Using Constraint-based Inference to Identify Relational Knowledge

Quang Do
Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, IL 61801
quangdo2@illinois.edu

Dan Roth
Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, IL 61801
danr@illinois.edu

ABSTRACT

We present a novel approach to identifying relations between a pair of concepts. We focus on identifying relations that are essential to support textual inference: determining whether two concepts have an ancestor relation, a sibling relation, or no relation. We develop a machine learning-based approach that makes use of Wikipedia as a main source for background knowledge, but we also propose an effective approach of searching the Web to improve the coverage of our method and support inference between concepts not present in Wikipedia. Our key innovation is that, in order to accurately determine the relations between concepts C_1 and C_2, we consider an automatically generated collection of related concepts, evaluate all pairwise relations, and use constraint-based inference to force them to cohere, thus improving the local prediction of the pairwise relation identification. We demonstrate that the inference technique significantly enhances the local prediction methods and consequently exhibit very large improvements over methods using existing knowledge sources.

Categories and Subject Descriptors

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

General Terms

Relation identification

Keywords

Concept, relation, classification, inference, Wikipedia

1. INTRODUCTION

Many data and knowledge management problems require some sort of textual inference. These range from query expansion and interpretation in information retrieval to query schema matching and question answering. Especially, in advanced web search and contextual advertising, textual inference plays an important role as the core technique to finding related information that can attract users' interest. For instance, while searching for the reviews of *Nikon D90*, one may also be interested in reading the reviews for other *Nikon's cameras*, or *cameras* in general. To satisfy users' interest, advanced search engines need to be equipped with textual inference techniques that can perform search on related concepts in the index of web documents. Textual inference 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.

In this paper, we present a novel approach to a fundamental problem that recognizes basic relations between two given concepts. This view of relations is different from that of Open Information Extraction [1] and On-Demand Information Extraction [24], which aim at extracting large databases of open-ended facts building on the concepts occurring together in some local context. It is also different from the supervised relation extraction[21] effort which requires additional supervised data to learn new relations.

While there has been a large body of work in the direction of extracting all *easy to find* facts in a given corpus [2, 7, 17], it is not clear how to make use of these if one is interested in determining whether two given concepts, A and B, are related and how. The key reason is that the knowledge acquisition methods alluded to 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.

We focus here on relations between *any* two concepts, but only basic relations such as *ancestor* and *sibling*. This is motivated by work on Textual Inference [6, 13, 3], where it has been argued quite convincingly, (e.g. [14]), that many inferences are largely compositional and depend on the ability of models to recognize these basic relations between 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 a test set of a textual entailment challenge:

> **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 paper proposes to address the problem of relation identification and classification in a form that is directly applicable to textual inference. We focus here on the *ancestor* relation and the *sibling* relation that were identified as key relations also in [14] (they call a *sibling* relation an *alternation*, and our *ancestor* relation *forward entailment* and *backward entailment*). Following their argument, we expect that the resource developed in this work can be used compositionally to support robust textual inference.

We develop an approach that makes use of Wikipedia and that, given a pair of concepts will, on the fly, recognize and classify the relation between these concepts. Because Wikipedia is a vast, rapidly-changing resource, our approach needs to take into account its noisiness and non-uniformity. Our algorithmic approach therefore treats Wikipedia and its category structure as an open resource and uses statistical text mining techniques to gather robust information. While Wikipedia has broad coverage, there is a need to go beyond it. We suggest a simple but efficient technique to accomplish this using web search, and show its effectiveness when at least one of the target concepts is not mentioned in Wikipedia. Our key technical contribution is a novel approach that makes use of constraint-based inference with relational constraints to accurately identify relations; we make use of our machine learning approach multiple times, on automatically generated network of concepts that are related to the target pair, and use constrained optimization techniques to force these decisions to be coherent, thus improving the accuracy of the decision.

The key contributions of this paper are (i) the definition of a relation identification problem so that it is directly relevant to supporting textual inference, (ii) the development of a robust and accurate machine learning-based approach to address this problem, and (iii) a novel approach that incorporates prior knowledge within a constraint-based inference model to accurately identify concept relations. We show that our system performs significantly better than other systems built on existing large-scale resources in identifying relational knowledge.

2. RELATED WORK

To the best of our knowledge, no previous study has directly addressed the problem we study here in this form: given two concepts, identify the relation between them. However, there are several lines of related work that we would like to mention and compare to.

In [25], the authors construct a hypernym-only classifier between concepts. The classifier builds on dependency path patterns discovered in sentences that contain noun pairs in hypernym and hyponym relations. The best system presented there is a hypernym-only classifier that is trained with hundreds of thousands of dependency paths extracted from Wikipedia. The system outperforms the best WordNet classifier in identifying coordinated terms,, improving the relative F-score by over 54%. More recently, [26] proposed a method to train their (m,n)-cousin relationship classifier (defined similarly to our *sibling* relation) to significantly increase coverage and improve the F-score by 23% over the WordNet-2.1 hypernym classifier. The classifier is then applied to build the *extended WordNet* by augmenting WordNet-2.1 with over $400,000$ synsets. These works are different than ours because they largely build on the redundancy of information in the web—many related terms appear often enough in some simple explicit form in close proximity in a sentence. In our work, we relax this fact and identify relations between any two input concepts, exhibiting significantly better coverage and accuracy.

Other related works attempts to build relational knowledge bases (semantic taxonomies and ontologies) that represent entities and relations among them. E.g. [18] generates a taxonomy using Wikipedia as the knowledge source. They use the category system of Wikipedia as a conceptual network consisting of the subsumption (*isa*) relation. Similarly [27] presents the the YAGO ontology, which is automatically constructed using both Wikipedia and WordNet to mine entities and their relations. The YAGO approach builds on the *infoboxes* and *category pages* in Wikipedia and links to the clean taxonomy of concepts in WordNet. YAGO provides many useful relations between entities such as *subClassOf*, *bornOnDate*, *locatedIn*, and *type*. One could possibly use the taxonomy in [18] or the YAGO ontology, with some additional work, to find relation of two input concepts. Our work differs from this line of work in that we do not build an offline knowledge base, but rather directly identify relations between input concepts. Nevertheless, as we show, our approach has both better coverage and better accuracy.

3. IDENTIFYING RELATIONAL KNOWLEDGE

In in section, we present our overall algorithm addressing the problem of identifying relational knowledge. The input of our problem is a pair of concept (X, Y), and the output is the relation between X and Y. In this paper, we focus on addressing the following relational knowledge as the output of the problem:

1. X is an ancestor of Y, denoted by $X \leftarrow Y$.
2. X is a child of Y, 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 $X \nleftrightarrow Y$.

Our approach consists of two key components:

1. A machine learning-based algorithm to classify concept relations.
2. A constraint-based inference model to make final decision using relational constraints enforced among concept relations.

Given two concepts X, and Y, we first determine if they are *Wikipedia concepts* or *non-Wikipedia concepts* by searching the concept space in Wikipedia. If they are *non-Wikipedia concepts*, we use a web search to look for their replacements. The replacements are expected to be *Wikipedia concepts* and in the same semantic class with the input *non-Wikipedia concept*. We then use two input concepts (or their replacements) to disambiguate themselves. Following, a machine learning-based classifier is used to identify the concept relations. Finally, the constraint-based inference model is performed to make the final decision. Figure 1 presents our overall algorithm.

Although most commonly used concepts can be found in Wikipedia, there is still a need to cover the *non-Wikipedia concepts* to improve the coverage of the algorithm.

Function `findReplacement`(A, B) takes a *non-Wikipedia concept* A and a supporting concept B as its input. The

```
RELATIONAL KNOWLEDGE IDENTIFICATION ALGORITHM
    INPUT: A concept pair $(X, Y)$
    OUTPUT: Relation of $X$ and $Y$

    If $X$ is a non-Wikipedia concept then
        $X \leftarrow$ findReplacement($X, Y$)
    End if
    If $Y$ is a non-Wikipedia concept then
        $Y \leftarrow$ findReplacement($Y, X$)
    End if

    $(X, Y) \leftarrow$ disambiguate($X, Y$)
    $R =$ classifyRelation($X, Y$)
    $R^* =$ inference($X, Y, R$)

    RETURN: $R^*$;
```

Figure 1: Relational Knowledge Identification Alg.

function searches the web to find a *Wikipedia concept* A' which is in the same semantic class of A to be its replacement. Our method was motivated by [23]. In our work, we use the Yahoo! Web Search APIs[1] to search for list structures in web documents such as "... $\langle$*delimiter*$\rangle$ c$_a$ $\langle$*delimiter*$\rangle$ c$_b$ $\langle$*delimiter*$\rangle$ c$_c$ $\langle$*delimiter*$\rangle$ ...". The search query is "A AND B" (e.g. *"bass"* AND *"trout"*). The delimiters used in our experiments are comma(,), punctuation(.), and asterisk(*). For text snippets that contain the patterns of interest, we extract c_a, c_b, etc. as replacement candidates. To reduce noise from concepts extracted from the snippets, we constrain the list structure to contain at least 4 concepts that are no longer than 20 characters each. Once a list of replacement candidates is extracted, the candidates are ranked based on their occurrence frequency. The top candidate in the Wikipedia concept space is used to replace A. To determine whether a concept is in the *Wikipedia concept* space, we use the concept disambiguation algorithm presented in Sec. 4.1.

In the following sections, we describe other components of our algorithm in details.

4. LEARNING CONCEPT RELATIONS

Given two input concepts (X, Y), we first disambiguate them to get lists of relevant Wikipedia articles. We then extract features of disambiguated concepts and train a supervised multi-class classifier to classify relations between them.

4.1 Concept Disambiguation

Given two input concepts (X, Y), we must first disambiguate them. The more accurate the disambiguation is, the more precise the relation recognition is. 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, Wisconsin - USA*. Our approach is motivated by observing that the concept *Bush* can be used to disambiguate *Ford* and vice versa. Ultimately, we can place articles about *George W. Bush* and *Gerald Ford* the top of a retrieved articles list for the two input concepts *Bush* and *Ford*.

Figure 2 shows the concept disambiguation algorithm.

Function `ESA`(*query*) retrieves the most semantically relevant articles in Wikipedia to *query*. We use Explicit Semantic Analysis (ESA) [11] to retrieve semantically relevant articles in Wikipedia for two given concepts. For all articles in the relevant list, we extract and keep their categories by the function `getCategories`(*list*). All categories are then tokenized and weighted using the TF-IDF score. Only top N tokens, which are different from the two input concepts, are considered. Each input concept is then concatenated with the top weighted tokens to create a new query that, in turn, is the input for the `search`(*query*) function. Our search function returns relevant articles in Wikipedia for the input *query*. The lists of relevant articles are later used to provide features for the local relation classifier. If a returned list is empty, its corresponding concept is considered as a *non-Wikipedia concept*.

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

```
CONCEPT DISAMBIGUATION ALGORITHM
    INPUT: Concept pair $(X, Y)$; Number of top tokens $N$
    OUTPUT: Two list of relevant articles for $X$ and $Y$.

    Query $q \leftarrow$ Concatenating $X$ and $Y$;
    $\mathcal{L} =$ ESA($q$);
    $\mathcal{C} =$ getCategories($\mathcal{L}$);
    $\mathcal{T} =$ tokenize($\mathcal{C}$);
    $\mathcal{T}_{top} =$ topTfidf($\mathcal{T}, N$);

    Query $q_X \leftarrow$ Concatenating $X$ and tokens in $\mathcal{T}_{top}$;
    Query $q_Y \leftarrow$ Concatenating $Y$ and tokens in $\mathcal{T}_{top}$;
    $\mathcal{A}_X =$ search($q_X$);
    $\mathcal{A}_Y =$ search($q_Y$);

    RETURN: $\mathcal{A}_X$ and $\mathcal{A}_Y$;
```

Figure 2: Concept disambiguation algorithm. The algorithm takes two input concepts, disambiguates them and returns a list of relevant Wikipedia articles for each.

4.2 Learning Relations

We first extract expressive features associated with the two input concept pairs. These features are necessarily independent of the semantic class of the given concepts so that our classifier can generalize well. Recall that, after concept disambiguation, each input concept is associated with a list of relevant articles in Wikipedia. In other words, from these articles, an input concept is represented with several Wikipedia titles, texts, and categories. 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 a learning algorithm, we use a regularized averaged Perceptron [10]. The features selected for our learning problem include the words used in 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.

4.2.1 Bags of Words

One of the most important features in recognizing relations between concepts is the words used 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; and the categories classify the concept into one or more concept groups which can be further categorized. To collect the categories for a concept, we

Concept/Title	**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.*

take the categories of its associated articles and go up K levels in the Wikipedia category system. In our experiments, we use abstracts of Wikipedia articles instead of whole texts.

We define four bag-of-word features associated with any two given concepts X and Y: (1) the degree of similarity between the texts of concept X and the categories of concept Y, (2) the similarity between the categories of X and the texts of Y, (3) the similarity between the text of X and the text of Y, and (4) the similarity between the categories of X and the categories of Y.

Table 1 shows three related concepts and their associated articles in Wikipedia[2]. We see the overlap in the words used in the title, the text and the categories of these three concepts. The overlap in word usage determines the degree of similarity of concepts holding relations of interest.

There are several ways to calculate the degree of similarity between two text fragments [15]. We use the cosine similarity metric. The following equation shows the cosine similarity metric applied to the term vectors T_1 and T_2 of two text fragments.

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

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

4.2.2 Association Information

Another useful feature that can help in recognizing the relation between two concepts is their association information. We capture the association information between two concepts X and Y by measuring their pointwise mutual information, defined in the following equation.

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

where X and Y are two input entities, N is the total number of documents in Wikipedia, and $f(.)$ is a function returning document frequency. We measure association information at the document level.

4.2.3 Overlap Ratio

Wikipedia articles are not only helpful for definitions and explanations of concepts, they are also useful for categorizing concepts into groups and topics. We capture the fact that the titles of a concept often 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 Tab. 1.) 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 to be a *common phrase* if it appears in the titles of X and the categories of Y and is one of the following: (1) the *whole string* of a category of Y, or (2) the *head* in its root form of a category of Y, or (3) the *post-modifier* of a category of Y. We use the Noun Group Parser from [27] to extract the *head* and *post-modifier* 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-modifier *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.* Therefore, we have a strong indication that *Chicago* is a descendant of both *Illinois* and *City.* We measure the overlap ratio between the titles of concept X and the categories of concept Y up to K levels in the Wikipedia category system by using the Jaccard similarity coefficient.

$$\sigma_{ttl}^{K}(X,Y) = \frac{\left|L_X \cap \mathcal{P}_Y^K\right|}{\left|L_X \cup \mathcal{P}_Y^K\right|}$$

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 $\sigma_{ttl}^{K}(Y,X)$ as one of our features.

We also use a feature that captures 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. The Jaccard similarity coefficient for overlap ratio between the categories of two concepts X and Y is shown in the following equation.

$$\sigma_{cat}^{K}(X,Y) = \frac{\left|\mathcal{Q}_X^K \cap \mathcal{Q}_Y^K\right|}{\left|\mathcal{Q}_X^K \cup \mathcal{Q}_Y^K\right|}$$

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

All of the features described above are used to train a local multi-class classifier to recognize relations between concepts.

5. INFERENCE WITH RELATIONAL CONSTRAINTS

Analyzing concepts and the relations between them reveals several relational constraints among the relations iden-

[2]Note that each concept can be associated with more than one article in Wikipedia depending on the list of relevant articles returned by the concept disambiguation algorithm.

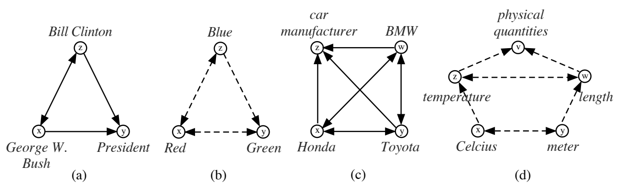

Figure 3: Examples of n-vertex concept networks formed by two input concepts x, y, and related concept z, w and v. The relation between concepts is predicted by a local classifier (see Sec. 4.2). Concept networks (a) and (c) show valid combinations of edges, whereas (b) and (d) demonstrate invalid combinations of edges. (b) and (d) form two relational constraints that we do not allow. For simplicity, we do not draw *no relation* edges in (d).

tified for the target pair and those identified for related concepts. For instance, concept *George W. Bush* cannot be an ancestor or sibling of concept *president* if we are confident that concept *president* is an ancestor of concept *Bill Clinton*, and *Bill Clinton* is a sibling of concept *George W. Bush*. Another example would be that if concepts *red* and *green* are known to be siblings, and concept *blue* is also known to be a sibling of *red*, the prediction identifying that *green* is an ancestor of *blue* should be invalid. We call the combination of concepts and their relations *concept network*. Fig. 3 shows some n-vertex concept networks consisting of two input concepts (x, y), and additional concepts z, w, v. Fig. 3(b) and 3(d) illustrate two relational constraints. In general, n concepts can be involved to construct n-vertex concept network ($n > 2$). In this paper, we focus on observing 3-vertex concept networks consisting of the two input concepts and an additional one. However, our formalization applies to general n-vertex concept networks.

The aforementioned observations show that, if we can obtain additional concepts that are related to the two input concepts, we can enforce such relational constraints as prior knowledge and guide the final decision of the relation identifier. Our formulation follows constraint-based formulations that were introduced in the NLP community and were shown to be very effective in exploiting declarative background knowledge as a way to achieve significant improvement in performance [21, 20, 4, 8, 19]. While it is also possible to inject prior knowledge *indirectly*, by adding more features, it has been argued (e.g., [22, 5]) that a *direct* way, in the form of soft or hard constraints, is beneficial due to better expressivity and simpler learning. Our model follows this approach.

5.1 Constraints as Prior Knowledge

Let two input concepts (x, y), and a set of additional concepts $\mathcal{Z}^* = \{z_1, z_2, ..., z_m\}$. For a subset $\mathcal{Z} \in \mathcal{Z}^*$, a set of concept networks is constructed. Each concept network contains two input concepts x, and y, and all additional concepts in $\mathcal{Z}$. Every two concepts in a concept network are connected by their relations which are the four relations of interest in this paper. A local relation classifier is used to give weights for 4 relation classes of an edge in a concept network. We use e and $w(e)$ to denote an edge in a concept network and its corresponding weight, respectively. If n is the number of concepts in a network ($n > 2$), there will be $\left[\frac{1}{2}n(n-1)\right]^4$ concept networks that can be constructed because there are 4 relations for every 2 concepts. For instance, with $n = 3$, we have 64 concept networks. A relational constraint is defined as an invalid concept networks that is not allowed to happen. Figure 3(b) demonstrates a constraint where the combination of edges is (*sibling*, *sibling*, *ancestor*) moving clockwise direction with respect to x and y.

Let $\mathcal{C}$ be a list of relational constraints. Following the approaches to inject prior knowledge directly in [5], we incorporate relational constraints into our model to choose the best network t^* from a set of concept networks constructed from $\langle x, y, \mathcal{Z}\rangle$. The scoring function is a linear combination of the edge weights $w(e)$ and the penalties ρ_k of concept networks violating constraint $C_k \in \mathcal{C}$.

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

In Eqn. 1, function $d_{C_k}(t)$ measures the degree that concept network t violates constraint C_k. In our work, relational constraints are mined from the training data (see Sec. 5.2). Therefore, the selected constraints are considered to have high confidence. We, thus, use them as hard constraints and set their penalty ρ_k to ∞ [5]. Objective function 1 allows us to pick the best setting of all edges connecting concepts in the networks with respect to relational constraints.

After having the model to choose the topmost concept network for a particular subset of additional concepts $\mathcal{Z} \in \mathcal{Z}^*$, we are going to make the final decision on the relation between x and y. Note that for each subset of additional concepts $\mathcal{Z} \in \mathcal{Z}^*$, the topmost concept network contains the most likely relation ℓ of the two input concepts x and y. After collecting all the topmost concept networks by going through all $\mathcal{Z} \in \mathcal{Z}^*$, the topmost concept networks are divided into four group with respect to the relation of x and y in each network. Each group of the topmost concept networks is denoted by $\mathcal{T}_\ell$. To make the final decision concerning the relation of two input concepts (x, y), we solve the objective function defined in Eqn. 2.

$$\begin{aligned}
\ell^* &= \text{argmax}_\ell \frac{1}{|\mathcal{T}_\ell|} \sum_{t \in \mathcal{T}_\ell} \lambda_t score(t) \qquad (2) \\
&= \text{argmax}_\ell \frac{1}{|\mathcal{T}_\ell|} \sum_{t \in \mathcal{T}_\ell} \lambda_t \left(\sum_{e \in t} w(e) - \sum_{k=1}^{|\mathcal{C}|} \rho_k d_{C_k}(t) \right)
\end{aligned}$$

where

$$\lambda_t = \frac{\text{\# of occurrences of } t}{\text{\# of predicted concept networks in training data}}$$

is the occurrence probability of concept network t in the training data with additional concepts. To measure λ_t, a local relation classifier is used to predict the best relation between concepts in the concept networks in the training data.

5.2 Constraint Selection

In Eqn. (2), a list of relational constraints $\mathcal{C}$ is used. Recall that a relational constraint is an invalid concept network that is not allowed to happen. In our work, relational constraints are mined from the training data by applying the

```
Algorithm FORWARD CONSTRAINT SELECTION
    INPUT: Data set with related concepts $\mathcal{D} = \langle(x, y, \ell, \mathcal{Z}^*)\rangle$
        A trained local relation classifier $\mathcal{L}$.
        A list of concept networks $\mathcal{T} = \{t_1, t_2, \ldots, t_m\}$;
    OUTPUT: Constraint set $\mathcal{C}$.

    $\mathcal{C} = \emptyset$; $isIncreasing$ = true;
    $bestAcc$ = evaluate($\mathcal{D}$, $\mathcal{L}$, $\mathcal{C}$);
    While ($isIncreasing$) do
        $isIncreasing$ = false;
        For each $t \in \mathcal{T}$ do
            $\mathcal{C} = \mathcal{C} \cup \{t\}$;
            $acc$ = evaluates($\mathcal{D}$, $\mathcal{L}$, $\mathcal{C}$);
            If ($acc > bestAcc$) then
                $t^* = t$; $isIncreasing$ = true;
                $bestAcc = acc$;
            End if
            $\mathcal{C} = \mathcal{C} \backslash \{t\}$;
        End for
        If ($isIncreasing$) then
            $\mathcal{C} = \mathcal{C} \cup \{t^*\}$; $\mathcal{T} = \mathcal{T} \backslash \{t^*\}$;
        End if
    End while

    RETURN: $\mathcal{C}$;
```

Figure 4: Forward constraint selection algorithm. Function evaluate($\mathcal{D}$, $\mathcal{L}$, $\mathcal{C}$) evaluates the system on all examples and their related concepts in $\mathcal{D}$ using a trained local classifier $\mathcal{L}$ with respect to a set of constraints $\mathcal{C}$ by solving Eqn. (2).

forward feature selection technique discussed in [12]. The algorithm starts with an empty set of constraints $\mathcal{C}$, then grows the constraint set gradually by including in the set the *best* concept network from each iteration. The *best* concept network is defined as the constraint used in Eqn. (2) that improves the system's performance most. Our *forward constraint selection* algorithm is described in Fig. 4. Function `evaluate`($\mathcal{D}$, $\mathcal{L}$, $\mathcal{C}$) evaluates the system on all examples and their related concepts in $\mathcal{D}$ using a trained local classifier $\mathcal{L}$ with respect to a set of constraints $\mathcal{C}$ by solving the objective function in Eqn. (2)

One of the inputs of the algorithm, $\mathcal{T}$, is a set of m concept networks. At each iteration, all concept networks $t \in \mathcal{T}$ are tried; at the end of the iteration, the *best* network is added to $\mathcal{C}$ and removed from $\mathcal{T}$. In this paper, for simplicity, we only use 3-vertex concept networks, thus, $m = 64$. In general, n-vertex concept networks can also be selected as relational constraints and used in our model by decomposing n-vertex networks into 3-vertex networks to allow greedy search performed in the concept network space.

5.3 Related Concepts Extraction

The input of our problem is a pair of two concepts. To apply our proposed constraint-based inference model, we need to acquire concepts additional to the input pair. It can be argued that, with strong relational constraints, one can use random concepts as additional concepts to perform the inference. However, there is a high probability that a random additional concept will simply produce *no relation* from two input concepts. This will not help the inference model much because there are many other relational constraints with relations other than *no relation.* To address this issue, we investigate different approaches to obtain additional concepts which are related to input concepts. From now on, we refer to additional concepts as *related concepts.*

```
YAGO QUERY PATTERNS
    INPUT: concept "$X$"
    OUTPUT: lists of ancestors, siblings, and children of "$X$"

    PATTERN 1:
        "$X$" MEANS ?A
        ?A TYPE ?B
        ?C TYPE ?B
    PATTERN 2:
        "$X$" MEANS ?D
        ?E TYPE ?D

    RETURN: ?B, ?C, ?E as
            lists of ancestors, siblings, and children, resp.
```

Figure 5: Our patterns used to obtain lists of direct ancestors, siblings, and children from YAGO.

In the first direction, as the first step to verifying the correctness of our proposed constraint-based inference model, we manually add *gold related concepts*, which are very related to input concepts. By using *gold related concepts*, a significant improvement in the system's performance is necessary to prove the correctness of the inference model. Our experiments (see Sec. 6.3.2) following this direction indeed prove that our proposed inference model is correct and accurate.

However, in real world applications, *gold related concepts* are not available. We, therefore, propose an approach that makes use of YAGO ontology [27] to provide related concepts. It is worth noting that YAGO is chosen over the Wikipedia category system used in our work because YAGO is a clean ontology built by carefully combining Wikipedia and lexical database WordNet.[3]

In YAGO model, all objects (e.g. *cities*, *people*, etc.) are represented as *entities.* Following the YAGO model, our input concepts are considered to be words. To map our input concepts to entities in YAGO, we use MEANS relation. Furthermore, similar entities are grouped into *classes.* This allows us to obtain direct ancestor of an *entity* by using TYPE relation which gives us the entity's *classes.* By using two relations MEANS and TYPE in YAGO model, we can obtain direct ancestors, siblings, and also children of an input concept using the following query patterns. By default, we use this approach to provide related concepts for our constraint-based inference model.

In Fig. 5, PATTERN 1 is used to get lists of ancestors and siblings of a given concept. For example, with concept "X"= "*honda civic*", PATTERN 1 returns lists of ancestors {*wikicategory_1970s_automobiles, wordnet_car_102958343, wikicategory_Honda_vehicles,* ...}, and siblings {*Ford_Capri, Mercury_Comet, Honda_Jazz, Honda_Accord,* ...}. The prefix and suffix annotation of the ancestors are dropped. In the case that ancestors are noun phrases, we only use the head of the phrase. We use the Noun Group parser from [27] to extract the *head* of a noun phrase. In the other hand, PATTERN 2 returns an empty list of children of "*honda civic*", because its corresponding entity `Honda_Civic` is at the deepest level in YAGO ontology. The list of children will be non-empty if "X" is some generic concept such as "*actor*", "*president*", etc.

[3]However, YAGO by itself is worse than our approach in identifying concept relations (see Sec. 6.2.)

A snapshot of the original dataset	
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.

Concept pair examples		
Relation	Concept X	Concept Y
$X \leftarrow Y$	actor food	Mel Gibson rice
$X \rightarrow Y$	Makalu Monopoly	mountain game
$X \leftrightarrow Y$	Paris copper	London oxygen
$X \not\leftrightarrow Y$	Roja egg	C++ Vega

Table 3: Some examples in our data set.

6. EXPERIMENTAL STUDY

In this section, we first describe the data sets used in our experimental study. Follow that, we present the experiments to evaluate our overall system and compare it with other systems using existing large-scale resources. We also give a deeper understanding about our system by performing several experimental analyses.

6.1 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 [16]. This dataset was manually constructed[4] and was used to evaluate several information extraction tasks [16, 17]. Each semantic class is an incomplete set of representative instances and has about 272 instances in average. The smallest class is *search engine* with 25 instances, and the largest class is *actor* with 1500 instances. Table 2 shows a snippet of the dataset. 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*) and classes with common nouns (e.g. *basic food* with *rice*, *milk*).

In the original dataset, the semantic class names are not often written in a meaningful form, such as *chemicalelem*, and *proglanguage*. These names are not valid concepts. 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*, *terrorism*. We use these expansions for all systems in evaluation.

An example in our learning problem is a pair of two concepts (X, Y) such as (*city*, *Dubai*), (*lead*, *aluminum*). Note that in this paper, we refer to both the name of the semantic classes and their instances as concepts, such as *city* and *Dubai*. We pair the semantic classes and instances in the original data set to create training and testing examples. The examples cover all types of relational knowledge of interest: $X \leftarrow Y$, $X \rightarrow Y$, $X \leftrightarrow Y$, and $X \not\leftrightarrow Y$. Specifically, examples are created with the following guidelines.

[4]Private communication with Marius Pasca, 2009.

Data	$X \leftarrow Y$	$X \rightarrow Y$	$X \leftrightarrow Y$	$X \not\leftrightarrow Y$	Total
Training	1,739	1,754	1,664	1,802	6,959
TestAll	3,045	3,025	2,965	2,965	12,000

Table 4: Details of the training and test sets with the number of examples in each relation class. The *Training* set contains only examples in Wikipedia. *TestAll* includes *non-Wikipedia* examples.

- $X \leftarrow Y$ examples: For each semantic class, we pair the name of the class with its instances. These examples have the genearl form (*semantic class X*, *child of X*).
- $X \rightarrow Y$ examples: These examples have the general form (*concept X*, *semantic class of X*).
- $X \leftrightarrow Y$ examples: The general form of these examples is (*concept X*, *concept Y*), where X and Y are two instances of a semantic class, and $X \neq Y$.
- $X \not\leftrightarrow Y$ examples: To make examples with two concepts having no relation, we pair either a semantic class name and an instance of another semantic class (and vice versa), or an instance in a semantic class and another instance in other classes.

We randomly create $20,000$ examples following the guidelines above. Table 3 shows some examples created from the original data set. We use $8,000$ examples for the training set, and $12,000$ examples for the test set. We refer to the 12,000-example test set as the *TestAll* test set. From the training set, we discard examples with one or both concepts not in Wikipedia (*non-Wikipedia examples*). This results in a training set of $6,959$ examples. It is important to note that, we do not discard *non-Wikipedia* examples in the *TestAll* test set. Table 4 shows the statistics of the training and test data.

To evaluate our system, we use a snapshot of Wikipedia from 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. We also do not use articles without titles. After pre-processing, 5,503,763 articles remain. We index the articles using the Apache Lucene Information Retrieval library[5]. Lucene is a high-performance text search library written in Java and widely used as an off-the-shelf IR system.

6.2 Overall Results and Comparison

To the best of our knowledge, no prior work directly targets the problem of identifying the relational knowledge defined in this paper. Most prior work which relates to our problem focuses on building lexical taxonomy or knowledge ontology [26, 18, 27]. By searching the knowledge base, one can find the answer for relationships between input concepts. We compare our system with three other systems which are built upon different existing resources.

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

Overall results and comparison				
K	STRUBE 07	SNOW 06	YAGO 07	OURS
1	23.83	40.24	64.43	**82.13**
2	23.84	42.63	63.94	**84.79**
3	23.88	40.96	62.02	**85.07**
4	24.32	40.65	60.57	**84.08**

Table 5: Performance of our overall system compared to other systems. Performances are measured by accuracy. The *TestAll* test set with $12,000$ examples is used in this experiment. K is the number of levels to climb up in the hierarchical structure of knowledge sources as described in the text.

Comparison on special data sets				
Data set	STRUBE 07	SNOW 06	YAGO 07	OURS
TestWiki	24.59	44.34	70.29	**90.92**
TestWn	24.13	47.79	70.81	**90.76**

Table 6: Comparison of systems' performance with different test sets derived from the *TestAll* test set. *TestWiki* contains $10,456$ examples in Wikipedia, and *TestWn* contains $8,625$ examples in the *extended WordNet*. The best performance of each system (by varying K) is reported.

1. STRUBE 07 uses a large scale taxonomy which was derived from Wikipedia [18], 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 [18]. Given an input with two concepts 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, using *isa* links of articles, 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 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 from March, 2008[6].

2. SNOW 06 uses the *extended WordNet* [25, 26] as background knowledge. To build the *extended WordNet*, the authors first identified lexico-syntactic patterns indicative of hypernymy from corpora and use them 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 [9], 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 effective background knowledge for identifying the relational knowledge of interest by looking for the input concepts in the extended WordNet tree in all possible senses of the concepts and then inferring their relationship. We also vary the value of K as the number of levels to go up on the WordNet tree from input concepts to find their common subsumptions.

3. YAGO 07 uses YAGO ontology [27] as the main source of background knowledge. Because YAGO ontology is a combination of Wikipedia and WordNet (see our brief description in Sec. 5.3), this system is expected to be powerful in recognizing concept relationships. To access a concept's ancestors and siblings, we combine PATTERN 1 in Fig. 5 and the SUBCLASSOF relation in YAGO model to go up on the ontology. The SUBCLASSOF relation can be cascaded K times to climb up in the ontology.

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

Our system, OURS, is described in Fig. 1. All systems are evaluated on the *TestAll* test set with K from 1 to 4. Table 5 shows the comparison of the systems in details. It is shown that our system significantly outperforms other systems implemented using existing sophisticated resources. Our system also uses Wikipedia as its main background knowledge. However, ours is superior to other systems because it employs advance machine learning techniques along with a powerful and effective constraint-based inference model.

Because the systems in the previous experiments do not use the same source of background knowledge, we, furthermore, perform experiments in which these systems are compared with different data sets covered in different knowledge sources. This experiment provides a fair comparison for systems that use different resources as their background knowledge. New data sets are derived from the *TestAll* test set. The first derived data set is *TestWiki* which contains only *Wikipedia concepts*. By removing all examples with *non-Wikipedia concept*, there are $10,456$ examples remained in *TestWiki*. The second derived data set is *TestWn*, which contains only concepts in the *extended WordNet* [25, 26]. *TestWn* has $8,625$ examples after dropping $3,375$ examples with concepts not in the *extended WordNet* from *TestAll*. The results of this experiment are shown in Tab. 6. We only report the best results achieved by the systems on different test sets. Our system still significantly outperforms other systems when compared on specific data sets.

6.3 Experimental Analysis

In this section, we give several experimental analyses on our system. These analyses provide a deeper understanding of different aspects on our system.

6.3.1 Contributions of the Components in our System

In this experiment, we show the contribution of the components in our overall algorithm (see Fig. 1). We use different data sets in this experiments to study the variation in behavior of our system. The first data set is the *TestWiki* in the previous experiment. This test set contains $10,456$ examples with *Wikipedia concepts*. These are the remained examples after dropping *non-Wikipedia concepts* from $12,000$ examples in the *TestAll* test set. The second test set contain only $1,544$ concept pairs with at least one *non-Wikipedia* concept in each. In other words, the second test set is the complement of the *TestWiki* test set with respect to the *TestAll* test set. The third test set is the *TestAll* test set. The results are shown in Tab. 7.

Contributions of the components in our system			
System	Results on $10,456$ *Wikipedia examples*	Results on $1,544$ *non-Wikipedia examples*	Overall results on $12,000$ examples
Baseline	88.79	31.22	81.38
w/o Inference	88.79	44.88	83.14
with Inference	**90.92**	**45.40**	**85.07**

Table 7: Contributions of the components in our system evaluated on three types of data. The first column shows results only on examples with concepts that appear in Wikipedia. The second column shows results on examples having concepts that do not appear in Wikipedia, and thus exhibits the power of our method to extend outside Wikipedia. The third column shows results on the overall data set, that consists of around 13% of the concepts pairs that are outside Wikipedia. The *Baseline* system is our local relation classifier that uses neither the approach of finding replacements for *non-Wikipedia concepts* nor the inference model. The baseline on *non-Wikipedia examples* is computed by assuming sibling relation all the time. By adding the component of finding replacements for *non-Wikipedia concepts*, we have the *w/o Inference* system.

In Tab. 7, the *Baseline* system is our local relation classifier that uses neither the component finding replacements for *non-Wikipedia concept* nor the constraint-based inference model. The baseline on *non-Wikipedia examples* is computed by assuming sibling relation all the time. In the *w/o Inference* row, we evaluate our system with the contribution of the component finding replacements for *non-Wikipedia concepts*, but inference model. The result on $10,456$ *Wikipedia examples* remains the same because, all of concepts in these examples are in Wikipedia, the system does not need to find replacement concepts. The significant improvement in the results on $1,544$ *non-Wikipedia examples* shows the effectiveness of our approach in finding replacements for *non-Wikipedia concepts.* Overall, we obtain almost 2% of improvement in accuracy after adding the finding replacement component. The third experiment is carry out by evaluating our overall algorithm in Fig. 1 on all test sets. The inference process significantly improve the results of our system on $10,456$ *Wikipedia examples.* It also gains improvements on the *non-Wikipedia examples* so that in overall, by adding the constraint-based inference component, the system improves $\sim 2\%$ in accuracy. Overall, our system achieves significant improvement reaching to 85.07% accuracy in overall by adding in all components, compared to 81.38% accuracy gained by the *Baseline* system.

6.3.2 *Contribution of Related Concepts in Inference*

In our inference process, we used related concepts added to form relational constraints in concept networks. The relational constraints will then enforce concept networks to eliminate violated ones and pick the best concept network available. From our inference model, we make two following claims: (1) The more relevant to the input concepts the related concepts are, the better performance the system gets, and (2) the more related concepts added to the inference process, the better performance the system gets.

To prove the first claim, we use the original data of forty semantic classes (see Sec. 6.1) to provide *gold related concepts* to the inference process as discussed in Sec. 5.3. Our experiment shows that by using this approach, the performance of our best system reaches to 86.52% in accuracy on the *TestAll* test set, compared to 85.07% accuracy obtained when using another knowledge source to provide related concepts. For the second claim, we evaluate our system with several numbers of related concepts added to the inference process. In this experiment, we use both the *gold related concepts* provided by the original dataset and the *YAGO related concepts* extracted from YAGO ontology (see Sec. 5.3) for the inference process.

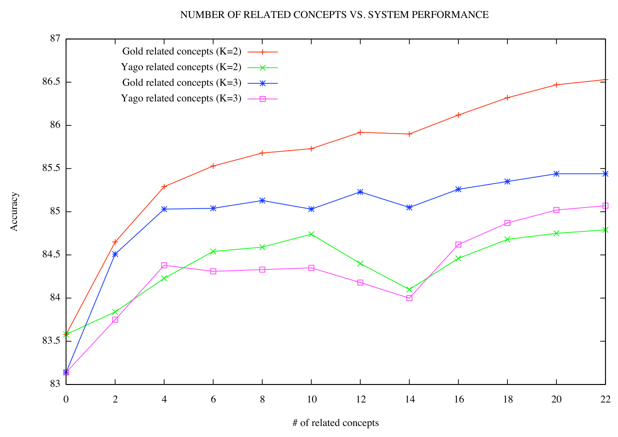

Figure 6: Relation between our system performance and the number of related concepts used in the constraint-based inference model.

Fig. 6 shows the improvement in the performance of our system with respect to the quality of the related concepts used and also the number of related concepts. This experiment is done on the *TestAll* test set. It is shown clearly that the related concepts from gold data outperform the related concepts extract from the other source. Due to the quality of the related concepts extracted from YAGO ontology, sometimes, adding more concepts hurts the performance. We use $K = 2$ and $K = 3$ in this experiment[7]. The related concepts include the ancestors, siblings, and children of an input concept. For simplicity, in this experiment we only report the total number of related concepts extracted for two input concepts of an example.

6.3.3 *System's Output Examples*

Table 8 shows some output examples of our overall system in Fig. 1 (see column *withInference*). We also present the

[7]K is the number of levels to go up in the Wikipedia category system to extract features for input concept pairs.

No.	*X*	*Y*	True Label	LocalClassifier Prediction	Replacement Needed?		withInference Prediction
					X	*Y*	
1	city	lisbon	$\mathbf{X} \leftarrow \mathbf{Y}$	$\mathbf{X} \leftarrow \mathbf{Y}$	(no)	(no)	$\mathbf{X} \leftarrow \mathbf{Y}$
2	*bartlomiej strobel*	painter	$\mathbf{X} \rightarrow \mathbf{Y}$	$\mathbf{X} \rightarrow \mathbf{Y}$	drew struzan	(no)	$\mathbf{X} \rightarrow \mathbf{Y}$
3	taiwan	singapore	$\mathbf{X} \leftrightarrow \mathbf{Y}$	$X \leftarrow Y$	(no)	(no)	$\mathbf{X} \leftrightarrow \mathbf{Y}$
4	jean ingres	*harald slott-moller*	$\mathbf{X} \leftrightarrow \mathbf{Y}$	$X \nleftrightarrow Y$	(no)	george inness	$\mathbf{X} \leftrightarrow \mathbf{Y}$
5	*southern herald*	newspaper	$\mathbf{X} \rightarrow \mathbf{Y}$	$X \leftrightarrow Y$	liberty	(no)	$\mathbf{X} \rightarrow \mathbf{Y}$
6	somalia	hurricane	$\mathbf{X} \nleftrightarrow \mathbf{Y}$	$\mathbf{X} \nleftrightarrow \mathbf{Y}$	(no)	(no)	$X \leftrightarrow Y$

Table 8: Relationship prediction examples from *TestAll* test set. Concepts in *italic font* are not in Wikipedia, and they are replaced by other *Wikipedia concepts* as shown.

predictions made by the system without inference component (*LocalClassifier*). Concepts which are not in Wikipedia will be replaced by *Wikipedia concepts* (*Replacement Needed?*). Example #1 and #2 show two concept pairs correctly classified by both *LocalClassifier* and *with Inference*. Especially, in #2, that concept *bartlomiej strobel* is replaced with *drew struzan* helps the systems make correct predictions. In examples #3, *LocalClassifier* make incorrect predictions, but *withInference* remedies this. Example #4 and #5 show two cases where two *non-Wikipedia concepts* are correctly replaces with two corresponding *Wikipedia concepts*, but *LocalClassifier* still makes incorrect prediction. However, *withInference*, with its power of internal reasoning using relational constraints, proves to be successful. Example #6 is an example where *LocalClassifier* makes correct prediction, but *withInference* does not. This is probably because of bad related concepts extracted for the two input concepts.

7. CONCLUSION AND FUTURE WORK

Textual inference problems often arise in data and knowledge management and 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, using a small number of annotated examples. We have shown results that are significantly better than existing methods, showing that our approach generalizes well across semantic classes and handles well concepts that are not mentioned in Wikipedia. The key lesson from the success of our approach has to do with our combined learning and global inference approach. Our machine learning approach makes use of Wikipedia resource, but views it as an open and noisy resource; this is augmented with a novel use of a constraint-based inference model that allows us to make these decisions more robust. The key technical step needed to improve our method further is to better generate concepts that are related to the target concepts, so that our global inference method becomes even more effective. Along with this, future research will include an evaluation in the context of supporting textual inference applications.

8. REFERENCES

[1] M. Banko, M. Cafarella, M. Soderland, M. Broadhead, and O. Etzioni. Open information extraction from the web. In *IJCAI*, 2007.

[2] M. Banko and O. Etzioni. The tradeoffs between open and traditional relation extraction. In *ACL-HLT*, 2008.

[3] R. Braz, R. Girju, V. Punyakanok, D. Roth, and M. Sammons. An inference model for semantic entailment in natural language. In *AAAI*, 2005.

[4] M. Chang, D. Goldwasser, D. Roth, and Y. Tu. Unsupervised constraint driven learning for transliteration discovery. In *NAACL*, 2009.

[5] M. Chang, L. Ratinov, and D. Roth. Constraints as prior knowledge. In *ICML Workshop on Prior Knowledge for Text and Language Processing*, 2008.

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

[7] D. Davidov and A. Rappoport. Unsupervised discovery of generic relationships using pattern clusters and its evaluation by automatically generated SAT analogy questions. In *ACL-HLT*, 2008.

[8] P. Denis and J. Baldridge. Joint determination of anaphoricity and coreference resolution using integer programming. In *NAACL*, 2007.

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

[10] Y. Freund and R. E. Schapire. Large margin classification using the perceptron algorithm. *Machine Learning*, 1999.

[11] E. Gabrilovich and S. Markovitch. Computing semantic relatedness using wikipedia-based explicit semantic analysis. In *IJCAI*, 2007.

[12] I. Guyon and A. Elisseeff. An introduction to variable and feature selection. *JMLR*, 2003.

[13] A. Haghighi, A. Ng, and C. Manning. Robust textual inference via graph matching. In *HLT-EMNLP*, 2005.

[14] B. Maccartney and C. D. Manning. Modeling semantic containment and exclusion in natural language inference. In *COLING*, 2008.

[15] M. Mohler and R. Mihalcea. Text-to-text semantic similarity for automatic short answer grading. In *EACL*, 2009.

[16] M. Paşca. Organizing and searching the world wide web of facts step two: Harnessing the wisdom of the crowds. In *WWW*, 2007.

[17] M. Paşca and B. Van Durme. Weakly-supervised acquisition of open-domain classes and class attributes from web documents and query logs. In *ACL-HLT*, 2008.

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

[19] V. Punyakanok, D. Roth, and W. Yih. The necessity of syntactic parsing for semantic role labeling. In *IJCAI*, 2005.

[20] V. Punyakanok, D. Roth, W. Yih, and D. Zimak. Learning and inference over constrained output. In *IJCAI*, 2005.

[21] D. Roth and W. Yih. A linear programming formulation for global inference in natural language tasks. In *CoNLL*, 2004.

[22] D. Roth and W. Yih. Integer linear programming inference for conditional random fields. In *ICML*, 2005.

[23] L. Sarmento, V. Jijkuon, M. de Rijke, and E. Oliveira. "more like these": growing entity classes from seeds. In *CIKM*, 2007.

[24] S. Sekine. On-demand information extraction. In *ACL*, 2006.

[25] R. Snow, D. Jurafsky, and A. Y. Ng. Learning syntactic patterns for automatic hypernym discovery. In *NIPS*, 2005.

[26] R. Snow, D. Jurafsky, and A. Y. Ng. Semantic taxonomy induction from heterogenous evidence. In *ACL*, 2006.

[27] F. M. Suchanek, G. Kasneci, and G. Weikum. Yago: A Core of Semantic Knowledge. In *WWW*, 2007.