RoDEO: Reasoning over Dependencies Extracted Online

Reda Siblini and Leila Kosseim

CLaC Laboratory
Department of Computer Science and Software Engineering
Concordia University
1400 de Maisonneuve Blvd. West
Montreal, Quebec, Canada H3G 1M8

R_Sibl@encs.concordia.ca, Kosseim@encs.concordia.ca

Abstract

The web is the largest available corpus, which could be enormously valuable to many natural language processing applications. However it is becoming very difficult to identify relevant information from the web. We present a system for querying dependency tree collocations from the web. We show its usefulness in identifying relevant information by evaluating its accuracy in the task of extracting classes of named entities. The task achieved a general accuracy of 70%.

1. Introduction

(McEnery and Wilson, 2001) has described corpus linguistics as the study of language based on examples of 'real life' language use. And where can we find more examples of 'real life' language use than the web? With the growth of the World Wide Web, more and more researchers have been used it in their study of language. Collocations are a rich source of information that could be very useful in many natural language processing tasks. The richness of collocations is attributed to its relationship with meaning. This relationship has been noticed by many researchers, and probably two of the most quoted ones are (Firth, 1957) "you shall know the word by the company it keeps" and (Harris, 1968) distributional hypothesis that "words with similar meaning tend to appear together". The term collocation by itself has more than one definitions in the literature: (Firth, 1957) described it as "habitual" word combinations, (Manning and Sch'eutze, 1999) as some conventional way of saying things, (Bartsch, 2004) as a frequently recurrent, relatively fixed syntagmatic, combinations of two or more words, and (Evert, 2005) as word combination whose semantic and/or syntactic properties cannot be fully predicted from those of its components, and which therefore has to be listed in a lexicon.

Although the importance of collocation is obvious, it is not always easy to collect collocation information for words that are usually unavailable in a typical corpus; for example collocations of a specific proper noun. As the web contains the largest repository of text, it is seen as a solution of such a problem. However, using the web as a corpus poses its own set of challenges. In this paper we describe a system for extracting, representing, and querying collocations from the web, and we attempt to respond to some of those challenges posed by the web.

In the remainder of this paper, we first review related work, then describe a system for extracting collocations online titled RoDEO: for Reasoning over Dependencies Extracted Online. And then we evaluate it on the task of extracting classes of named entities.

2. Related Work

In this section we will present some of the related work in the literature that extract, represent, and query collocation information from the web.

(Kilgarriff et al., 2004)'s sketch engine, is a corpus tool that creates "Word sketches", or one-page summaries of a word's grammatical and collocational behavior, from a corpus, in addition to other functionalities. The corpus query language and search is based on regular expression. The tool also makes uses of a web service that produces corpora from the web, but the time needed to build a corpus from the web could varies between minutes and hours.

(Resnik and Elkiss, 2005) described the Linguist's Search Engine (LSE), a system that enables language researchers to query the web for examples based on syntactic and lexical criteria. The system allows users to create queries by example: by supplying a sample sentence the system parse the sentence, and find similar structure in the user selected corpus. To use the web as a corpus the user must supply a web query to a search engine from within the system, the system will extract, parse and index sentences from the web search results and the result can be used by the user as a new corpus. The user has to wait between minutes and hours for this process to be completed, however the user can begin querying the results as soon as they are collected. The indexing and search of the collected data is based on a method for querying XML Data by tree structures. It should be noted that parse trees data are not similar to the typical semi-structured data that is usually stored and queried in XML, as phrases structure are usually highly recursive.

(Renouf et al., 2007) presented WebCorp as a tool that helps corpus linguists in retrieving linguistic output from the web. The request for linguistic information is translated and feed to web search engine, the returned documents will be processed, and the concordance results are returned to the user. The author presented some of the current WebCorp problems, such as the performance issues, the need of a grammatical and better collocational analysis, in addition to a more sophisticated pattern matching.

Most of these systems are similar to the system we are presenting in this paper; however the differences and the advantage of our system will be evident in the following sections.

3. Extracting Collocations Online

Annotated corpora contain linguistic information that enables linguists to accurately search for the specific occurrences of linguistic information. The usefulness of the corpus is increased with the level of annotation or linguistic information explicitly present in it, and with the expressiveness of the search query formalism utilized to investigate these data.

Most of the related work presented in the previous section annotates the utilized corpus, or its extracted part, with syntactic structure. The syntactic structure used follows a certain syntactic theory, nevertheless most researches, justifiably, try not to select a theory specific structure. Two main categories of syntactic structure are usually described: phrase structure and dependency structure. Phrase structure combines words or phrases into syntactic categories or constituent parts. Dependency structure, on the other hand, is a representation of relationships between words. (Hays, 1964) define a dependency relationship as a binary relationship between a word called head and another called modifier. For example the sentence: *"Tom drives a car"* can be represented by the following set of dependency relationships: (Tom—subject—drives), (a—determiner—car), and (car—object—drives). Figure 1 shows a graphical

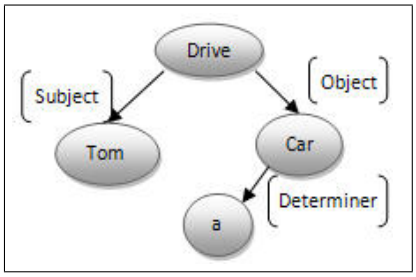

Figure 1: An Example Dependency Tree

representation of this dependency tree. The expressiveness of the dependency formalism is usually related to the different types of syntactic relations used to characterize the words relationships.

We prefer the use of dependency structure over phrasal structure since they represent syntactic relations explicitly; however these relationships are more implicit in a phrasal structure. For example, the actuality that *"Tom"* is the subject of *"drives"* in the above example can only be uttered in the phrasal structure as a sentence having noun phrase headed by the noun *"Tom"*, and related to a verb phrases headed by the verb *"drives"*. As such using a dependency structure allows us to directly represent and query syntactic relations. However selecting the syntactic structure is not the main issue in querying or representing syntactic collocations. The main issue is related to how to represent the created structure and how to query it. In the next section we are going to present the preferred annotation representation structure that will allows us, in addition to the incorporation of syntactic structure, the inclusion of semantic information. The selected representation is a scalable and tractable structure that will enable expressive query formalism in addition to an advanced reasoning capabilities based on the availability of syntactic and semantic information.

3.1. Corpora Representation

Before describing the preferred corpora representation, we will revisit the collocation definition in order to specify our preference, which correspond to our syntactic structure preference that we have described in the previous section.

(Lin, 1998b) defines a collocation as a dependency relationship between two words that occurs significantly more frequently than by chance. He also proposed a method for extracting dependency collocations from text corpora. His method involves the parsing of a corpus using the Minipar (Lin, 1998a) parser, and storing the resulted dependencies into a database. By using dependency frequencies and mutual information he separated collocations from dependency triples that occurred by coincidence. We would like to extend (Lin, 1998a)'s collocation definition to a dependency relationship between a word and one or more dependency trees. We refer to this type of collocations as Dependency Tree Collocation or (DTC). This extension of collocation would allow us to filter the extracted dependencies into very specific ones. This restriction is required as the most frequent collocations vary by context, and in a given context we are not usually talking about all the possibilities of a word collocating with another, but to a restricted subset that the context is related to. As such, this extension will allow us the restriction of the returned collocations to a selected context.

A dependency collocation between two words could be represented in the following linear form: [Word1] Dependency1—[Word2]; where Word1 collocates with Word2 via the dependency relationship Dependency1. On the other hand, a dependency collocation between a word and a dependency tree could be represented in the following form [Word1] Dependency1—[Word2] Dependency2—[Word3] Dependency3—[Word4]... or graphically as in Figure 2.

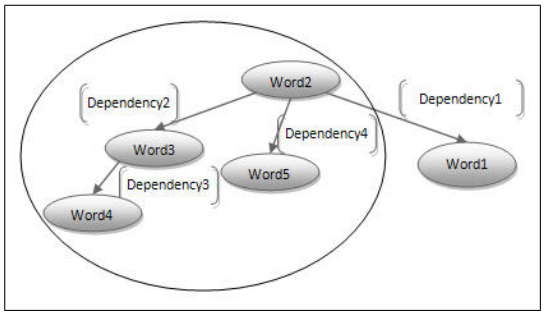

Figure 2: A Graphical Representation of a Dependency Tree Collocation

One of (Lin, 1998b)'s example of a collocation as a dependency relationship between two words is [Word]Object—[Drink], or finding the word that col-

locates with the word *"Drink"* through an object dependency relationship. We can extend this example to find [Word]Object—(Drink)—Subject(Child) or the word that collocates through an object dependency relationship with the dependency tree consisting of the verb *"Drink"* having a subject dependency relationship with the word *"Child"*. Although the top ranked collocation of (Lin, 1998b)'s example is *"Tap Water"* the top words for our DTC is *"Milk"*. In this sense we have restricted the retrieved collocations to the instances of the verb that collocate with a specific object. We could have different restriction types depending on the formulated DTC query.

Multiple method have been proposed in the literature to represent and query corpora annotated information, from a simple regular expression match such as the sketch engine of (Kilgarriff et al., 2004), to a more complex semi-structured way as in the linguistic Search Engine case of (Resnik and Elkiss, 2005). Nevertheless, what we are presenting here is our preferred annotation representation structure that will allows us, not only to represent syntactic structure, but also to include semantic information in a scalable, tractable structure. This structure will enable expressive query formalism, in addition to an advanced reasoning capabilities based on the availability of syntactic and semantic information. Our representation is based on a decidable logic representation of the DTC structure. Since dependency trees could be seen as a semantic network, using description logic that is equipped with a formal logic-based semantics fits exactly our needs. Especially when such a logic is a tractable, decidable subset of predicate logic, and has several well studied theorem provers to query it.

Description logic (DL) is a logical formalism for defining concepts and their relations (Terminologies) specifying properties of individuals (Assertions). Lately, description logics became the foundation of the semantic web. The Semantic web is a collaborative effort led by World Wide Web consortium (W3C) that provides a framework for making World Wide Web content processable by machines (Berners-Lee, 1998). W3C endorsed the web ontology language (OWL) as the language for the semantic web (Dean et al., 2004). OWL is a semantic markup language for defining and instantiating web ontologies, and it is based on description logic. It is a vocabulary extension of RDF (the Resource Description Framework), derived from the DAML+OIL (DARPA Agent Markup Language and Ontology Interchange Language), and based on XML (Extensible Markup Language). An OWL ontology may include descriptions of classes (or concepts), properties (relations between a main class called domain and another called range), and the classes instances (or individuals).

We have selected OWL-DL as the structure for representing corpora annotation, it is the subset of OWL supporting a decidable (SHOIN(D)) description logic (Horrocks and Patel-Schneider, 2004). The intuition behind this selection is the effortless mapping between a dependency relation and an OWL-DL property between two classes (the head of dependency relation and its modifier). In addition the lemma and part of speech information of each word are mapped to classes. For example, the DTC: (Drinks_Verb)—Subject(Child_Noun) will be mapped to the following OWL descriptions:

- "Subject" as an OWL property having as domain the class "Drink" and as range the class "Child" (note that the lemma of the word is used to describe the class) .
- "Drink" is subclass of the class "Verb" and "Child" is a subclass of the class "Noun".
- In addition to indexed instance of the class "Drink", such as: "Drinks_1", and an indexed instance of the class "Child" such as "Child_1".

The indexing is needed to relate sentence instances. The end result could be illustrated graphically as in figure 3.In this graph rectangles represent classes, ovals represent properties, and double brackets represent instances.

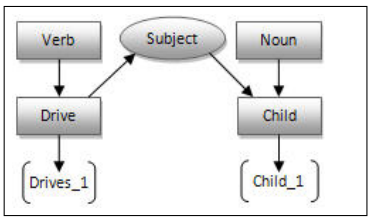

Figure 3: A Graphical Representation of a Dependency Tree Collocation ontology

This representation has a lot of advantages, some of these advantages:

1. Availability of well studied, powerful query formalism.
2. Possibility of applying rules to the created knowledge base using backward or forward chaining.
3. Ability to integrate multiple knowledge bases, some of which might include general semantic information.
4. The existence of well studied reasoners that can be used to answer queries over the created knowledge base schema and instances.
5. OWL is a standard meant to be used for the semantic web, in which semantics of information and services on the web are defined to be used and exchanged. As such, most of the available reasoner services are built for large scalability.

Investigating in details all of these advantages are beyond the scope of this paper. Nevertheless we are going to focus on the first advantage: a powerful query formalism and related answering mechanism, which will be introduced in the next section.

3.2. Query Formalism

The reasoner that we have selected for the query answering is the RACER reasoner (Haarslev and Moller, 2003). RACER (an acronym for Renamed ABox and Concept Expression Reasoner) is a reasoner that implements tableau calculus for description logic (DL) and supports the web

ontology languages DAML+OIL, RDF, and OWL. nRQL (new RACER Query language) is an expressive DL-query language. An nRQL query consists of a query head and a query body. For example, the query (retrieve (?x) (?x Noun)) has the head (?x) and the body (?x Noun). It returns all *"Nouns"* from the ontology which is queried. A detailed description of nRQL is given in (Haarslev et al., 2004). Although nRQL is closely related to Horn logic query languages such as Datalog, it has been argued (Wessel and Moller, 2006) that nRQL is a more general query language framework that provides more flexibility and options for extensions than Datalog. In addition, it provides an optimized implementation by bounding the variables in a query to explicitly mentioned instances in the knowledge base.

The DTC query could be represented in nRQL, which enables us to formulate complex conjunctive queries in order to retrieve a specific DTC from the created dependency ontology structure. Then we can use the RACER conjunctive query answering to prove the nRQL, over the created DT ontology, and return the corresponding results.

For example, if we are looking for the dependency tree collocation [Noun]Subject—[Drive]—Object[Ball], that is the nouns that are subjects of the verb *"Drive"* and having *"Ball"* as its object, we can formulate it as the following nRQL query: (Retrieve (?x)(AND(?y ?x Subject)(?y ?z Object)(?y Drive)(?x Noun)(?z Ball)).

Although we are using nRQL as the query formalism of choice, other query languages could be easily used such as the OWL-QL (Fikes et al., 2004) query language of the semantic web.

Using RACER we can then run an nRQL query and return related Dependency Tree Collocations from the web corpora that would be presented as an OWL-DL ontology. Representing the whole World Wide Web as a well structured description-logic knowledge representation would be the ideal solution to query DTCs, however as it is not feasible for us to do so, we are going provide an alternative: a method to generate sentences from a DTC query, which will enable us to created related web queries and retrieve very specific and related documents that will represent a corpus related to the DTC in question. This method will be described in details in the next section.

3.3. From DTC to Sentences

Most web search engines provide an unstructured query language to query the web. Transforming a DTC conjunctive query to a web search query puts forward its own set of challenges. Using the DTC conjunctive query content words as web query keywords may return millions of documents. However, not all the returned documents are related to the dependencies collocation that we are looking for, but barely documents containing the supplied keywords. To narrow down the returned results to the dependency relationship that we are looking for, we formulate the web query as a specific search phrase. A search phrase is a sequence of words that must co-occur together.

For the DTC example query above: (Retrieve (?x)(AND(?y ?x Subject)(?y ?z Object)(?y Drive)(?x Noun)(?z Ball)). The content words of this DTC query are: "Drive" and "Ball". When we used the Google API to search for documents having the keywords "Drive" and "Ball", we obtained about 39 million documents. When we searched for the search phrase *"Drive.Ball"*, we found 167,000 documents. Still this search phrase is not the search phrase that returns the narrowed down documents to the DTC above. In addition, concatenating the content words from the DTC does not always match what we are looking for. So we need a way to know how we usually write sentences that match the dependency tree we are looking for.

Our solution to this problem is the creation of a dependency tree corpus from a subset of the open American National Corpus (ANC), which would act as a representation of the grammatical structure of the sentences in terms of dependency trees. The corpus is represented as an OWL-DL knowledge base in the same way described in section 3.1. We then use the reasoner over the create knowledge base to query for the subclasses of the DTC conjunctive query. For the example above the subclasses of DTC query is (Retrieve (?x)(AND(?y ?x Subject)(?y ?z Object)(?y Verb)(?x Noun)(?z Noun)). Running this query over the created tree ontology will help in the creation of search phrases that correspond to the DTC conjunctive query. The dependency tree corpus only contains dependency trees of grammatical categories and their relationships, but does not contain the actual words of sentences. This knowledge base is meant to be a representation of various grammatical phenomena, where each dependency tree represents a typical sentence in a text, and is ranked by its length and by the number of non-content words it contains. The reasoning behind the ranking is that the more non-content words in a search phrase the more specific the documents that will be returned from the search engine.

For example, one of the search phrases that matched the DTC query above in the created dependency tree knowledge base, and ranked high as having a length of 6 words, where 3 of them are non-content words, is [Determiner.Noun-Subject.Pronoun.Verb.Determiner.Noun-Object].

This search phrase contains a set of related grammatical categories, such as determiner, verb, noun.., which act as placeholders. Non-Content categories are then replaced by a non-content word, for example *"determiner"* could be replaced by *"a"*, and *"pronoun"* with *"that"*. Content words (Noun, Verb) are replaced by the content words of the DTC query; so *"Verb"* would be replaced by *"Drive"* in this example. The rest of the categories will be replaced by search query wildcards. So one of the resulted web search phrase from this example that is specific to the Google API is: [a.*.that.drive.*.ball]. This query with Google returned only 69 documents.

For each DTC conjunctive query we could automatically generate hundreds of similar search phrase queries. Then using a search engine we run the created search phrase queries. The resulted top 10 documents of each search query are downloaded, stripped from HTML, and a regular expression match is performed in order to extract the complete sentences that conform to the search phrase query. For the example query above some of top returned

sentences are:
"A golfer that drives a golf ball that..." —Rank = 8
"He has a swing that drives the ball..." —Rank = 6
"A batter that drives a ball forward..." —Rank = 4
"a force that drives your ball back..." —Rank = 2
...
The above ranks are the rank of the document returned by the search engine.

3.4. From Sentences to DTCs

In order to be able to execute the DTC conjunctive query over the extracted web sentences from the previous section, we need first to transform the extracted sentence to a dependency tree. To do so we use the Minipar dependency parser. But before transforming the extracted sentences to dependency trees, we first filter out interrogative, negated, and conditional sentences, in order to represent only factual and positive sentences. Minipar represents the grammar as a network of nodes representing grammatical categories and the links representing grammatical relationships. Minipar's lexicon is derived from WordNet. In addition to proper names, it contains about 130k entries in base forms. Lexical ambiguities are handled by the parser. Minipar constructs all possible parses for an input sentence, and outputs a single parse tree with the highest ranking. Parsing is based on a manually constructed grammar and is guided by statistical information obtained by parsing a 1GB corpus with Minipar. The resulted dependency trees will then be mapped into OWL-DL as we showed in the section 3.1. All the resulted sentences will be mapped into one knowledge base. Figure 4 shows a graphical representation of a part

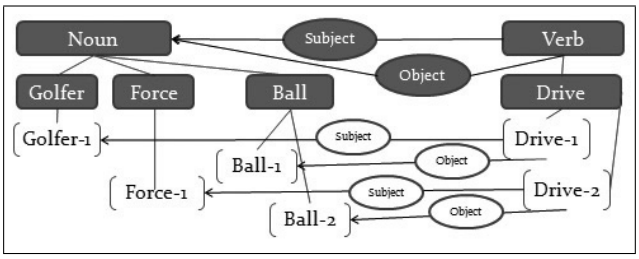

Figure 4: RoDEO Ontology Example

of the dependency ontology created by RoDEO for some of the returned sentences of the example query above. In this graph rectangles represent classes, ovals represent properties, and double brackets represent instances.

3.5. Reasoning over Dependencies Extracted Online

In order to achieve high precision is answering the DTC nRQL query, the query and the resulted OWL-DL knowledge base will be supplied to the reasoner RACER. RACER will try to prove the query over the created knowledge base and return any instances that conform to it.
For example, some of the nouns that were returned by RoDEO for the nRQL query (Retrieve (?x)(AND(?y ?x Subject)(?y ?z Object)(?y Drive)(?x Noun)(?z Ball)) are: *"Golfer, swing, batter, stroke..."*. Notice that introducing the object *"Ball"* that is modifying the verb *"Drive"* returns subjects that are related to one sense of *"Drive"*, that is *"Hit very hard, as by swinging a bat horizontally"*, which is the only sense related to the word *"Ball"*.
The reasoning however is not only in nRQL answering, but also in the possibility of reasoning over word semantic relationship. Such semantic information could be easily integrating from general available ontologies. For example, if a sports ontology has been added to the created ontology of the previous example, the object *"Ball"* will also match to instances under its subclasses, such as the subclasses: *"baseball, basketball, or even a marble..."*
Inferring new information from the available one is also possible by running rules over the created knowledge base. For example, if we create and executed a rule saying that: "if an object is round and is hit or thrown or kicked in games then it is a subclass of the class ball", we will be able to reason over the new inferred information that has been added to the ontology by this rule.
The following section will describe our results in using RoDEO on the task of extracting classes of named entities.

4. Evaluation Application: Extracting classes of Named Entities

In this section we evaluate the extracted collocations using an application of the RoDEO system. The application that we developed over RoDEO is the application of extracting classes of named entities.

4.1. Named Entity Recognition

Named Entity Recognition (NER) as described by the Message Understanding Conferences (MUC)-7 (Chinchor, 1998) is the task consisting of identifying and classifying entities that are considered to belong to one of the following classes: person, location, organization, temporal entities and numeric quantities. Different approaches have been introduced to deal with NER, however two approaches are mainly adopted. The first uses resources, such as gazetteers, and handcrafted rules to match the term to the resources, and the other use machine learning techniques on a tagged corpus in order to learn a set of patterns or to train some sort of a supervised learning algorithm such as the work of (Bikel et al., 1999).

4.2. Extracting classes of Named Entities with RoDEO

Our aim is to automatically extract the most specific class(es) of a selected named entity. For example, we need the ability to extract the class that *"Paul Krugman"* belongs to, in this case a general class would be a "person", but a more specific one would be a "columnist". To accommodate the utmost coverage in selecting fine-grained classes of named entities, many researchers have used the web. Most of the techniques used rely on a set of pattern, and the main difference between one technique and the other is usually the type of patterns used. Some used text patterns such as the work of (Etzioni et al., 2005), other used wrapper or HTML patterns such as the work of (Nadeau et al., 2006). (Etzioni et al., 2005) KNOWITALL system aims to automate the extraction of instances of classes such as the names of scientist from the web by using a set of patterns.

We will be building upon (Etzioni et al., 2005)'s work, however instead of using a set of text patterns over the web, we will be using dependency tree conjunctive query patterns, and instead of learning instances of classes, we will be learning classes of instances. By creating the dependency tree collocation patterns we can then use the RoDEO system to extract dependencies that conform to the selected dependency pattern from the web in order to extract classes of named entities.

One of the dependency tree collocation patterns that we will introduce here is the **predicate noun** pattern. Grammatically, a predicate-noun follows a form of the verb *to be*, like in the sentence: *"Margaret Thatcher was the Prime Minister"*. In this example *"Margaret Thatcher"* is the subject of verb *to be* and *"Prime Minister"* is its predicate noun. As a result, it would be an appropriate pattern to extract classes of named entities. The predicate-noun DTC query pattern is of the form: (Retrieve (?x)(VerbtoBe(?z) AND Predicate(?z ?x) AND Subject(?z ?y) AND Named-Entity(?y))), that is a pattern that looks for ?x having a dependency relationship of type predicate with ?z an instance of verb to be that is having a subject relationship to the named entity in question.

Another pattern is the **appositive pattern**. Appositive is a word that usually describes another word, as in: *"Rudy Giuliani, New York City Mayor, is..."*. The noun *"Mayor"* is an appositive to *"Rudy Giuliani"*. The corresponding dependency tree collocation pattern would be: ((Retrieve (?x) (Noun(?y) AND Named-Entity(?x) AND Appositive(?x ?y)))).

We have also derived other patterns from (Hearst, 1992)'s lexico-syntactic patterns, such as:

1. NP such as NP, (or/and) NP.
   Where NP stands for noun phrase. Example: *Columnist, such as Paul Krugman*.

2. Such NP as NP, (or/and) NP.
   Example: *Work by such columnist as Paul Krugman, and Paul Romer*.

3. NP, or other NP.
   Example: *Paul Krugman, or any other columnist in the N.Y. Times*.

4. NP, and other NP.
   Example: *Read Paul Krugman and other economists and healthcare experts....*

The first of Hearst's patterns, for example is translated to the following DTC query: ((Retrieve (?x)((?x Noun) AND Named-Entity(?y) AND Modify(?x such-as) AND Pcomp-n(such-as ?y)))), where Pcomp-n stands for a nominal complement of a preposition.

Using RoDEO, a dependency tree collocation pattern will return a list of nouns that conforms to a selected DTC query from the web. First we replace the named entity into the Named-Entity DTC patterns, then using RoDEO we run the created DTC pattern that will collect related dependencies and store them into an ontology. We then Match the DTC query using the Reasoner over the created ontology and we count the resulted dependency tree collocations. If the resulted collocation is less than a certain threshold t, we run the next DTC patterns until we have enough collocations returned. The returned collocations are all ranked by Google's document ranking. In order to select the most appropriate noun corresponding to the term in question, we first cluster the returned collocations, and then rank the clusters by collocation frequency. The clustering is simply based on the semantic relatedness of nouns as defined by (Miller, 1995)'s WordNet's hypernym relations. In addition, we filter out the nouns that belong to certain classes that could not represent a named entity class, such as the nouns belonging to the following hypernyms: *"feeling, psychological, status..."*.

Table 1 shows an example of the top 5 returned classes with their ranks for the named entity *"Al Franken"*, grouped by clusters.

Cluster	Class	Rank
Cluster 1	Guy, man, adult male, male...	25
Cluster 1	Author, communicator, person...	12
Cluster 1	Candidate, politician, politico...	7
Cluster 1	Comedian, performer, artist...	5
Cluster 2	Dog, canine, carnivore...	2

Table 1: Named Entity Classes Returned By RoDEO For "Al Franken"

Only the classes of the cluster with the highest frequency are considered as possible types of the named entity, so in this example only cluster 1 is returned.

4.3. Evaluation Results

As we classify named entities into very specific types, we evaluated the application of extracting classes of named entities over a set of 1019 named entities extracted from a shared online database of structured knowledge called FreeBase (Bollacker et al., 2007). FreeBase contains named entities with their general and specific types. For example, according to FreeBase, the named entity: *"Al Franken"* belongs to the following types *"Person, author, writer, and actor"*. The evaluation scoring has been done by comparing our extracted types to the FreeBase types. As the system returned classes do not have to exactly match the FreeBase types, we used the WordNet::Similarity (Pedersen et al., 2004) Path Length method in comparing two types. The path length method is a simple node-counting scheme, which returns a relatedness score between two concepts. The score is inversely proportional to the number of nodes along the shortest path between the synsets in WordNet. The shortest possible path occurs when the two synsets are the same, in which case the length is 1. If the compared types had a relatedness score that is over a threshold t, t=0.21, we considered that it as correct. The threshold has been selected after manually comparing a set of 50 classes. For example, if the returned class is an *"Actor"* for a named entity, and its FreeBase corresponding type is an *"Artist"*, the WordNet::Similarity Path Length method returns a relatedness of 0.25 for the two concepts. As such we assume that the returned class is correct. We have evaluated a total of 1019 named entities. The total number of

different FreeBase types, that these entities belong to is 69 types. The total number of classes returned by our system for the 1019 named entity is 678 types. That shows that our system is returning far more specific results than the FreeBase types. For example, the *"Athlete"* FreeBase type has been matched to *"Blocker, bowler, boxer, cornerback, cricketer, footballer, keeper, receiver, scorer, skater, swimmer, tackle..."*.
To compute the accuracy of the extracted classes of a single named entity we use the following:

$$\text{Accuracy} = \frac{\text{Number of correct types}}{\text{Total number of types}}$$

The total accuracy of the system is computed as the average of the accuracy for all the evaluated named entities.
Overall, the application achieved an accuracy of 0.7. Table 2 shows some of the accuracy results grouped by types and sub-types. For example, for the high level *"Person"* type the accuracy achieved is up to 0.87, whereas the type *"Company"* achieved an accuracy of 0.62. The person type can be subdivided into several subtypes, for example the *"Actor"* type achieved an accuracy of 0.78.

Types	Accuracy	Subtypes	Accuracy
Person	0.87	...	...
		Actor	0.78
		Athlete	0.76
		Author	0.75
		...	...
		Publisher	0.14
		...	...
Company	0.62	...	...
		Airline	0.66
		Employer	0.34
		Owner	0.31
		...	...
		Chain	0.16
		...	...

Table 2: Sample Of The Evaluation Results

There are many related work in the named entity recognition and classification field; however most of the available work fall under the initially task set at the MUC conference for identifying and classifying named entities into five classes, which is much easier than classifying named entities into more fine grained classes. Most methods that classifies named entities into five classes achieved an accuracy of well above 90%. However, this has not been the case when classifying named entities into more fine grained classes. As such, we are going to focus our comparison to some of the approaches that classify named entities into more than just five classes. Table 3 shows a comparison of some of these approaches ordered by the number of classes they consider. (Cimiano and Staab, 2004)'s PANKOW system is a leixco-syntactic pattern based system that uses the web frequency to select the appropriate class from a set of 59 classes. The PANKOW system achieved an accuracy of 24.9%. (Nadeau, 2007)'s BaLIE system uses semi-supervised machine learning and the web to classify named entities into 100 classes. It achieved an accuracy of 57.4%. BaLIE creates large gazetteers of named entities, using a hand crafted HTML markup in web pages and a seed of named entities, and then uses a simple heuristic to identify and classify named entities. (Sekine, 2004)'s system achieved 72% by classifying named entities into 200 classes, however they used about 1,400 handcrafted rules and a dictionary of 130,000 instances that are classified into the 200 classes. The last system is (Alfonseca and Manandhar, 2002)'s system that adopted a vector space model having syntactic dependencies as vector features, and compared the named entity vector into the most similar vector. They had considered 1200 classes and achieved an accuracy of 17.39% using the verb/object dependencies as a feature.

Systems	Types	Accuracy
MUC	5	>90%
PANKOW	59	24.9%
BaLIE	100	57.4%
Sekine's tagger	200	72%
RoDEO	678	70%
Alfonseca's system	1200	17.39%

Table 3: Comparison Table

Although we are extracting a large number of fine grained classes, we are not classifying the named entities into these set of classes, but extracting the most frequent classes associated with each named entities. We notice from this comparison that RoDEO's accuracy is comparable to the system using hand crafted rules, although we are extracting a much larger number of classes.
While analyzing the system results, we noticed that some of the low scores are a result of the restriction that we have set on the RoDEO system regarding the total number of returned collocations. It seems that the number is too low, and increasing it would probably boost the overall system accuracy. In addition, the similarity path length method that was used for the scoring is not very adequate. For example the *"Chain"* concept relatedness score to the *"Company"* concept is 0.2, which is less than the threshold set. As such a *"Chain"* is not a treated as of type *"Company"*. At the same time, if we lower the threshold to less than 0.21, then concepts such as a *"Person* and a *"Set"* would be related.
It should be noted that comparing the results of extracting classes of named entity is also an issue by itself. As many techniques have been proposed for the ranking of named entity recognition and classification task, a recent survey of named entity recognition and classification systems (Nadeau and Sekine, 2007) has showed that a score of a simple example made of only five named entities, varied between 20 and 40 %, using three scoring techniques that have been used in the major conferences related to named entity recognition and classification.

5. Conclusion and Future Work

In this paper we have presented a system for extracting dependency tree collocations online, using a dependency

parser, an advanced representation based on description logic, and a reasoner. We showed the usefulness of such a system in extracting classes of named entities, and the usefulness of using the web as a corpus. Without the quantity of the text that the web provides, such an application would not have been possible.

In our future work, we plan to enhance the extraction of classes of named entities, mainly by increasing the threshold of returned collocations, and integrating the resulted ontology with a named entity types ontology which will automatically reason over these types without relying on WordNet Similarity. In addition, we would like to show the usefulness of the RoDEO system in other NLP applications, such as in semantic relations of noun compounds, in commonsense rule discovery, and in other applications.

6. References

E. Alfonseca and S. Manandhar. 2002. Extending a lexical ontology by a combination of distributional semantics signatures. *Proceedings of the 13th International Conference on Knowledge Engineering and Knowledge Management (EKAW 2002)*, pages 1–7.

S. Bartsch. 2004. *Structural and Functional Properties of Collocations in English.: a corpus study of lexical and pragmatic constraints on lexical co-occurrence*. Gunter Narr Verlag.

T. Berners-Lee. 1998. Semantic Web Road Map. *World Wide Web Consortium (W3C). http://www.w3.org/DesignIssues/Semantic.html*.

D.M. Bikel, R. Schwartz, and R.M. Weischedel. 1999. An Algorithm that Learns What's in a Name. *Machine Learning*.

K. Bollacker, R. Cook, and P. Tufts. 2007. Freebase: A shared database of structured general human knowledge. *Proceedings of the National Conference on Artificial Intelligence*, 22(2).

N. Chinchor. 1998. Muc-7 named entity task definition. *Proceedings of the 7th Message Understanding Conference (MUC-7)*.

P. Cimiano and S. Staab. 2004. Learning by googling. *ACM SIGKDD Explorations Newsletter*, 6(2):24–33.

M. Dean, G. Schreiber, et al. 2004. OWL Web Ontology Language Reference. *W3C Recommendation*, 10.

O. Etzioni, M. Cafarella, D. Downey, A.M. Popescu, T. Shaked, S. Soderland, D.S. Weld, and A. Yates. 2005. Unsupervised named-entity extraction from the web: An experimental study. *Artificial Intelligence*, 165(1):91–134.

Stefan Evert. 2005. *The Statistics of Word Cooccurrences Word Pairs and Collocations*. Ph.D. thesis, University of Stuttgart.

R. Fikes, P. Hayes, and I. Horrocks. 2004. OWL-QLa language for deductive query answering on the Semantic Web. *Web Semantics: Science, Services and Agents on the World Wide Web*, 2(1):19–29.

J.R. Firth. 1957. *Papers in linguistics 1934-1951*. Oxford University Press New York.

V. Haarslev and R. Moller. 2003. Racer: A core inference engine for the semantic web. *Proceedings of the 2nd International Workshop on Evaluation of Ontology-based Tools*, pages 27–36.

V. Haarslev, R. Moller, and M. Wessel. 2004. Querying the Semantic Web with Racer+ nRQL. *Proceedings of the KI-04 Workshop on Applications of Description Logics*.

Z.S. Harris. 1968. *Mathematical Structures of Language*. Interscience Publishers New York.

D.G. Hays. 1964. Dependency theory: A formalism and some observations. *Language*, 40(4):511–525.

M.A. Hearst. 1992. Automatic acquisition of hyponyms from large text corpora. *In Proceedings of the 14th International Conference on Computational Linguistics*.

I. Horrocks and P. Patel-Schneider. 2004. Reducing OWL entailment to description logic satisfiability. *Web Semantics: Science, Services and Agents on the World Wide Web*, 1(4):345–357.

A. Kilgarriff, P. Rychly, P. Smrz, and D. Tugwell. 2004. The Sketch Engine. *Proceedings of Euralex*, pages 105–116.

D. Lin. 1998a. Dependency-based evaluation of minipar. *Workshop on the Evaluation of Parsing Systems*, pages 317–330.

D. Lin. 1998b. Extracting collocations from text corpora. *First Workshop on Computational Terminology*, pages 57–63.

C.D. Manning and H. Sch'eutze. 1999. *Foundations of Statistical Natural Language Processing*. MIT Press.

T. McEnery and A. Wilson. 2001. *Corpus Linguistics*. Edinburgh University Press.

G.A. Miller. 1995. WordNet: A Lexical Database for English. *COMMUNICATIONS OF THE ACM*, 38(11):39.

D. Nadeau and S. Sekine. 2007. A survey of named entity recognition and classification. *Linguisticae Investigationes*.

D. Nadeau, P.D. Turney, and S. Matwin. 2006. Unsupervised named-entity recognition: Generating gazetteers and resolving ambiguity. *19th Canadian Conference on Artificial Intelligence*.

David Nadeau. 2007. *Semi-Supervised Named Entity Recognition: Learning to Recognize 100 Entity Types with Little Supervision*. Ph.D. thesis, University of Ottawa, November.

T. Pedersen, S. Patwardhan, and J. Michelizzi. 2004. WordNet::Similarity-Measuring the Relatedness of Concepts. *Proceedings of the Nineteenth National Conference on Artificial Intelligence (AAAI-04)*.

A. Renouf, A. Kehoe, and J. Banerjee. 2007. WebCorp: an integrated system for web text search. *Corpus Linguistics and the Web*.

P. Resnik and A. Elkiss. 2005. The Linguists Search Engine: An Overview. *Proceedings of the ACL 2005 on Interactive poster and demonstration sessions*, pages 33–36.

S. Sekine. 2004. Definition, dictionaries and tagger for Extended Named Entity Hierarchy. *Actes LREC*.

M. Wessel and R. Moller. 2006. A Flexible DL-based Architecture for Deductive Information Systems. *IJCAR Workshop on Empirically Successful Computerized Reasoning (ESCoR)*, pages 92–111.