concepts $\mathcal{Z} = \{z_1, z_2, ..., z_m\}$. For each related concept z_i, a set of triangles is constructed with a triple $\langle x, y, z_i \rangle$, and edges between every two concepts. Our local classifier (see Sec. 3.2) is used to predict weights for four possible classes of an edge. In our problem, each edge have four classes. For reading convenience, we name the relations from x to y, from x to z_i, and from y to z_i as the first, second, and third edge, respectively. There are 64 possible triangle constructed from $\langle x, y, z_i \rangle$ in total. Figure 2 illustrates some triangles formed with two input concept x and y, and an related concept z_i. We use e and $w(e)$ to denote an edge in a triangle and its corresponding weight, respectively.

Our relational constraints are defined as a set $\mathcal{C}$ of invalid triangles with particular combinations of three edges. Figure 2(b) shows a constraint where the combination from the 1st to the 3rd edge is $\langle sibling, sibling, ancestor \rangle$.

Following the approaches to inject the prior knowledge directly in (CITATIONS XXX YYY), we incorporate the relational constraints into our model to choose the best triangle out of 64 possible triangles in a set of those constructed from $\langle x, y, z_i \rangle$. The scoring function is a linear combination of the edge weights $w(e)$ and the penalties ρ_k of triangles violating constraint $C_k \in \mathcal{C}$.

$$\hat{t} = \text{argmax}_t \sum_{e \in t} w(e) - \sum_{k=1}^{|\mathcal{C}|} \rho_k d_{C_k}(t) \quad (5)$$

In Eq. 5, function $d_{C_k}(t)$ measures the degree that triangle t violates constraint C_k. In our work, relational constraints are mined from the training data (see Sec. 4.2). The selected constraints are considered to have high confident about the knowledge. We, thus, use them as hard constraints, and set their penalty ρ_k to ∞ (CITATIONS XXX YYY).

The objective function for triangles allows us to pick the most possible setting for all edges connecting triple $\langle x, y, z_i \rangle$ with respect to relational constraints.

So far, we have presented our model to pick the topmost triangle for a particular related concept z_i. For each z_i, the topmost triangle also presents to us the most likely relation between two input concept x and y. We repeat the process of finding the topmost triangle for all extracted related concept in $\mathcal{Z}$. After going through all $z_i \in \mathcal{Z}$, each relation ℓ has a pool of topmost triangles $\mathcal{T}_\ell$ in which each triangle contains the first edge with relation ℓ. To make the final decision on the relation of two input concept (x, y), we solve the objective function defined in Eq. 6.

$$\begin{aligned} \hat{\ell} &= \text{argmax}_\ell \frac{1}{|\mathcal{T}_\ell|} \sum_{t \in \mathcal{T}_\ell} \lambda_t score(t) \quad (6) \\ &= \text{argmax}_\ell \frac{1}{|\mathcal{T}_\ell|} \sum_{t \in \mathcal{T}_\ell} \lambda_t \left(\sum_{e \in t} w(e) - \sum_{k=1}^{|\mathcal{C}|} \rho_k d_{C_k}(t) \right) \end{aligned}$$

where

$$\lambda_t = \frac{\text{\# of occurrence of } t}{\text{\# of predicted triangles in training data}} \quad (7)$$

is the occurrence probability of triangle t in the training data with related concepts.

4.2 Constraint Selection

To mine relational constraints from the training data with respect to extracted related concepts, we apply the *forward feature selection* technique (CITATIONS XXX YYY). The algorithm starts with an empty set of constraints, then grow the set gradually by adding to the set the *best* triangle in each iteration. The *best* triangle is defined as the constraint that help improve system performance most. Our *forward constraint selection* process is described in details in Alg. 3.

```
Algorithm FORWARD CONSTRAINT SELECTION
    INPUT: Data set with related concepts D = ⟨(x, y, ℓ), Z⟩
        A trained local classifier L.
    OUTPUT: Constraint set C.

    C = ∅;
    bestAcc = evaluate(D, L, C);
    T = {t_1, t_2, ..., t_64};
    isIncreasing = true;

    While (isIncreasing) do
        isIncreasing = false;
        For each t ∈ T do
            C = C ∪ {t};
            acc = evaluates(D, L, C);
            If (acc > bestAcc) then
                t̂ = t;
                bestAcc = acc;
                isIncreasing = true;
            End if
            C = C\{t};
        End for
        If (isIncreasing) then
            C = C ∪ {t̂};
            T = T\{t̂};
        End if
    End while

    RETURN: C;
```

Figure 3: Forward constraint selection algorithm. The *evaluate* function evaluates all examples from the input data set with related concepts using a trained local classifier $\mathcal{L}$ and constraint set $\mathcal{C}$ by solving the objective function 6 .

4.3 Extract Related Concepts

1. Gold related concepts.
2. Yago related concepts.

4.4 Discussion on Constraints

- polysemous concepts.

5. EXPERIMENTAL STUDY

5.1 Datasets

Our approach to identifying relational knowledge is evaluated by using a dataset of 40 semantic classes of almost 11,000 instances, which was used in [14]. This dataset was manually constructed[3] and was used to evaluate many information extraction tasks [14, 15]. Each semantic class is an incomplete set of representative instances, and has about 272 instances in average. The smallest class is *search engine*

[3]Private communication with Marius Pasca, 2009.

with 25 instances, and the largest class is *actor* with 1500 instances. Table 2 shows a snippet of the dataset. Interestingly, we have both types of closed word semantic class (e.g. *chemical element*, *country*), and open word semantic class (e.g. *basic food*, *hurricane*). Moreover, there are classes with proper nouns (e.g. *actor* with *Mel Gibson*, *Sharon Stone*), and also classes with common nouns (e.g. *basic food* with *rice*, *milk*, *eggs*). This fact helps evaluate the ability of systems that can deal with not only concepts like name entities, but also common instances.

Semantic class (Size)	Examples of Instances
basic food (155)	rice, milk, eggs, beans, fish
chemical element (118)	lead, copper, aluminum, calcium
city (589)	San Francisco, Dubai, Chicago
disease (209)	arthritis, hypertension, influenza
actor (1500)	Kate Hudson, Mel Gibson

Table 2: A snippet of 40 semantic classes with instances. The class names in the original dataset (*basicfood*, *chemicalelem*) were presented in a meaningful form as shown in the left column.

In the original dataset, the semantic class names are not often written in a meaningful form, such as *chemicalelem*, *proglanguage*, and *worldwarbattle*. These names are not usable for our system to find corresponding concepts in Wikipedia. In our experiments, each semantic class name in the original dataset is expanded to meaningful forms. For example, *terroristgroup* is expanded to *terrorist group*, *terrorrist*, *torrorism*. The expansion is kept to be minimum. Moreover, we also use these expansions for other systems to compare to our approach in the experiments. We refer to these expansions as *class expansion*.

An example in our learning problem is a pair of two concepts X and Y such as (*city*, *Dubai*), (*lead*, *aluminum*). Note that in this paper, we refer to both the name of semantic classes and their instances as concepts. We pair the semantic classes and instances in the original dataset to create learning examples used in training and evaluating our approach. The examples cover all types of relational knowledge of interest: $X \leftarrow Y$, $X \rightarrow Y$, $X \leftrightarrow Y$, and $X \nleftrightarrow Y$. We create learning examples with the following guidelines.

- $X \leftarrow Y$ examples: For each semantic class, we pair the name of the class with its instances. Because the order of the class name and the instance in a pair defines the direction of the relation, we make the class name and the instance as the first and the second concept in a pair, respectively.
- $X \rightarrow Y$ examples: Similar to the previous guideline, for each semantic class, the class name and its instances are paired. The class name and the instance are the first and the second concept in a pair, respectively.
- $X \leftrightarrow Y$ examples: For each semantic class, instances in the class are paired with each other. Order is not a matter in this case.
- $X \nleftrightarrow Y$ examples: To make examples with two concepts having no relation, we pair either a class name and an instance of another class (and vice versa), or an instance in a semantic class and another instance in other classes.

Table 3 shows some examples in the our dataset.

Relation	Concept X	Concept Y
$X \leftarrow Y$	actor	Mel Gibson
	food	rice
	wine	Champagne
$X \rightarrow Y$	Makalu	mountain
	Monopoly	game
	krooni	currency
$X \leftrightarrow Y$	Paris	London
	copper	oxygen
	Nile	Volga
$X \nleftrightarrow Y$	Roja	C++
	egg	Vega
	HotBot	autism

Table 3: Some examples of relations in our dataset.

We form our learning dataset by randomly generating 20,000 examples following the guidelines above. We divide these examples into two sets of 8,000 and 12,000 examples for train and test sets, respectively.

For the train set of 8,000 examples, we discard any examples which contain one or both concepts that our concept disambiguation algorithm retrieves no relevant articles in Wikipedia. This results in a train set of 6,959 examples. Similarly, for the test set, if we discard examples with one or both concepts that we cannot find relevant articles in Wikipedia, we have 10,456 out of 12,000 examples remain. We call this test set *TestWiki*. The test set with all 12,000 examples is called *TestAll*. Table 4 presents the details of our train and test sets with the number of examples in four relation types. In our experiments below, we use the *TestWiki* test set as default, except explicitly specified.

Data	$X \leftarrow Y$	$X \rightarrow Y$	$X \leftrightarrow Y$	$X \nleftrightarrow Y$	Total
Train	1,739	1,754	1,664	1,802	6,959
TestWiki	2,684	2,654	2,483	2,635	10,456
TestAll	3,045	3,025	2,965	2,965	12,000

Table 4: Details of the training and test sets in our learning task with the numbers of examples in four relation types. *TestWiki* is the test set with examples having concepts in Wikipedia. The concepts of examples in *TestAll* are not necessarily in Wikipedia. In our experiments, we use the *TestWiki* test set as default, except explicitly specifying.

To evaluate our system, we use a snapshot of Wikipedia in July, 2008. We first clean up articles in Wikipedia and remove articles that are not of interest. The removed articles include articles without a category, except the redirect pages, or articles with useless categories such as *Protected redirects*. We also remove administrative articles including *Wikipedia* pages, *Template* pages, *Image* pages, and *Portal* pages. Furthermore, we do not use articles without titles. After the pre-processing step, we have 5,503,763 articles left. These articles will be our source to query relevant Wikipedia pages to input concepts. We index the articles using the Apache Lucene Information Retrieval library[4]. Lucene is a

[4]http://lucene.apache.org, version 2.3.2

high-performance text search library that is written in Java and is a widely used off-the-shelf IR system.

5.2 Experimental Results

In this section, we describe our experiments and evaluate the performance of our approach in identifying relational knowledge. For each experiment, we first present the experimental setup and then show experimental results.

We first compare the performance of our approach with other systems that use existing resources to identify concept relation.

The first system uses a taxonomy which was derived from Wikipedia [16], as the background knowledge. The taxonomy was created by applying several lexical matching and methods based on connectivity in the network to the category system in Wikipedia. As a result, the taxonomy contains a large amount of subsumption, i.e. *isa*, relations [16]. We use the taxonomy to recognize relations between given concepts when they are in the taxonomy. Given an example with two conceptx X and Y, using the taxonomy, X is an ancestor of Y if one of the articles about Y is subsumed by an article about Y. We implement this idea by looking at the *isa* links of articles of Y up to K levels in the taxonomy. Similarly, for the case that Y is an ancestor of X. If X and Y share a common ancestor within K levels in the taxonomy, they are considered as siblings. We first apply our concept disambiguation algorithm on given concepts and then mount them onto the taxonomy to infer the relations. The taxonomy used in this experiment is in the latest version of March, 2008[5]. We refer to this system as *Strube 07* in our experiment.

The second system that we compare to in this experiment uses the *extended WordNet* [21, 22] as the background knowledge. The authors of the extended WordNet [21] first identified lexico-syntactic patterns indicative of hypernymy from corpora. These patterns were used to extract candidate noun pairs that may hold the hypernym relation. A trained classifier is applied on these noun pairs to recognize the pairs holding hypernym relation. Starting from WordNet-2.1 [6], the latest version of the extended WordNet has augmented 400,000 synsets. Words that are added into the extended WordNet can be common nouns or proper nouns. The extended WordNet can serve as the very good background knowledge to identify relational knowledge of interest by looking for input concepts in the extended WordNet taxonomy for all possible senses of the concepts. Given a pair of two concepts (X, Y), X is an ancestor of Y if X subsumes Y within K levels up from Y in the taxonomy. Similarly for the case that Y is an ancestor of X. If there exists a common subsumption for both X and Y within K level up from both concepts in the taxonomy, X and Y are sibling. We refer to this system as *Snow 06* in our experiment.

For our system, we also vary the value of K as the number of levels that we go up in the category system of Wikipedia to calculate features (see section 3) for any pair of concepts. In this experiment, we vary K from 0 to 4. With $K = 0$, the systems use no information from the hierarchical structure of the taxonomy. We train our classifier on the train set and evaluate all systems on the test sets described in table 4. We evaluate performance of the systems by calculating the accuracy in recognizing relation between concepts in pairs. The accuracy is computed by the percentage of the number of correct prediction over the total number of examples used in testing. The experimental results with the *TestWiki* test set are showed in table 5. From the results, our approach significantly outperforms other systems that use existing resources as background knowledge to identify relations between concepts. Our system reaches the best performance with $K = 3$ and achieves 88.79% in accuracy.

[5]Private communication with Michael Strube and Simone Paolo Ponzetto, 2009.

	$K=0$	$K=1$	$K=2$	$K=3$	$K=4$
Strube 07	23.98	24.02	24.04	24.08	24.59
Snow 06	25.2	41.78	44.34	42.23	41.58
Ours	**27.22**	**85.19**	**87.95**	**88.79**	**87.55**

Table 5: Performance in accuracy of our approach compared to other systems using existing resources as background knowledge. The *TestWiki* test set is used in this experiment.

	$K=0$	$K=1$	$K=2$	$K=3$	$K=4$
Strube 07	23.78	23.83	23.84	23.88	24.32
Snow 06	24.71	40.24	42.63	40.96	40.65
Ours	**27.97**	**77.99**	**81.21**	**81.54**	**79.23**

Table 6: Performance in accuracy of our approach compared to other systems with the *TestAll* test set.

Table 6 presents performance of the system on the *TestAll* test set. It is clear that including examples with *non-Wikipedia* concepts and having no special approach to deal with them will hurt the performance of the classifier. This is the place where our method to cover concepts beyond Wikipedia comes into. We will present the experiment on this later.

For now, we compare our system with *Snow 06* only on examples having concepts in the *extended WordNet*. To do this, we carry out the experiment where we discard in our test set the examples which have one or both concepts that are neither in Wikipedia nor the *extended WordNet*. This results in a test set with 8,625 examples. Table 7 shows the experimental results. From the table, we see that *Snow 06* now performs better than in the previous experiment for all values of K. However, our approach still significantly outperforms *Snow 06*.

	$K=0$	$K=1$	$K=2$	$K=3$	$K=4$
Snow 06	24.59	44.68	47.79	45.24	44.45
Ours	**27.92**	**87.35**	**86.18**	**88.65**	**73.74**

Table 7: Comparing the performance of our system with *Snow 06* only on examples having concepts in the latest version of Snow's *extended WordNet*.

Improving system coverage: We evaluate our approach to improve our system coverage described in section 3.3. In this experiment, we use the top 50 snippets returned by the Web search engine. Table 8 shows the error reduction of our new system *+BeyondWiki* covering concepts not mentioned in Wikipedia. The results show that our approach to covering *non-Wikipedia* concepts is very effective.

K	OrgApproach	+BeyondWiki	**Error Reduction**
0	27.97	37.99	**13.91**
1	77.99	80.62	**11.95**
2	81.21	83.58	**12.61**
3	81.54	83.14	**8.67**
4	79.23	82.0	**13.34**

Table 8: Extending the coverage of our original approach. We add the *BeyondWiki* extension to our original system *OrgApproach* to recognize the relation between concepts outside Wikipedia. The result are on the *TestAll* dataset. K is the maximum number of level to go up in the Wikipedia category system.

5.3 Experimental Analysis

5.3.1 Concept Disambiguation

To exhibit the performance of our concept disambiguation algorithm, we choose 3 semantic classes, representing varying level of ambiguty. These three classes are *England Football Clubs*, *Superheroes*, and *Rivers in England*. We choose these three classes because they have many ambiguous concepts. For example, the concept *Chelsea* may refer to places such as a railway station in London (*Chelsea tube station*), a city in Massachusetts in the United States (*Chelsea, Massachusetts*), or a city in Canada (*Chelsea, Quebec*); *Chelsea* also refers to people such as Miss USA 2005 *Chelsea Cooley*, or actress *Chelsea Noble*; furthermore, *Chelsea* can refer to sport organization as in *Chelsea Football Club*. In short, football clubs are often named after the name of the city where it resides, superheroes are named with random names which can match with many other things such as *Thunderbird*, *Tiny*, and *Speedy*; similarly, rivers are named with ambiguous names such as *Burn* and *Hun*.

For the *England Football Clubs* class, we create 40 pairs in the form (*football*, ¡name of a football club¿), and also 40 pairs in the form (¡name of a football club¿, ¡name of another football club¿). Similarly, with the *Superheroes* class, we generate 40 pairs in the form (*superhero*, ¡name of a superhero¿), and 40 pairs of (¡name of a superhero¿, ¡name of another superhero¿). For the *River class*, we have only 34 instances used in [25], we use all of them to make pairs of (*river*, ¡name of a river¿) and (¡name of a river¿, ¡name of another river¿). In total, there are 80, 80, and 68 pairs for the *Football Clubs*, *Superheroes*, and *Rivers*, respectively.

We compare our algorithm with the search process performed on two concepts in a pair separately, which is called *Separate search*. The *Separate search* method only search titles of articles in Wikipedia.

We present our experimental results of concept pair disambiguation in table 9. We evaluate the performances by manually examining the top 5 (*Top5*) and top 10 (*Top10*) Wikipedia articles retrieved by the search process. Without loss of generality, we only evaluate the accuracy of the second concept in the pairs of $X \leftrightarrow Y$. For the pairs of $X \leftarrow Y$, such as (*superhero*, ¡name of a superhero¿), we also only evaluate the accuracy of the second concept because the first concept is not ambiguous (*superhero* in this case). Our algorithm outperforms the *Separate search* method about 10% with top 5 and 6% with top 10 in average accuracy. It worth noting from the the result table that our algorithm produces no different result for *Top5* and *Top10* lists. This shows that our algorithm is effective in retrieving relevant article to the input concepts by putting relevant articles to very top in the relevant list if the input concepts can be found in Wikipedia. The result of *River* is quite low with our approach, where we get about 78% in accuracy for both *Top5* and *Top10*. This can be explained by observing that two river names are not often appear in the same article about a certain river. Therefore, our algorithm when searches for articles which contain two rivers in focus may return unexpected articles. This expalin why in *Top10* of *River*, we lose about 1.5% of accuracy compared to the *Separate search* method. For *Superheroes*, our algorithm significantly outperforms the *Separate search* method with 81.25% in accuracy. A careful observation shows that there are 8 out of 15 pairs, which we retrieve irrelevant articles, containing at least one *superhero* which is not in Wikipedia at all. Some of these concepts include *renegade*, *big red*, and *majestic agent*.

5.3.2 Relation Classification Analysis

In the next experiment, we analyze the performance of our approach on individual semantic class with $K = 3$. We use the *TestWiki* test set as default, except explicitly specified.

Performance on Individual Semantic Class: We first analyze our classifier by looking at the performance on individual semantic class. This experiment shows us classes which may cause problems to our classifier. Moreover, this experiment also shows us the performance of our classifier on individual relation type including $X \leftarrow Y$, $X \rightarrow Y$, $X \leftrightarrow Y$, and $X \nleftrightarrow Y$. There are 40 semantic classes in the datasets. Recall that examples in our datasets are pairs of concepts. To get the total number of pairs of a semantic class, we count the number of examples having one of the two concepts belongs to that class. If two concepts of a pair hold a sibling relation (i.e. they are in the same semantic class), the pair is counted only one time for the class. Again, the performance of the classifier is measured in accuracy which is the portion of correct prediction over the total number of examples. Table 10 presents the performance of our classifier on individual semantic class, and also the average accuracy on each relation type.

From the last row in the table, the average accuracies of the classifier on relations $X \leftarrow Y$ and $X \rightarrow Y$ are roughly the same. This essentially occurs because these two learning classes are symmetric. The performance of the classifier on sibling relation is lower than other relations. This can be explained by the fact that disambiguating two sibling concepts is not a trivial problem. As we discussed in the experiment evaluating the concept disambiguation algorithm above, two sibling concepts such as two rivers are not often mentioned in the same article. This fact may cause the algorithm to retrieve unexpected articles that associate with two input concepts. This leads to making wrong classification decision. The ancestor relation, in both directions, gets the best performance. One of the reason for this result is that disambiguating two concept holding true ancestor relation is often easy because the ancestor concept automatically disambiguate the descendant concept in the pair.

Internal and External evaluation: In section 3, we argue that our features do not depend on the semantic class. We want our classifier to be trained on some semantic classes

Method	Football Clubs		Superheroes		Rivers	
	Top5	Top 10	Top5	Top10	Top5	Top10
Separate search	95.0	96.25	60.0	65.0	75.0	79.41
Our algorithm	100	100	81.25	81.25	77.94	77.94

Table 9: The accuracy of concept disambiguation algorithm. *Top5* and *Top10* corresponds to examining top 5 and 10 articles retrieved. The *Separate search* method refers to the search performed on the two concepts in the example separately.

Semantic class	$X \leftarrow Y$	$X \rightarrow Y$	$X \leftrightarrow Y$	$X \nleftrightarrow Y$	Average
hurricane	100	100	100	97.20	**99.30**
cellphonemodel	100	100	100	96.85	**99.21**
carmodel	100	100	100	94.12	**98.53**
searchengine	100	100	100	93.18	**98.30**
nationalpark	100	100	100	90.51	**97.63**
aircraftmodel	100	100	95.65	94.32	**97.49**
nbateam	100	100	100	87.88	**96.97**
videogame	98.43	99.31	100	87.59	**96.33**
skyscraper	100	100	93.85	90.55	**96.10**
digitalcamera	91.89	94.87	100	92.96	**94.93**
stadium	100	100	93.59	85.06	**94.66**
soccerclub	100	100	97.14	80.71	**94.46**
chemicalelem	100	100	90.48	86.11	**94.13**
cartooncharacter	100	100	95.83	80.30	**94.03**
worldwarbattle	100	100	85.07	88.71	**93.45**
treaty	93.55	94.55	89.47	92.00	**92.39**
mountain	100	98.51	83.87	86.52	**92.23**
university	100	100	83.87	82.96	**91.71**
painter	97.13	96.88	77.94	93.89	**91.46**
holiday	95.45	100	81.63	82.84	**89.98**
newspaper	92.13	96.88	73.44	96.64	**89.77**
actor	92.74	92.24	89.47	83.33	**89.45**
skybody	93.75	93.10	76.00	93.84	**89.17**
wine	100	86.67	80.77	86.44	**88.47**
terroristgroup	100	100	68.18	83.33	**87.88**
disease	85.00	87.67	81.43	95.04	**87.29**
currency	100	90.00	70.00	89.06	**87.27**
proglanguage	96.55	86.67	73.44	90.97	**86.91**
movie	90.23	87.06	93.62	75.91	**86.71**
country	98.25	98.36	91.57	57.75	**86.48**
sportevent	100	100	53.19	84.00	**84.30**
religion	93.88	88.57	65.88	86.40	**83.68**
award	96.00	98.00	41.03	92.17	**81.80**
city	97.08	98.76	62.63	61.43	**79.98**
flower	89.47	85.00	60.71	84.13	**79.83**
basicfood	81.82	91.11	58.89	86.11	**79.48**
company	93.05	94.74	55.56	74.45	**79.45**
river	89.47	97.50	41.05	81.29	**77.33**
drug	80.00	81.91	53.85	93.29	**77.26**
empire	83.33	75.00	56.00	83.21	**74.39**
Average	**95.73**	**95.33**	**80.38**	**86.58**	

Table 10: Performance of our system on individual semantic classes. The semantic classes are presented in a decreasing order of the last column, showing the average accuracy on all relations of the corresponding semantic class. The last row shows the average accuracy of each relation type evaluated on all semantic classes. This experiment is done with $K = 3$.

and then can be used to identify relations between concepts in other semantic classes. For example, we do not have examples in the semantic class *fish* in our training data, but we want to get high accuracy on examples in the class *fish* with our trained classifier. Our features are designed with this as motivation. In this experiment, we exhibit that our system can indeed generalize across semantic classes. We present the performance of our system when trained on some semantic classes and tested on other classes. We do this experiment using 5-fold cross validation. For each fold in this experiment, we randomly divide the 40 semantics classes into two sets. One set gets examples from 30 semantic classes. All examples in this set are randomly divided into two sub-sets which are used as the training set and the *Internal* test set. The other set get examples from the other 10 semantic classes and is used as the *External* test set. The results of the 5-fold cross validation process are showed in table 11. We report the average accuracy of our classifier, which is trained on the training set in this experiment, applied on both *Internal* and *External* test sets. We learn from the table that our system performs no worse on the external test set. It is even slightly better than the performance of the system when applied on the internal test set. It worth noting that the performance of our system depends on two important factors: the performance of the concept disambiguation algorithm and the performance of the relation classifier. If the classifier is trained on examples from hard semantic classes, such as the classes at the bottom of table 10, it may still perform poorly on the *Internal* test set compared to when it applies on examples in "easy" semantic classes, such as classes at the top of table 10. This explains why sometime we may get slightly better results on the *External* test set. The experimental results show that our system generalizes well to new semantic classes.

5-fold cross validation	
Internal Testset	External Testset
88.77	89.87

Table 11: Genearlization to new classes: Performance of our system on Internal and External test sets using 5-fold cross validation. The Internal test set is formed by examples having concepts in the same semantic classes as those in the train set. The External test set is formed by examples having concepts in different semantic classes than those of the train set. As shown in this table, this has no significant impact on the results. This experiment is done with $K = 3$.

Different sizes of train set: In this experiment, we present the performance of our system when the classifier is trained with different number of examples in the train set. The learning curve of the average performance is shown in figure 4. The results show that our system achieves over 80% of accuracy by using only 100 examples to train the classifier. Furthermore, the average performance converges to about and over 88% starting with 1,000 examples used in training. The results illustrate the effectiveness of our system on the problem of relational knowledge identification.

6. CONCLUSION

Textual inference problems that often arise in data and knowledge management problems necessitate some level of "common sense" inference that relies of identifying relations such as *ancestor* and *sibling* among concepts.

We have developed a robust and accurate machine learning approach to this problem, that is based using a very small number of annotated examples. The results presented

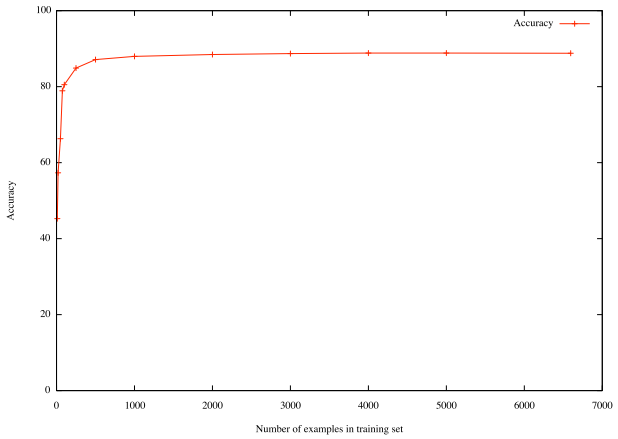

Figure 4: Learning curve of the average performance of our system, as a function of the number of training examples. This experiment is done with $K = 3$ on the *TestWiki* test set.

show that our approach generalizes well across semantic classes and, even though we use Wikipedia as the key knowledge source, it can handle well also concepts that are not mentioned in Wikipedia. We showed that our approach has significantly better coverage and accuracy than existing knowledge acquisition and relation identification approaches. While this study focused on developing and analyzing our approach, our immediate future step will be the evaluation of it in the context of supporting textual inference applications.

7. REFERENCES

[1] M. Banko, M. Cafarella, M. Soderland, M. Broadhead, and O. Etzioni. Open information extraction from the web. In *Proc. of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 2670–2676, 2007.

[2] M. Banko and O. Etzioni. The tradeoffs between open and traditional relation extraction. In *Proceedings of ACL-08: HLT*, pages 28–36, Columbus, Ohio, June 2008. Association for Computational Linguistics.

[3] R. Braz, R. Girju, V. Punyakanok, D. Roth, and M. Sammons. An inference model for semantic entailment in natural language. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 1678–1679, 2005.

[4] I. Dagan, O. Glickman, and B. Magnini, editors. *The PASCAL Recognising Textual Entailment Challenge.*, volume 3944. Springer-Verlag, Berlin, 2006.

[5] D. Davidov and A. Rappoport. Unsupervised discovery of generic relationships using pattern clusters and its evaluation by automatically generated SAT analogy questions. In *Proceedings of ACL-08: HLT*, pages 692–700, Columbus, Ohio, June 2008. Association for Computational Linguistics.

[6] C. Fellbaum. *WordNet: An Electronic Lexical Database*. MIT Press, 1998.

[7] Y. Freund and R. E. Schapire. Large margin classification using the perceptron algorithm. *Machine Learning*, 37(3):277–296, 1999.

[8] E. Gabrilovich and S. Markovitch. Computing semantic relatedness using wikipedia-based explicit semantic analysis. In *IJCAI*, pages 1606–1611, 2007.

[9] A. Haghighi, A. Ng, and C. Manning. Robust textual inference via graph matching. In *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*, pages 387–394, Vancouver, British Columbia, Canada, October 2005. Association for Computational Linguistics.

[10] M. A. Hearst. Automatic acquisition of hyponyms from large text corpora. In *Proceedings of the 14th conference on Computational linguistics*, pages 539–545, Morristown, NJ, USA, 1992. Association for Computational Linguistics.

[11] Z. Kozareva, E. Riloff, and E. Hovy. Semantic class learning from the web with hyponym pattern linkage graphs. In *Proceedings of ACL-08: HLT*, pages 1048–1056, Columbus, Ohio, June 2008. Association for Computational Linguistics.

[12] B. Maccartney and C. D. Manning. Modeling semantic containment and exclusion in natural language inference. In *Proceedings of the 22nd International Conference on Computational Linguistics (Coling 2008)*, pages 521–528, Manchester, UK, August 2008. Coling 2008 Organizing Committee.

[13] M. Mohler and R. Mihalcea. Text-to-text semantic similarity for automatic short answer grading. In *Proceedings of the 12th Conference of the European Chapter of the ACL (EACL 2009)*, pages 567–575, Athens, Greece, March 2009. Association for Computational Linguistics.

[14] M. Paşca. Organizing and searching the world wide web of facts – step two: harnessing the wisdom of the crowds. In *WWW '07: Proceedings of the 16th international conference on World Wide Web*, pages 101–110, New York, NY, USA, 2007. ACM Press.

[15] M. Paşca and B. Van Durme. Weakly-supervised acquisition of open-domain classes and class attributes from web documents and query logs. In *Proceedings of ACL-08: HLT*, pages 19–27, Columbus, Ohio, June 2008. Association for Computational Linguistics.

[16] S. P. Ponzetto and M. Strube. Deriving a large scale taxonomy from wikipedia. *AAAI-07*, 2007.

[17] W. W. C. Richard C. Wang. Automatic set instance extraction using the web. In *Proceedings of ACL-09: IJCNLP*, 2009.

[18] D. Roth and W. Yih. A linear programming formulation for global inference in natural language tasks. In H. T. Ng and E. Riloff, editors, *Proc. of the Annual Conference on Computational Natural Language Learning (CoNLL)*, pages 1–8. Association for Computational Linguistics, 2004.

[19] L. Sarmento, V. Jijkuon, M. de Rijke, and E. Oliveira. "more like these": growing entity classes from seeds. In *CIKM '07: Proceedings of the sixteenth ACM conference on Conference on information and knowledge management*, pages 959–962, New York, NY, USA, 2007. ACM.

[20] S. Sekine. On-demand information extraction. In *Proc. of the Annual Meeting of the ACL*, pages 731–738, 2006.

[21] R. Snow, D. Jurafsky, and A. Y. Ng. Learning syntactic patterns for automatic hypernym discovery. In *Advances in Neural Information Processing Systems (NIPS 2005)*, November 2005. This is a draft version from the NIPS preproceedings; the final version will be published by April 2005.

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

[23] F. M. Suchanek, G. Kasneci, and G. Weikum. Yago: A Core of Semantic Knowledge. In *16th international World Wide Web conference (WWW 2007)*, New York, NY, USA, 2007. ACM Press.

[24] P. P. Talukdar, T. Brants, M. Liberman, and F. Pereira. A context pattern induction method for named entity extraction. In *Proceedings of the Tenth Conference on Computational Natural Language Learning (CoNLL-X)*, pages 141–148, New York City, June 2006. Association for Computational Linguistics.

[25] P. P. Vishnu Vyas. Semi-automatic entity set refinement. In *Proceedings of NAACL-09*, 2009.

[26] R. Wang and W. Cohen. Language-independent set expansion of named entities using the web. pages 342–350, Oct. 2007.

[27] R. Wang and W. Cohen. Iterative set expansion of named entities using the web. pages 1091–1096, Dec. 2008.

[28] Z. Zhang. Weakly-supervised relation classification for information extraction. In *CIKM '04: Proceedings of the thirteenth ACM international conference on Information and knowledge management*, pages 581–588, New York, NY, USA, 2004. ACM Press.