Mining Semantic Structures from Syntactic Structures in Free Text Documents

Hamid Mousavi^1, Deirdre Kerr^2, Markus Iseli^3, Carlo Zaniolo^4

Technical Report #140005

^{1,4}*Computer Science Department, UCLA*

^{2,3}*CRESST, UCLA*

Los Angeles, USA

^{1}hmousavi@cs.ucla.edu

^{2}dkerr@cse.ucla.edu

^{3}iseli@cse.ucla.edu

^{4}zaniolo@cs.ucla.edu

Abstract— **The advances in the Web has risen many ambitious text-mining applications such as review or news summarization, essay grading, question answering, and semantic search. For many of such applications, statistical text-mining techniques are ineffective and provide very low recall, since they do not utilize morphological structures of the text. Thus, many approaches are now using deeper NLP-based techniques, by parsing the text and employing patterns to mine and analyze it. However, in addition to being noisy, parse trees and other similar structures contain many of the syntactical structures in the text. Analyzing such structures requires many complex patterns, which are very costly to make. To address this major issue, we present a weighted graph-based representation of text, called *TextGraph*, which provides the grammatical and semantic relations between words and terms in the text. TextGraphs are generated using a new text mining framework which is the main focus of this paper. Our framework, *SemScape*, uses a statistical parser to generate few of the most probable parse trees for each sentence and employs a novel two-step pattern-based technique to extract *candidate terms* and their grammatical relations from the parse trees. Additionally, SemScape presents a novel technique for resolving coreferences, accepts ontologies to generated more domain-specific TextGraphs, and provides a SPARQL-like query language and an optimized engine for semantically querying and mining TextGraphs.**

I. Introduction

A tremendous amount of publicly available data in the World Wide Web is in free text format. Systems such as online encyclopedias, online reviewing systems, news agencies, social networks, online publications, costumer complaint systems, blogs, etc. are constantly generating textual data. With this increase on the volume of the textual data, users are demanding for more advanced mechanisms for retrieving and accessing the data. Nowadays, people are willing to read short summaries of long articles or news rather than the entire text. They prefer to see the average rating for different features of a service or a product instead of going over all textual reviews of other costumers. They often want to know the hottest topics in the social networks or blogs without spending too much time reading uninteresting items. More importantly, advanced structured search [12], faceted search [25], question answering systems, and automatic personal assistants (e.g., *Siri* and *Google Now*) are getting more popular than traditional keyword-based searches.

Above applications, as well as many domain-specific ones, require a more effective approach for analyzing and mining text [48]. This approach should be able to represent the semantic of text in a more standard structured format, hide grammatical and syntactical features of the text, mine domain-specific text, handle multi-lingual text, and support ambiguities and exceptions in natural languages.

Current text mining approaches can be divided into two main categories: bag-of-words (also called statistical or machine learning) and NLP-based (or sometimes called parser-based) techniques. Since the former does not exploit morphological structures in the text, they are mostly incapable of addressing all mentioned requirements and usually need to use larger data sets and ignore less frequently mentioned information just to exclude some exceptions. On the other hand, NLP-based approaches parse the sentences in the text and convert them into tree-based structures, called *parse trees*. Parse trees contain [some of] morphological structures in the text in a more machine readable format, and thus provide a better structure for text analyzing. Although NLP-based approaches are much more resource-demanding than keyword-based ones, they are proven to be more effective in addressing the current text mining needs, specially once combined with statistical and machine learning techniques [35], [36], [38]. Moreover, the recent advances in distributed computing techniques also has hugely alleviated the time performance issue of NLP-based techniques [2].

Text mining through NLP-based techniques is often performed by employing some patterns on parse trees [30] (or similar structures [33]). Generating these patterns, either manually or by statistical patten learning techniques, is a challenging and costly task. Since a simple piece of information can be expressed in many different ways through natural languages, many patterns need to be created to extract that piece of information. Generating such patterns is both costly and time-consuming, which is mainly emanated from the fact that parse trees are still carrying various syntactical structures in the text. Thus, to mine from parse trees one should deal with these different structures with many complex patterns. Notice that, automatic pattern generation techniques are not any better, since they often need large training data sets (that often have to be created manually) and are not usually able to learn patterns for exception cases.

To address this issue and ease the process of text mining, we present a new and more expressive text representation, called *TextGraph*, that is much closer to the semantic of the text, and thus requires simpler and fewer patterns to be analyzed. As opposed to parse trees [30] and dependency trees [33], TextGraphs capture single- and multi-word (*candidate*) terms and grammatical connections between words and terms in the text. Moreover, by providing weights for the edges/links, TextGraph can better handle noises in text or parse trees and the ambiguity in text.

In this paper, we present a new text mining framework, called *SemScape*, which employs a pattern-based technique to generate TextGraphs from free text. SemScape uses statistical parsers [30], [8] to generate few most probable pares trees of the sentences, and then uses a small set of tree-based patterns to annotate the parse trees with some useful information called *MainParts*. MainParts carry up the hidden information in the leaves and lower branches of the parse trees to the upper non-terminal nodes. In this way, one need much simpler and more general patterns to mine the annotated trees, referred to as the *MainPart* (*MP*) Trees. Finally, SemScape uses another set of tree-based patterns over the MP Trees to extract grammatical relations between words and terms in the text to generate the TextGraphs. Generating TextGraphs from parse trees in above two steps has significantly reduced the number and the complexity of required patterns. Additionally, SemScape presents several contributions which are more specifically discussed bellow:

- We introduce an annotated parse tree called *MainPart tree* or *MP Tree* in which non-terminal nodes are annotated with important information resides in their branches. We also introduce a tree-based query language, called Tree-Domain (TD) Rules. TD rules can be used to query both parse trees and MP trees. Using MP trees, one can extract information from the parse trees with fewer and less complex patterns, which consequently eases the entire process of text-mining.
- Utilizing the MainPart trees, we propose a weighted graph representation of the text, called *TextGraph*, which hides many syntactical features of the text. TextGraphs are more expressive than tree-based structures, include multi-word (*candidate*) terms in the text, and are able to present ambiguity of the text through weighted edges.
- To be able to query and mine TextGraphs, SemScape provides a SPARQL-like query language. This query language, called *Graph-Domain* (*GD*), introduces few new features to simplify expressing queries on TextGraphs. We also present an optimization technique for searching graph patterns in TextGraphs in order to avoid several unnecessary join operations in regular SPARQL engines.

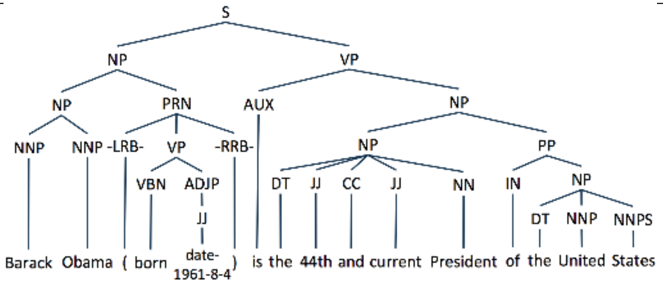

Fig. 1. Most probable parse tree for our running example.

- We propose a new Coreference Resolution technique to resolve pronouns and other references in the text. This is performed through a new component in SemScape called *Story Context* (*SC*) which uses a large body of contextual, taxonomical, and categorical information. SC also takes advantage of many syntactical patterns specifying possible or impossible resolutions. impossible resolutions significantly improve the quality of the final resolutions, which is completely overlooked in the current state-of-the-art.
- SemScape is also able to adapt with different domains by accepting an ontology. Once an ontology is fed to the framework, it generates TextGraphs with higher focus on the known terms and concept and eliminates many unrelated terms. This makes the framework capable of dealing with very noisy text data sets as shown in [38].

Since SemScape uses a pattern-based mining technique (with supports for syntactical exceptions in natural languages) to generate TextGraphs, it provides a natural way for incrementally improving the system by adding more rules to capture missing grammatical connection and exclude wrongly generated connections. Currently, all patterns mentioned in this work are created manually, however supervised or semi-supervised techniques can be used to create more of such patterns. The SemScape framework has been already used in several text mining applications [28], [35], [36], [37], [38] and is proven to be very effective.

II. Preparing Parse Trees

To prepare the text, we first partition it into its paragraphs and sentences and then simplify the sentences so that the parsing takes place more effectively and efficiently. To illustrate this process as well as other steps toward generating TextGraphs, we use the following example text throughout the paper:

Motivating Example: "*Barack Obama (born August 4, 1961) is the 44th and current President of the United States. He is the first African American to hold the office. Born in Honolulu, Hawaii, Pres. Obama is a graduate of Columbia University and Harvard Law School, where he was president of the Harvard Law Review.*"

As for the first step, SemScape finds terms and values in known formats such as dates, floating point numbers, url addresses, etc. and uses a uniform way to represent them

```
[-1]    (S
[0]       (NP
[0, 0]      (NP (NNP Barack_1) (NNP Obama_2))
[0, 1]      (PRN
[0, 1, 0]     (-LRB- (_3)
[0, 1, 1]     (VP (VBN born_4) (ADJP (JJ date-1961-8-4_5)))
[0, 1, 2]     (-RRB- )_6)))
[1]       (VP
[1, 0]      (AUX is_7)
[1, 1]      (NP
[1, 1, 0]     (NP
[1, 1, 0, 0]    (DT the_8)
[1, 1, 0, 1]    (JJ 44th_9)
[1, 1, 0, 2]    (CC and_10)
[1, 1, 0, 3]    (JJ current_11)
[1, 1, 0, 4]    (NN President_12))
[1, 1, 1]     (PP
[1, 1, 1, 0]    (IN of_13)
[1, 1, 1, 1]    (NP (DT the_14) (NNP United_15) (NNPS States_16))))))
```

Fig. 2. Most probable parse tree for our running example in parenthesized format with our addressing schema.

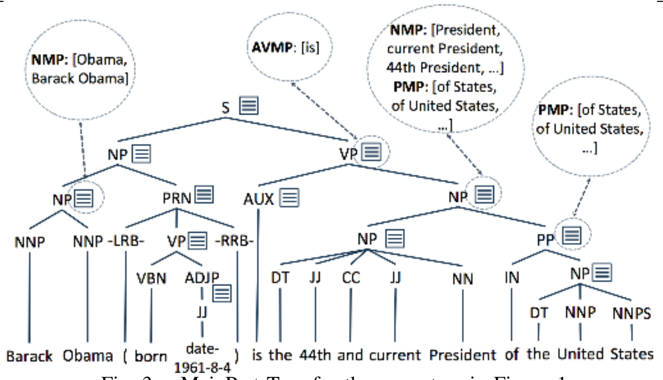
NMP: [Obama, Barack Obama]
AVMP: [is]
NMP: [President, current President, 44th President, ...] PMP: [of States, of United States, ...]
PMP: [of States, of United States, ...]
S
NP
VP
NP
PRN
AUX
NP
NNP
NNP
-LRB-
VP
-RRB-
NP
PP
VBN
ADJP
JJ
DT
JJ
CC
JJ
NN
IN
NP
DT
NNP
NNPS
Barack
Obama
(
born
date-1961-8-4
)
is
the
44th
and
current
President
of
the
United
States

Fig. 3. MainPart Tree for the parse tree in Figure 1.

(by eliminating all occurrences of the *period* character). For instance, in our running example date '*August 4, 1961*' is converted to a standard format ('*date-1961-8-4*'). Then, SemScape tags abbreviation terms (e.g. Pres., Mr., U.S., etc.) used in the text. After these simple steps, SemScape partitions the text into paragraphs considering the *NewLine* character as the delimiter. Paragraphs are important to SemScape since they specify the scope of the pronouns used in them as explained in Section V. Finally, SemScape uses *end of sentence* characters ('.', '?', '!', etc.) to extract the sentences.

Next, SemScape parses each sentence using a probabilistic parser (e.g. Charniak [8] and Stanford [30] parsers). For each sentence, we generate N_{pt} (>1) parse trees (PTs) using the parser. Having more than one PT will i) help us better deal with the inaccuracy and noisiness of the parsers in many cases, ii) increase the amount of extracted information, and iii) provide a better way for representing ambiguity in the text. For many cases, the first parse tree is not completely correct, so using the secondary parse trees may help improving the results. In some cases, more than one parse trees may be correct and using them helps generating more information as well as capturing possible ambiguity in text. One such parse tree for our motivating example is shown in Figure 1.

Each word in the generated parse trees is assigned an ID to make the system capable of uniquely addressing words in the text. This is required to avoid confusion among repeated words and more importantly, to preserve the order of words and terms in the TextGraphs. SemScape also uses a simple addressing scheme to address nodes in the tree (Figure 2). In this scheme, each node address contains its parent address plus its position in the ordered list of siblings.

III. MainPart Trees

Parse trees are much richer structures than the text. In fact they have been frequently used in different studies to improve the bag-of-words techniques for extracting information from text. However, they still suffer from two important issues that make their use challenging, and thus limited:

- The most important issue is that the structure of the parse trees is hugely dependent to the grammar and morphological structures in text. In other words, parse trees are still far from the semantic of the sentences. Thus, extracting information from such parse trees still requires dealing with such various syntactic structures.
- The second issue is that parse trees (as well as dependency trees) are only connecting words together. Multi-word terms (A.k.a. *Candidate Terms*) and their roles in the sentences are completely missing from these structures.

To address these issues, we propose a richer structure by annotating the non-terminal nodes in the parse trees with useful information about their underlying sub-trees as shown in Figure 3. For instance consider the left most NP in Figure 3. As shown in the figure, this noun phrase is representing either '*Barack Obama*' or '*Obama*'. These pieces of information will carried up to the upper nodes in the parse trees so other application will not need to search deep in the trees branches. These types of information are referred to as *Main Parts* (*MPs*) as they specify the main part of data in each branch. MPs may contain multi-word (candidate) terms as well, which addresses the second issue mentioned earlier. The annotated parse trees are referred to as *MainPart Tree*s or *MP Tree*s.

To extract MPs in the parse trees and assign them to their corresponding nodes, we use tree-based patterns/rules, which are also called Tree Domain (TD) rules. Take the NP at address $[1, 1, 0]$ in Figure 2 as an example. This NP, which specifies the object of the verb '*is*' in the sentence, contains the phrase '*the 44th and current President*'. The most important component (or part) of this phrase is obviously the term '*President*', which is referred to as a Noun MainPart (NMP) of the mentioned NP. A TD rule for extracting this NMP is shown below:

Rule 1.

```
RULE mainPartRule1 ('NMP') {
   PATTERN:   (NP *
                 (? |JJ |ADJP )
                 (? |CC )
                 (JJ |ADJP )
                 (NP |NN |NNS )
                 !* )
   RESULT:    < [−1], [3] >
   RESULT:    < [−1], [0] + [3] >
   RESULT:    < [−1], [2] + [3] >
}
```

```
[-1]  S ⇒ NMP: {Barack Obama, Obama}
[0]       NP ⇒ NMP: {Barack Obama, Obama}
[0, 0]        NP ⇒ NMP: {Barack Obama, Obama}
[0, 0, 0]         NNP ⇒ NMP: {Barack}
[0, 0, 1]         NNP ⇒ NMP: {Obama}
[0, 1]        PRN
[0, 1, 0]         -LRB-
[1, 1, 1]         VP ⇒ AVMP: {born}
[1, 1, 1, 0]          VBN ⇒ AVMP: {born}
[1, 1, 1, 1]          ADJP ⇒ NMP: {date-1961-8-4}
[1, 1, 1, 1, 0]           JJ ⇒ NMP: {date-1961-8-4}
[1, 1, 2]         -RRB-
[1]       VP ⇒ AVMP: {is}
[1, 0]        AUX ⇒ AVMP: {is},
                      PMP:{{of, the United States}, {of, States}, {of, United States}}
[1, 1]        NP ⇒ NMP: {President, current President,
                      44th President, President of United States, ...},
                      PMP: {{of, the United States}, {of, States}, {of, United States}}
[1, 1, 0]         NP ⇒ NMP: {President, current President, 44th President}
[1, 1, 0, 0]          DT ⇒ NMP: {the}
[1, 1, 0, 1]          JJ ⇒ NMP: {44th}
[1, 1, 0, 2]          CC ⇒ NMP: {and}
[1, 1, 0, 3]          JJ ⇒ NMP: {current}
[1, 1, 0, 4]          NN ⇒ NMP: {President}
[1, 1, 1]         PP ⇒ PMP:{{of, the United States}, {of, States}, {of, United States}}
[1, 1, 1, 0]          IN ⇒ NMP: {of}
[1, 1, 1, 1, 0]       NP ⇒ NMP: {States, United States}
[1, 1, 1, 1, 0, 0]        DT ⇒ NMP: {the}
[1, 1, 1, 1, 0, 1]        NNP ⇒ NMP: {United}
[1, 1, 1, 1, 0, 2]        NNPS ⇒ NMP: {States}
```

Fig. 4. MP Tree for the parse tree in Figure 2.

This rule consists of two parts: PATTERN and RESULT. PATTERN specifies a tree-like pattern for which we need to find matches in the PTs of the sentences in the text. The RESULT parts indicate how the MPs should be generated and to which node they should be assigned. We should add that PATTERNs are nested patterns and more expressive than regular expressions (or equivalently finite automata) [10]. This differentiates our work from most of the existing NLP-based techniques. Moreover, the tree-based format of our patterns makes them more readable and user friendly than regular expressions.

In Rule 1, PATTERN specifies noun phrases whose last four branches are i) an adjective or an adjective phrase (?|JJ|ADJP), ii) a conjunction (?|CC), iii) another adjective or adjective phrase (JJ|ADJP), and iv) a noun or a noun phrase (NP|NN|NNS). The first two branches are optional (indicated by a ?). From the parse tree shown in Figure 1 (and in parenthesized format in Figure 2), it is easy to see that '*44th and current President*' in our motivating example matches this pattern. If any match is found for this PATTERN, the first RESULT in Rule 1 adds the NMPs of the forth branch ('*President*' with address [3] in the pattern tree and address [1,1,0,4] in the matching tree) of the matching tree to the NMP list of its root (the node with address [-1] in the pattern tree and address [1,1,0] in the matching tree).

With its last two RESULTs, Rule 1 also suggests two multi-word terms, '*44th President*' and '*current President*'. This sort of terms are usually referred to as *Candidate Terms* in the literature, and can be directly used in *Name Entity Recognition* systems [39].

The MP Tree for our running example is depicted in Figure 4 in parenthesized format[1]. Currently, SemScape uses 135 TD rules (accessible at [5]) to generated the following four types of MP information:

- **Noun MainParts (NMPs):** As already explained, NMPs are defined for noun-related nodes (NP, NN, NNS, CD, JJ, $ADJP$, ...), and they indicate the actual term(s) represented by these node.
- **Active Verb MainParts (AVMPs):** A similar concept is used for the verbs-related non-terminal nodes (VP, VB, VBZ, VBD, VBN, ...) in the parse trees; however, since verbs have two forms, passive and active, we use two types of main-parts for verb-related nodes. Thus, AVMPs capture the active verbs of the verb-related nodes.
- **Passive Verb MainParts (PVMPs):** Similar to the previous case, PVMPs capture the passive verbs in verb-related non-terminal nodes. Passive verbs are of particular importance since they change the regular roles of the subject(s) and the object(s) in the sentences.
- **Preposition MainParts (PMPs):** The fourth MainPart set is for prepositions and preposition phrases. Both noun-related and verb-related nodes may contain PMPs. PMP of a node specifies a possible preposition for that node.

Generating PMPs is the most challenging among the four types of MainParts, since it often requires using the contextual knowledge, commonsense knowledge, and some sort of reasoning. For many cases, ambiguity in the text makes this task even more challenging. In the current implementation, we have only considered a very small set of patterns for generating PMPs and improving them are left for the future work.

Applying the mentioned 135 TD rules over the parse trees does not significantly increase the delay of generating MP tree comparing with the parse tree delay. However as shown next, MP trees require simpler and fewer patterns for being analyzed. Thus, they can serve as a good replacement of regular parse trees with a small effort in many existing text mining applications.

IV. TextGraphs

Although the MPTs generated in the previous subsection are richer structures than parse trees, they are still not completely suitable for representing semantics in the text. This is mainly because of the tree-based structure of parse trees, which limits the number of direct connections of terms to other terms to only one. This problem is alleviated to some extend by dependency trees [15] in which words can be used as non-terminal nodes as well. However, dependency trees are still limiting, since:

- Dependency trees do not still capture multi-word terms and their role in the sentences.
- They still inherit the limitation of tree-based representation. Representing semantics in a more standard way

[1]With this format, it is actually easier to grasp the idea of matching in our TD rules.

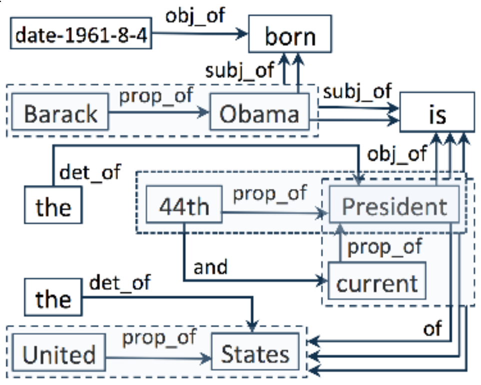

Fig. 5. Part of the TextGraph for our running example.

requires a more expressive structure, e.g. graph-base structure.

- They do not provide a systematic ways to represent the text ambiguity (e.g. confidences for the links).

To address these shortcomings, we introduce an even richer structured representation of text called *TextGraph*. *TextGraphs* are machine-friendly weighted graph structures, that represent grammatical connections between words and terms in the sentences, where terms are single- or multi-word phrases representing an entity or a concept. Each link in the TextGraphs is assigned a *confidence value* (weight) indicating SemScape's confidence on the correctness of the link and an *evidence count* indicating the frequency of observing the link.

A simplified TextGraph for the first sentence in our running example is shown in Figure 5. This graph connects words and terms to each others through grammatical links such as '*subj_of*', '*obj_of*', '*prop_of*', '*det_of*', etc. The complete list of link types in TextGraphs with their purposes is published in [38]. The graph also identifies multi-word (candidate) terms (as shown in dashed boxes) and their roles and links to other component of the sentences. For instance, the TextGraph contains two possible subjects for the verb '*is*' in sentence which are '*Barack Obama*' and '*Obama*'. These two at the same time play the subject role for the verb '*born*'.

With TextGraphs, more effective and efficient algorithms can be designed to extract knowledge from text by combining graph-based and statistical methods. This is mainly because:

- Representing the text with graph structures makes it possible to utilize many existing graph-based mining algorithms.
- TextGraphs already contain *candidate terms* in text that facilitates the process of many applications requiring these terms.
- They are closer to the semantic of the text, by providing meaningful terms and their grammatical connections.
- They are weighted which is very beneficial for dealing with ambiguity in natural languages and noisiness of the parsers.

These, in fact, hugely differentiate TextGraphs from their counterpart representation techniques such as parse trees and dependency trees. Later in next subsection, we also show how SemScape resolves pronouns and coreferences in the text to improve the TextGraphs. To generate the TextGraphs, we again take a pattern based approach to find grammatical relations between words and terms identified by the MP Trees. We refer to these relations as either *links* or *triples* throughout this paper. To extract links, we created more than 270 TD rules which are all available in [5]. The generated triples by these rules are later combined into the final TextGraph structure. An examples for such TD rules is shown bellow, This rule aims at capturing the '*subject of*' ('*subj_of*') links:

Rule 2.

```
RULE subjectToVerb {
   PATTERN:   (S
                 (NP )
                 (VP ))
   RESULT (FO1='NMP', FO3='AVMP', conf=.9):
              <[0], 'subj_of', [1]>
   RESULT (FO1='NMP', FO3='PVMP', conf=.9):
              <[0], 'pobj_of', [1]>
}
```

The PATTERN in Rule 2 specifies a pattern in which a Noun Phrase (*NP*) is followed by a Verb Phrase (*VP*). This is the most general form of subject-to-verb link structure in parse trees. Similar to MP rules, Rule 2 indicates that for the matching trees, the NMPs of the noun phrase (NP) should be connected to the active verb main-part (AVMP) of the verb phrase (VP) to generate a *subj_of* link with confidence 0.9. Moreover, the NMPs of the noun phrase (NP) should be connected to the passive verb main-part (PVMP) of the verb phrase (VP) as a '*passive_object_of*' (*pobj_of*) link. For our running example, this rule captures links such as <*Obama*, *subj_of*, *is*> and <*Barack Obama*, *subj_of*, *is*>.

Note that, with the assist of MP information, this single rule can catch most of the *subj_of* and *pobj_of* links in different sentences without needing to know their lower level structure under nodes NP and VP. This is actually one of the most important gains in the SemScape framework, and dramatically decreases the number of required patterns/rules as well as simplifying the patterns required for any text mining application.

Each generated triple has a confidence (indicated by keyword '*conf*' in Rule 2) showing SemScape's confidence on the correctness of the link. If the same link is generated from different rules or from different MP trees, SemScape increases its correctness confidence as discussed in Subsection IV-B. After applying all rules to the MPTs of a sentence and generating the triples, we combine them into the final TextGraph (e.g. Figure 5). That is each sentence is converted into a separate TextGraph. In section V, we show how SemScape improves the textGraph of each sentence by resolving its pronouns and coreferences with terms in the same or previous sentences. Next we discuss SemScape's approach for capturing syntactic exceptions in natural languages to improve the TextGraphs.

A. *Support for Exceptions*

Syntactic exceptions are the inseparable part of any natural languages. Although capturing exceptions can significantly enhance the quality of the text mining approaches, most of existing approaches do not provide an easy-to-use technique to handle exceptions. On the other hand, finding patterns with no exceptions in natural languages is a very tedious task, and a general pattern should be split down into many smaller patterns too avoid some exception cases. To simplify this process without needing to split our general patterns (e.g. Rule 2), SemScape uses patterns with negative confidence to specify exceptions and remove many of the incorrectly generated triples from the TextGraphs. Exception rules are considered superior to regular rules. That is if the same triple is extracted multiple times by different rules over the MP trees of the same sentence, and one of the extracted triples has a negative confidence, the triples with positive confidence will be eliminated.

To better illustrate this technique, consider the sentence "*In the woods are trees*". Since the sentence is in inverted form, which is not as common as the normal form, the parsers may not be able to recognize the structure correctly. For instance for the mentioned sentence, only one parse trees from the first three suggested parse trees by Stanford parser is correctly capturing the inverted form. Thus, our patterns may generate incorrect information from the incorrect parse trees. (<*trees*, *obj_of*, *are*> in our case). To eliminate this incorrect information, SemScape uses the following pattern:

```
———————————— Rule 3. ————————————
RULE subjectToVerb(Inverted) {
   PATTERN:   (SINV
                 (PP )
                 (VP )
                 (NP ))
   RESULT (FO1='NMP', FO3='AVMP', conf=-.9):
                 <[2], 'obj_of', [1]>
   RESULT (FO1='NMP', FO3='AVMP', conf=.9):
                 <[2], 'subj_of', [1]>
}
```

This pattern matches the correct parse tree and generated a triple with negative confidence (<*trees*, *obj_of*, *are*>) as well as the correct triple (<*trees*, *subj_of*, *are*>). Using the negative-weighted triple, SemScape can eliminate incorrect triples generated from wrong parse trees and improve the final TextGraph for the sentence.

B. *Combining Confidence Value*

As already mention, every triple in TextGraph is assigned a *confidence* value. Since the same triple may be generated more than once (from different rules or different parse trees), we need to combine their confidence value c. Similar to [32], the only assumption for the combination process is that evidences of the same piece of information are independent from each other. Thus, if a piece of information has been generated twice by different rules or from different MainPart Trees, once with confidence $c_1 \geqslant 0$, and once with $c_2 \geqslant 0$, we combine the confidence to $c = 1-(1-c_1)\ (1-c_2)= c_1+(1-c_1)c_2$. This new confidence is higher than both c_1 and c_2 which indicates the link's correctness probability is now higher. For each triple, we also count the number of time it has been generated and refer to it as the evidence frequency or count (e). We should note that, if one of the confidence values are negative for a particular triple (specifying an exception), we eliminate all the same triples with positive confidence as explained in previous subsection.

C. *Enriching TextGraphs with Ontologies*

Another important feature of SemScape is the ability to adapt an ontology and provide more related *candidate terms* with respect to the specified ontology. The Ontology here can be both domain-specific or domain independent in OWL. For simplicity, one can also specify a list of concepts instead of Ontology (e.g. the list of all subjects in Wikipedia). There are two main advantages of this feature which are discussed next.

The first advantage is to better control the volume of generated Noun MainParts. To understand how, we should note that for complex noun phrases (NPs) there might be several possible candidate terms. Not all of these candidate terms are useful or meaningful. Therefore, suggesting all of them as MPs may lead to a very large set of candidate terms which lowers the efficiency of the system. To prevent this problem, SemScape is made capable of utilizing an ontology, say O. Using O, SemScape only generates candidates terms that either i) contain less than three words, ii) are part of an existing concepts in O, or iii) contain a concept from O. In most simple cases this generates all possible candidate terms; however for many long noun phrases, this helps us reduce the size of the TextGraphs.

The second advantage of incorporating an ontology in the framework is to allow domain-specific applications to better utilize the framework. In a similar way, SemScape is also able to recognize *Named Enmities* [39] in the provided text which is not a trivial task in other approaches. It can also spot more related parts of the text with respect to the ontology which consequently improves system's robustness on dealing with noisy corpora. This feature is discussed in greater detail in [38].

D. *Graph Domain Patterns*

SemScape provides a graph-based query language, called *Graph Domain* or *GD* rules, to let users and applications mine the TextGraph. Although the format of GD rules is very similar to that of SPARQL [9], its implementation is somehow different as explained in Section VI. In GD rules, we also introduce few extended features for simplifying the information mining process from TextGraphs. These features are introduced later in this subsection. Besides the external applications that can benefit from GD rules, SemScape uses GD rules for two purposes. The first one is completing and improving the TextGraphs using few GD rules. These patterns are often much easier to be expressed by GD rules than TD rules. The second purpose is to perform Coreference resolution which is the topic of the next section in which we provide some examples of GD rules. Readers are also referred to [5], [38] and [37] for more examples of such rules.

We should add that GD rules are usually considered as in batch of patterns aiming at mining certain types of information. That is SemScape may be fed with sets of GD rules for different tasks (and applications). For instance as shown in [38], we fed the system with rules for generating ontological links in an automatic ontology generation system. Similar idea is used in [36] to generate structured information from free text. Thus, once the GD rules are specified by an application, SemScape will apply them on all TextGraphs of the provided text, combine the resulted tuples, and report them back to the application.

In addition to the features supported by SPARQL, GD rules introduce the following features to ease the process of TextGraphs mining[2]:

- Each rule may have multiple SELECT clauses to allow generating multiple pieces of information from same patterns.
- One may use keyword 'NEG' before the SELECT keyword to specifying exceptions similar to TD rules in subsection IV-A.
- One may use keyword 'NOT' before any triple in the WHERE clauses to indicate the absence of some links in the pattern, which requires a more complex expression in SPARQL.

V. Coreference Resolution

In textual documents, many pronouns and references are used to refer to other terms (concepts). For instance in our running example, the pronoun '*he*' in the second sentence is a pronoun referring to '*Barack Obama*' and '*Pres. Obama*' is a reference for '*Barack Obama*'. Resolving these types of coreferences, called *Coreference Resolution*, is a very challenging task since it requires a huge amount of contextual knowledge, commonsense knowledge, and in many cases complex and ad hoc inferencing techniques [34]. The ambiguity of the natural languages also aggravates this issue.

In SemScape, we propose a new technique to resolve coreferences through the *Store Context* (*SC*) component. At its highest level, SC recognizes characters (also known as *mentions*) in the text, learns contextual information about them from different resources, and uses this contextual information and a novel pattern-based technique to match characters to each other and resolve coreferences in the text. These steps are explained next in this section.

A. Recognizing Characters

The first step to construct the SC's structure is to recognize possible characters or mentions. This is a relatively easy task for us since TextGraphs already provide all the candidate terms (Noun MainParts) in text. We use these candidate terms as the characters of [the story of] the text. However, some characters in the text are more important (due to their role) than the others, and as a result they are more probable to be referred with pronouns or other references. To determine the importance of a character, each character is assigned an evidence count and a confidence weight. Whenever the same character is encountered in different roles (relations), we increase its evidence count by one and its confidence as explained in Section IV-B. We should mention that at this stage each occurrence of the same candidate term is considered as a separate character.

[2]Readers are referred to [37] for examples on each new feature.

B. Mining Characters Context

After generating the list of characters, we gather information about some of their important and distinguishing attributes or properties. These properties are sometimes called *agreement* properties. Currently, we consider seven properties: *isMale* (if it is a male or a female), *isPerson* (if it is a person or not), *isOrganization* (if it is an organization or not), *isLocation* (if it is an geographical location or not), *isAnimal* (if it is an animal or not), *isObject* (if it is a thing or not), and *isPlural* (if it is plural word or not). Other properties could be added to this set to improve the final resolutions results. However, we found these the most useful and differentiating properties. We should add that most of the exiting works in this area only consider person, gender, and number as their agreement properties [26], [31], [16], [24].

For each property, SC uses different sources of information to estimate their value, which ranges between -1 and 1. Value 1 means 100% confidence that the property holds and value -1 means 100% confidence it does not (value 0 indicates no information about the property). As opposed to most similar approaches which use only true/false values for characters properties [26], [31], [16], [24], SC uses probability of being true or being false to better deal with uncertainty .

To evaluate the mentioned properties, we use the rich structured knowledge provided by Wikipedia's categorical information. For instance if one of the ancestor of a character is the '*Category:People*' category in Wikipedia, we set its *isPerson* property to 1. The same approach is used for *isOrganization*, *isLocation*, *isAnimal*, and *isObject* properties. This technique can be used for many potential properties that one may want to add to our initial list of properties. However, the main drawback of this technique is that for many characters, there is either no equivalent title in Wikipedia or no categorical information. Thus, we use the following heuristics to mine more contextual information on each character:

- We use VerbNet [29] which for each verb v specifies its possible subject or object as either an organization, a person, an animal, or an object. For instance, in the sentence '*The agent was killed in a terrorist attack.*', due to the meaning of the verb '*killed*', '*agent*' is probably a person or an animal.
- For *isPlural*, we use the POS tags generated from the parser as well as some TD rules (e.g., terms containing the word '*and*' or '*or*' are considered plural).

- For *isMale*, we use lists of masculine (e.g., waiter, king, etc.) and feminine (e.g., waitress, queen, etc.) terms and lists of male or female proper first names.
- For *isPerson*, we use some POS tag information (e.g., NNP), our male and female first names lists, as well as some TD rules (e.g., any term *renaming* the words 'who' or 'whom' is a person).
- We also use a list of animal names to add more evidence to *isAnimal*.
- For *isObject*, if there is evidence that a term is not a person and is not an animal, we increase the confidence that it is an object. Any term *renaming* the word 'which' is also an object.

Combining Potentially Referencing Characters: Another new technique to improve our understanding about the values of the seven properties for different characters is to use taxonomical relations namely *type_of* relations between the characters. These relations can be generated using our OntoHarvester system [38] and Hyponym information in WordNet [47]. The key idea is that if character γ_1 is *type_of* character γ_2 with confidence c, then γ_2 may be used as a coreference for γ_1 (in other words, γ_1 may be referred to as γ_2) with confidence c. For instance, since '*algebraic equation*' is *type_of* '*equation*', after the first time '*algebraic equation*' is mentioned in text, it can be referenced with the '*equation*' for the rest of text. This essentially means that γ_1 can inherit the properties of γ_2 and vice versa. In order to do so, for each property f of γ_1, we update its confidence value, $\gamma_1.f$, as follows:

$$\gamma_1.f = \begin{cases} \gamma_1.f + (1-\gamma_1.f)c\gamma_2.f & \text{if } \gamma_1.f \times \gamma_2.f \geqslant 0 \\ (\gamma_1.f + c\gamma_2.f)/2 & \text{if } \gamma_1.f \times \gamma_2.f < 0 \end{cases}$$

The idea of combining the confidence values is essentially the same as in Section IV-B if properties' values have the same signs. If they have different signs, we simply take a weighted average on their values. By propagating the values of the properties for potentially coreference characters, we ease our later resolution technique which is based on the similarity (or *agreement*) of the properties of the characters.

C. Finding Patterns for Coreferences

Although in general resolving coreferences only based on morphological structures in text is not feasible, there are few cases for which these structures may indicate a resolution. For instance consider the sentence "*Bob relieved himself telling him the truth.*". Clearly from the structure of the sentence, one can tell the reflexive pronoun'*himself*' refers to '*Bob*' in this sentence. Here the pattern is that if the object of a verb is a reflexive pronoun, it always refers to the subject of the same verb. To capture such a pattern, we use the following GD rule:

Rule 4.

```
SELECT ( ?2 'CoRef' ?1 )
WHERE {
    ?1 'subj_of' ?3.
    ?2 'obj_of' ?3.
    FILTER (regex(?2, '^itself^|^herself^|^himself^...', 'i'))
}
```

In addition to reflexive pronouns, we use *predictive nominative* constructs, first exploited by [41], *appositive* and *role appositive* constructs [24], and *relative pronouns* construct [42]. As can be seen, there are only very few such constructs that explicitly specify a resolution. However, there are many cases that a construct explicitly indicates two characters can NOT refer to each other (be each others resolution). For instance, in our earlier example, '*him*' can refer to neither '*Bob*' nor '*himself*'. We refer to such cases as *impossible resolutions*. In general, the object and subject of most verbs can not be each other's resolutions unless the object is a reflexive pronoun. This pattern is specified by the following GD rule in SemScape:

Rule 5.

```
SELECT ( ?2 'NoCoRef' ?1 )
SELECT ( ?1 'NoCoRef' ?2 )
WHERE {
    ?1 'subj_of' ?3.
    ?2 'obj_of' ?3.
    FILTER NOT (regex(?3, '^be^|^become^|^remain^...', 'i'))
    FILTER NOT (regex(?2, '^itself^|^herself^|^himself^...', 'i'))
}
```

Some other obvious constructs indicating negative or impossible resolutions are terms connected through a preposition, terms for which one is part of the other, and terms connected with conjunctions such as '*and*', '*or*', '*except*', etc. Currently, SemScape uses 64 GD rules (available at [5]) to extract possible and impossible resolutions between characters from the morphological information in text. For possible resolutions, we combine the characters as explained in the previous subsection. Inheriting the property's value is even more useful when one of the characters is a pronoun, since most of the mentioned seven properties for pronouns are easily inferrable. The impossible resolutions, on the other hand, are used in the next subsection where we explain how SemScape performs the final character resolution.

D. Resolving Characters

Say that a character γ_1 from a sentence s needs to be resolved. We calculate the similarity (see next paragraph) between γ_1 in s and all other non-pronoun characters in s that are not an impossible resolution for γ_1. Our studies shows that filtering by impossible resolution reduces the search space by more than half for each sentence. Similarity values larger than a predefined threshold are reported as resolutions. If no value exceeds the threshold, resolution search for γ_1 is continued among characters in the previous same-paragraph sentences, iteratively. In this way, we take the *recency* into account. If γ_1 is a pronoun and no resolution is found for it in its paragraph, the search will be stopped; this should be a rare case. If γ_1 is not a pronoun and it is not resolved yet, sentences of the previous paragraph are also checked until a resolution is found or until the beginning of the document is reached. In other words, the scope for a pronoun's resolution is only its

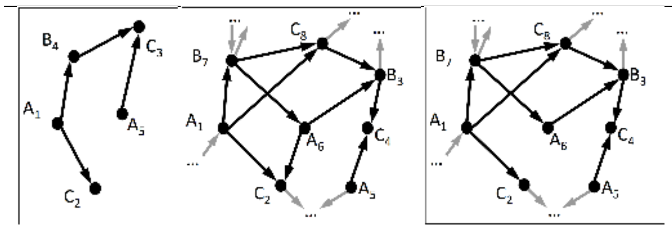

Fig. 6. From left to right (a) A GD query (q), (b) a TextGraph containing q (tg), and (c) a TextGraph not containing q (tg').

paragraph in SemScape, while the scope for other coreferences is the entire document.

Characters Similarity Measurements: To compute the similarity of two characters γ_1 and γ_2, we define the distance for property f as $\delta_f=(\gamma_1.f-\gamma_2.f)/2$ ($-1\leqslant \delta_f \leqslant 1$) and average for property f as $\mu_f=(\gamma_1.f+\gamma_2.f)/2$ ($-1\leqslant \mu_f \leqslant 1$) respectively. Thus, we compute the similarity of characters γ_1 and γ_2 by:

$$1-(\sum_{f\in F} d_f)/|F|$$

where F is the set of all properties (i.e. $|F|=7$ in our case) and d_f is the dissimilarity of γ_1 and γ_2 for property f which is defined as:

$$d_f=\frac{|\delta_f|}{|\mu_f|+1}$$

It is easy to see that $0 \leqslant d_f \leqslant 1$. The above formula indicates that more difference ($|\delta_f|$) essentially means more dissimilarity, especially when the average is smaller (e.g., the dissimilarity of properties with values -.2 and .2 is more than the dissimilarity of properties with values .5 and .9 even though their distances are the same).

VI. Graph Matching Optimization

As you may have already noticed, TextGraphs can be presented in RDF-like triple format, and thus queried by SPARQL. However, there is a subtle difference between TextGraphs and RDF graphs. As the TextGraphs are the graph representation of text, they may include several nodes with the same name or label while nodes in RDF must have unique names. For instance in the TextGraph in Figure 5, there are two nodes with label '*the*'. This makes querying the TextGraphs using SPARQL inefficient as explained next.

Consider the GD pattern/query (q) in Figure 6(a). Five nodes with three different labels (A, B, and C) are connected through four links in this query. As mentioned earlier, to differentiate among nodes with the same label, SemScape assigns an ID to each node (e.g. 'A_1', 'A_5', and 'A_6' in Figure 6(b) or '*the_8*' and '*the_14*' in Figure 5). To ease our discussions, we assume all edges have the same label, say '*link*'. Now consider two TextGraphs tg and tg' in Figures 5(b) and 5(c). As can be seen, although both tg and tg' contain all the individual edges of the query graph q, only tg contains a subgraph matching q. To find such matches, q can be expressed with the following SPARQL query:

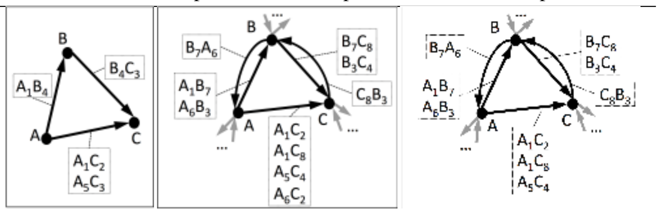

Fig. 7. From left to right: Collapsed version of (a) q (q_c), (b) tg (tg_c), and (c) tg' (tg'_c).

Rule 6.

```
SELECT *
WHERE {
    ?1 'link' ?2 .
    ?1 'link' ?4 .
    ?4 'link' ?3 .
    ?5 'link' ?3 .
    FILTER (strStarts(?1, 'A_'))
    FILTER (strStarts(?2, 'C_'))
    FILTER (strStarts(?3, 'C_'))
    FILTER (strStarts(?4, 'B_'))
    FILTER (strStarts(?5, 'A_'))
}
```

Although this approach perfectly works for GD query q using any SPARQL engine, it is very inefficient and slow, since it can not take advantage of any smart indexing techniques. Notice that the engine normally needs to traverse all triples for each where clause in the form of <?1 'link' ?2 .> and then filter them based on the specified FILTER commands. The same phenomenon can happen in other RDF resources, however it is much more frequent in TextGraphs due to too many same-label nodes.

To take advantage of the same-label nodes in our query engine, we first ignore all the IDs assigned by the SemScape to nodes and collapse both query graphs and TextGraphs by combining same-label nodes into a single node. The collapsed version of the examples in Figure 6 is depicted in Figure 7. As shown in Figure 7, for every link in the collapsed version, we store all the associated links in the actual graph for later retrieval. Now the collapsed version of query q, called q_c (Figure 7(a)), can be answered through the following SPARQL query:

Rule 7.

```
SELECT *
WHERE  {
    'A' 'link' 'C' .
    'A' 'link' 'B' .
    'B' 'link' 'C' .
}
```

Answering collapsed queries (e.g. Rule 7.) over collapsed TextGraphs is much faster than regular queries (e.g. Rule 6.) over original TextGraphs. This is due to two main reasons: i) the collapsed queries are smaller in the number of edges which consequently reduce the number of join operations to answer them by a SPARQL engine and ii) there are more literal nodes and less variable nodes in the collapsed query graphs, which makes a better use of indexing optimizations.

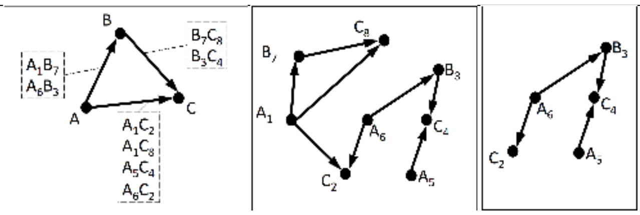

Fig. 8. From left to right: (a) the subgraph of tg_c matching q_c, (b) the expanded graph of part a, and (c) the subgraph of tg matching q.

However, finding a match for a collapsed query on a collapsed TextGraph does not necessarily mean that the actual query matches the actual TextGraphs (e.g. q_c matches tg'_c but q does not match tg'.) A match, here, only indicates that all edges in the query graph have at least one corresponding edge in the TextGraph. This check significantly reduces the search space by eliminating many TextGraphs in which one or more of the specified edges in q is missing. For the remaining cases, we use the following verification algorithm to verify that the matching edges construct the same graph structure specified by q.

Verification: After finding the matches for the collapsed queries on the collapsed TextGraphs, we expand the matching edges to those in the actual edges. For instance, consider TextGraph tg and its collapsed version tg_c (Figure 7(b)). Running the collapsed query q_c over tg_c will return the subgraph shown in Figure 8(a). Once we expand this subgraph to its corresponding subgraph in tg, we create a graph in which each edge matches at least one of the edges in the original query graph q. This subgrapg which is shown in Figure 8(b) is referred to as tg_m.

Now, the goal is to verify that the query graph q matches with tg_m or part of it. To this end, any sub-graph matching technique can be utilized. Fortunately, since both nodes and links in TextGraphs are labeled and the queries are mostly small graphs, the problem is much simpler than the general subgraph matching (subgraph isomorphism) problem which is NP-Complete [19]. In our current implementation, we use a simple recursive backtracking procedure.

VII. Related Work

Text-Mining Frameworks: Several NLP-based text analyzing packages and frameworks have been introduced in recent years. Apache UIMA [2], OpenNLP [1], NLTK [3], GATE [20], and CoreNLP [6] are among most commonly used such systems, which mostly support a set of basic NLP tasks such as tokenization, POS-tagging, parsing, NER, etc. However to our knowledge, none of these systems are able to support semantic analysis and querying (such as those in GD patterns) of the free text.

Semantic Representation of Text: Although much efforts are devoted to represent knowledge in more machine readable structures such as RDF graphs [4], there are very few research efforts to semantically represent text for the purpose of more effective and efficient text querying and mining. We already discussed the shortcomings of parse tree [30] and dependency tree [33] representations. But unfortunately most NLP-based text-mining approaches including the ones mentioned above are still using these tree-bases representations, due to lack of a better semantic representation [48]. There are some efforts to present textual knowledge in Concepts Maps [40], Concept Graphs [46], etc. However, the fact that nodes in these graphs must be unique concepts makes them less effective for text representation. Moreover, a general approach for automatically converting text into such formats is still lacking

Coreference Resolutions: Hobbs [26] introduced one of the early pronoun resolution techniques which traverses parse trees for antecedents of a pronoun. In [27], Kehler et al. utilized shallow parsing to resolve third-person pronouns and achieved only marginal improvement with respect to Hobbs. Lappin and Leass in [31] created a set of potential resolutions for each pronoun, and used recency, syntax-based preferences, and a limited number of agreement properties to suggest the final resolutions. Authors in [22] used a similar approach by employing the centering theorem. These approaches are only able to resolve pronouns and suffer from low accuracy problem. Thus, more recent works focus on either capturing some syntactic structures implying a resolution and combine them with semantic features [41], [24], [42] or utilizing statistical techniques [23], [16]. However, these approaches do not consider syntactic constructs indicating impossible resolutions, do not take advantage of taxonomical information, and can not identify the correct multi-word referent as they either consider the whole NP or single-word terms.

Named Entity Recognition: Named entity recognition or NER is first introduced by Rau in 1991 [43], and later in 1995 formalized as a fundamental task in information extraction from text [7]. Since then, several researchers proposed different techniques to extract named entities from the text, which can be divided into three categories: techniques based on manually generated patterns [11], techniques relaying on automatically learnt patterns [45], [14], [18], and statistical techniques [17], [13], [21], [44]. Unfortunately none of these techniques take complete advantage of morphological structure in the text.

VIII. Conclusion

Mining semantic information from free-text documents will provide the enabling technology for a host of important new applications. NLP-techniques, which utilize the morphological structure of the text to produce plausible parse trees, represent an important first step that must be followed by anaphora resolution, and consultation of contextual and taxonomical information from knowledge bases, including domain-specific ontologies. To perform these tasks, we have proposed a TextGraph representation and several TextGraph based techniques. These proved quite effective at identifying the semantic relationships between entities, properties and attributes in the sentences, and between those and their real-world counterparts. A number of experiments, including our recent text-mining of the whole Wikipedia [36][37] illustrate this conclusion.

REFERENCES

[1] Apache opennlp. https://opennlp.apache.org/.

[2] Apache uima. http://uima.apache.org/.

[3] Natural language toolkit. http://www.nltk.org/.

[4] Resource description framework (rdf). http://www.w3.org/RDF/.

[5] Semantic web information management system (swims). http://semscape.cs.ucla.edu.

[6] Stanford corenlp. http://nlp.stanford.edu/software/corenlp.shtml.

[7] *MUC6 '95: Proceedings of the 6th Conference on Message Understanding*, Stroudsburg, PA, USA, 1995.

[8] Charniak nlp parser. ftp://ftp.cs.brown.edu/pub/nlparser/, 2008.

[9] Sparql query language for rdf. http://www.w3.org/TR/rdf-sparql-query/, 2012.

[10] R. Alur and P. Madhusudan. Adding nesting structure to words. In *Developments in Language Theory*, 2006.

[11] D. E. Appelt, J. R. Hobbs, J. Bear, D. J. Israel, and M. Tyson. Fastus: A finite-state processor for information extraction from real-world text. In *IJCAI*, pages 1172–1178, 1993.

[12] M. Atzori and C. Zaniolo. Swipe: searching wikipedia by example. In *WWW*, pages 309–312, 2012.

[13] O. Bender, F. J. Och, and H. Ney. Maximum entropy models for named entity recognition. In *CONLL*, 2003.

[14] M. E. Califf and R. J. Mooney. Relational learning of pattern-match rules for information extraction. In *AAAI/IAAI*, pages 328–334, 1999.

[15] D. Cer, M.-C. de Marneffe, D. Jurafsky, and C. D. Manning. Parsing to stanford dependencies: Trade-offs between speed and accuracy. In *LREC*, 2010.

[16] E. Charniak and M. Elsner. Em works for pronoun anaphora resolution. In *EACL*, pages 148–156, 2009.

[17] H. L. Chieu and H. T. Ng. Named entity recognition with a maximum entropy approach. In *CONLL*, 2003.

[18] F. Ciravegna. Adaptive information extraction from text by rule induction and generalisation. In *IJCAI*, 2001.

[19] S. A. Cook. The complexity of theorem-proving procedures. In *STOC*, pages 151–158, 1971.

[20] H. Cunningham, D. Maynard, K. Bontcheva, and V. Tablan. Gate: an architecture for development of robust hlt applications. In *In Recent Advanced in Language Processing*, pages 168–175, 2002.

[21] J. R. Curran and S. Clark. Language independent ner using a maximum entropy tagger. In *CONLL*, 2003.

[22] B. J. Grosz, S. Weinstein, and A. K. Joshi. Centering: a framework for modeling the local coherence of discourse. *Comput. Linguist.*, 21(2):203–225, June 1995.

[23] A. Haghighi and D. Klein. Unsupervised coreference resolution in a nonparametric bayesian model. In *ACL*, 2007.

[24] A. Haghighi and D. Klein. Simple coreference resolution with rich syntactic and semantic features. In *EMNLP*, 2009.

[25] R. Hahn, C. Bizer, C. Sahnwaldt, C. Herta, S. Robinson, M. Bürgle, H. Düwiger, and U. Scheel. Faceted wikipedia search. In *BIS*, pages 1–11, 2010.

[26] J. Hobbs. Resolving pronoun references. *Lingua*, 44:311–338, 1978.

[27] A. Kehler, D. Appelt, L. Taylor, and A. Simma. Competitive self-trained pronoun interpretation. In *HLT-NAACL*, pages 33–36, 2004.

[28] D. Kerr, H. Mousavi, and M. Iseli. Automatic short essay scoring using natural language processing to extract semantic information in the form of propositions. In *(CRESST Report 831). University of California, Los Angeles*, 2013.

[29] K. Kipper, A. Korhonen, N. Ryant, and M. Palmer. A large-scale classification of English verbs. *Language Resources and Evaluation*, 42(1):21–40–40, Mar. 2008.

[30] D. Klein and C. D. Manning. Accurate unlexicalized parsing. In *ACL*, pages 423–430, 2003.

[31] S. Lappin and H. J. Leass. An algorithm for pronominal anaphora resolution. *Comput. Linguist.*, 20(4):535–561, 1994.

[32] T. Lee, Z. Wang, H. Wang, and S. won Hwang. Web scale taxonomy cleansing. *PVLDB*, 4(12):1295–1306, 2011.

[33] M. D. Marneffe, B. Maccartney, and C. D. Manning. Generating typed dependency parses from phrase structure parses. In *LREC*, 2006.

[34] O. Medelyan, D. N. Milne, C. Legg, and I. H. Witten. Mining meaning from wikipedia. *Int. J. Hum.-Comput. Stud.*, 67(9):716–754, 2009.

[35] H. Mousavi, S. Gao, and C. Zaniolo. Discovering attribute and entity synonyms for knowledge integration and semantic web search. *SSW*, 2013.

[36] H. Mousavi, S. Gao, and C. Zaniolo. Ibminer: A text mining tool for constructing and populating infobox databases and knowledge bases. *PVLDB*, 6(12):1330–1333, 2013.

[37] H. Mousavi, D. Kerr, M. Iseli, and C. Zaniolo. Deducing infoboxes from unstructured text in wikipedia pages. In *CSD Technical Report #130001, UCLA*, 2013.

[38] H. Mousavi, D. Kerr, M. Iseli, and C. Zaniolo. Ontoharvester: An unsupervised ontology generator from free text. In *CSD Technical Report #130003, UCLA*, 2013.

[39] D. Nadeau and S. Sekine. A survey of named entity recognition and classification. *Lingvisticae Investigationes*, 30(1):3–26, Jan. 2007.

[40] J. D. Novak and G. D. B. *Learning how to learn*. Cambridge University Press, N.Y., 1984.

[41] H. Poon and P. Domingos. Joint unsupervised coreference resolution with markov logic. In *EMNLP*, 2008.

[42] K. Raghunathan, H. Lee, S. Rangarajan, N. Chambers, M. Surdeanu, D. Jurafsky, and C. D. Manning. A multi-pass sieve for coreference resolution. In *EMNLP*, 2010.

[43] L. F. Rau. Extracting company names from text. In *Artificial Intelligence Applications, 1991. Proceedings., Seventh IEEE Conference on*, volume i, pages 29–32. IEEE, Feb. 1991.

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

[45] S. Soderland, D. Fisher, J. Aseltine, and W. Lehnert. Crystal inducing a conceptual dictionary. In *IJCAI*.

[46] J. F. Sowa. Conceptual graphs for a data base interface. *IBM Journal of Research and Development*, 20(4):336–357, 1976.

[47] M. M. Stark and R. F. Riesenfeld. Wordnet: An electronic lexical database. In *Proceedings of 11th Eurographics Workshop on Rendering*. MIT Press, 1998.

[48] A. Tan. Text mining: The state of the art and the challenges. In *PAKDD*, pages 65–70, 1999.