Learning to Generate Semantic Annotation for Domain Specific Sentences

Jianming Li, Lei Zhang, Yong Yu
Department of Computer Science and Engineering,
Shanghai JiaoTong University,
Shanghai, 200030, P.R.China
Jianming119@sina.com, tozhanglei@hotmail.com, yyu@mail.sjtu.edu.cn

ABSTRACT
Seas of web pages in the Internet contain free texts in natural language that are only read by human beings. To be understandable for machines, these pages should be annotated with semantic markups. Manually annotating large amounts of pages is an arduous work. This has made automatic semantic annotation an urgent challenge. In this paper, we propose a machine-learning based automatic annotation approach. This approach can be trained for different domains and requires nearly no manual rules. The annotation is on the sentence level and is in RDF format. We adopt a dependency grammar – Link Grammar [2] – for this purpose. ALPHA system, a prototype of this approach has been developed with IBM China Research Lab. We expect many improvements are possible for this approach and our work may be selectively adopted or enhanced.

1 Introduction

There are seas of web pages in the Internet and nearly all of them contain free texts in natural language that are only read by human beings. Annotating these pages with semantic markups is one promising way to make them understandable for machines. Unfortunately, automatic semantic annotation for the natural language sentences in these pages is a daunting task and we are often forced to do it manually or semi-automatically using handwritten rules. In this paper, we propose a machine-learning (ML) based automatic semantic annotation approach that can be trained for different domains and require almost no manual rules. The annotation resulted form this approach lies in the sentence level, i.e., we will annotate each sentence or prime sentences in a web page. This approach stems from our previous research on semantic analysis on natural language sentences using Conceptual Graphs (CG).

Free texts in the Internet contain various information in diverse domains. The method we proposed in this paper is for domain specific sentences that are sentences occur in a specific application domain. Though the sentences are limited in one domain, our method itself is domain independent and the system can be trained for various domains.

Domain specific sentences are usually very stylish in the words, phrases, grammar and semantics they employ, which lead to a strong patterned text for which machine learning based approach is effective. Our approach is independent on any ML algorithm. In the prototype ALPHA system, we employed instance-based learning. Link Grammar is first used to get the syntactic structures of sentences. The learning process then learns to map the syntactic structures to semantic structures – RDF graphs. WordNet [7] and the domain relation hierarchy are used as the domain ontology in the whole semantic analysis and representation process. Preliminary results gained from the ALPHA system demonstrated the feasibility of the approach.

The paper is organized as follows. Section 1.1 explains the concept of " Domain Specific Sentences" used in this paper. Section 1.2 briefly shows what the result RDF looks like. Section 1.3 explains the reason to adopt Link Grammar. Section 2 outlines the whole approach by giving an overview. Section 3 presents the detailed process that generates RDF graph from domain specific sentences. Section 4 discusses the result of ALPHA system. Section 5 concludes our work by comparing related work.

1.1 Domain Specific Sentences

Domain specific sentences point to those sentences that are frequently occurring in one certain application domain text but scarcely in others. They are assumed to own the following characteristics:

I. vocabulary set is limited
II. word usage has patterns
III.semantic ambiguities are rare
IV.terms and jargon of the domain appear frequently

The notion of sublanguage [3,4] has been well discussed last decade. Domain specific sentences actually can be seen as sentences in a domain sublanguage. As previous study has shown, a common vocabulary set and some specific patterns of word usage can be identified in a domain sublanguage. These results provide ground for us to assume the above characteristics about domain specific sentences. In the rest of this paper, we will show how characteristics I to III are employed in our work. Terms and jargon will be dealt with in the following section by adding them to the Link Grammar dictionary.

1.2 RDF Graph

After the annotation, sentences from web pages will be marked up with RDF statements. We illustrate the representation by using an example sentence " I go to

Shanghai" . The corresponding RDF statement will be like the following:

```
<rdf:RDF
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xmlns="http://cs.sjtu.edu.cn/apex/alpha-schema#"
>
<Concept rdf:ID="1">
<rdfs:label>I</rdfs:label>
   <WordNetSenseIndex>WN16-2-012345
   </WordNetSenseIndex>
</Concept>
<Concept rdf:ID="2">
<rdfs:label>go</rdfs:label>
   <WordNetSenseIndex>WN16-2-012345
  </WordNetSenseIndex>
</Concept>
<Concept rdf:ID="3">
   <rdfs:label>shanghai</rdfs:label>
</Concept>
<rdf:Description about="#1">
<AGNT rdf:resource="#2"/>
</rdf:Description>
<rdf:Description about="#2">
<DEST rdf:resource="#3"/>
</rdf:Description>
</rdf:RDF>
```

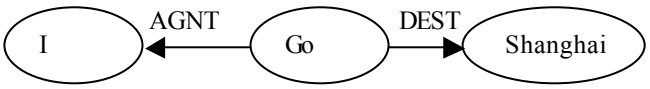

Fig. 1. RDF graph for the example sentence " I go to Shanghai"

Class " Concept" represents concept in sentence. In the current implementation, we are using WordNet [7] as experimental concept ontology. Property "WordNetSenseIndex" uniquely identifies a word sense (concept) in WordNet database. Properties such as "AGNT" (agent), " DEST" (destination) are sub-properties derived from a general property " Relation" . All the sub-properties of " Relation" are organized as a hierarchy and thus form the relation ontology. [18]

The RDF statement can also be diagramed as a directed labeled graph with nodes and arcs as depicted in figure 1. Since the diagram is simpler and easier to understand, we will use the diagram, which we call RDF graph, to represent RDF statements instead of writing long RDF statements in the rest of the paper.

1.3 Link Grammar

Link Grammar is a dependency grammar system we employ in our work. For the same sentence " I go to Shanghai" , the Link Grammar parse result is shown in the top of Fig.2. The labeled arcs between words are called links. The labels are the types of the links. For example, the " Sp*I" between " I" and " go" represents the type of links between " I" and a plural verb form. " MVp" connects verb to its modifying prepositional phrases. "Js" connects prepositions to their objects. In Link Grammar, there is a finite set of such link types.

Each word in Link Grammar has a linking requirement stating what types of links it can attach and how they are attached. The link requirements are stored in a Link Grammar dictionary. The parse result is called a linkage or a link structure. The Link Grammar parser is called a link parser. Currently, the link parser from CMU [5] has a dictionary of about 60000 words together with their linking requirements. Although the CMU link parser still has difficulties in parsing complex syntactic structures in real commercial environment, it is now ready for use in relatively large prototypes. Applying Link Grammar to languages other than English (e.g. Chinese [19]) is also possible.

The most important reason that makes us adopt Link Grammar in our work is the structure similarity between Link Grammar parse result and RDF graph. Fig.2 shows this similarity by comparing the Link Grammar parse result, the typical parse tree of a constituent grammar and the

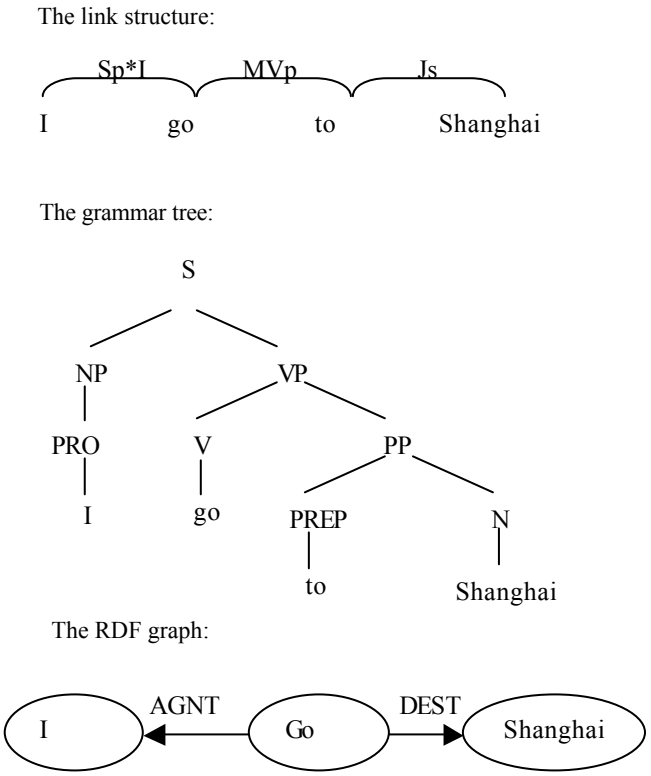

Fig. 2. Link structure is more like a RDF graph

RDF graph for the same example sentence. In fact, this similarity comes from the common foundation of both RDF graph and Link Grammar. RDF graph consists of concepts and relations. The relations denote the semantic associations between concepts. Similarly, link structure consists of words and links. The links directly connect syntactically and semantically related words [2]. Open words [17] (such as noun, adjective and verb) access concepts from the catalog of conceptual types, while closed words [17] (such as prepositions) and links help clarify the semantic relationships between the concepts.

Based on this similarity and restricted to a specific domain, we propose to automatically generate annotation by learning the mapping from link structure to RDF graph.

Another important feature of Link Grammar is that the grammar is distributed among words [2]. There can be a separate grammar definition (linking requirement) for each word. Thus, expressing the grammar of new words or words that have irregular usage is relatively easy. We can just define the grammar for these words and add them to the dictionary. This is how we deal with terms and jargon of a domain in our approach. Because the vocabulary set for a domain is limited (see section 1.1), we can add all unknown words (including terms and jargon) to the current dictionary of Link Grammar with affordable amount of work.

2 Overview of the approach

Our approach of automatic page annotation is a process consisting of two phases: the training phase and the generating phase, as shown in Fig.3.

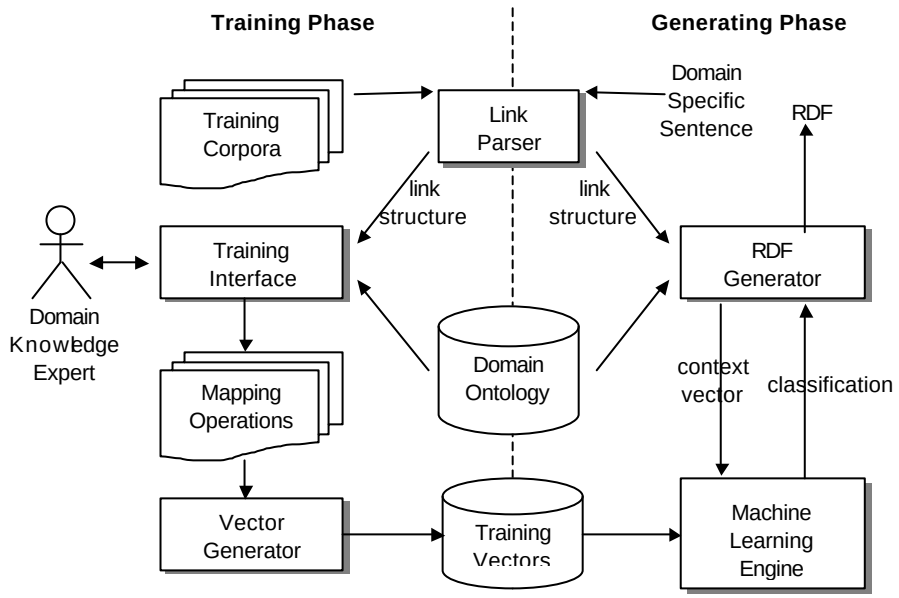

Fig. 3. Overview of the approach

The first step of both phases is to invoke Link Grammar, and parse the sentence into its link structure, which will be mapped to RDF through different means in the two phases.

In the training phase, some domain experts will go through a three-operation process to transfer the link structure into RDF graph manually based on their domain knowledge. Each operation maps a certain part of the syntactic structure to its corresponding semantic representation according to the syntactic and semantic context.

Concepts, schemata [1] and relations contained in the semantic representation are selected from the domain ontology. Since semantic ambiguities are rare in domain specific sentences (see section 1.1), it is relatively easy to perform these mapping operations (the process of semantic analysis).

What training phase does is a preparation. Before the system can learn to do the mapping in the generating phase, we convert the mapping into machine learning area. Most of studied tasks in machine learning area are to infer a function that classifies a feature vector into one of a finite set of categories [6]. We thus translate the mapping operation into classification operation by encoding the operation as category and encoding the context in which the operation is performed as feature vector. We call the feature vector *context vector* since it encodes the context information of an operation. The vector generator in the left down corner of Fig.3 is the component that executes this task.

After sufficient training vectors and categories are obtained in the training phase, the system can enter into the generating phase. RDF generator, the main part of the generating phase will implement the following algorithm under the help of ML engine and Link Grammar after it is given a sentence from object domain.

```
1  get the link structure for the sentence from link parser.
2  generate an empty RDF graph.
3  for (i = 1 to 3) { //perform the three kinds of operations
4      generate all possible context vectors from link
       structure for the i-th kind of operation.
5      for (every context vector) {
6          if (an operation is needed for this vector) {
7          classify the vector using ML engine.
8          decode the classified category as an operation.
9          perform the operation on the link structure and
```

[1] Schemata, is a set of RDF graphs describing background information in a domain.

```
10          modify the RDF graph according to the operation
11          result (using concepts, schemata and relations
12                  from the domain ontology).
13          }
14      }
15  }
16 do integration on the RDF graph.
17 output the final RDF annotation for the sentence.
```

Algorithm 1. The algorithm of the RDF Generator

Our approach is independent of specific ML algorithm used. In ALPHA system, we adopt IBL (Instance Based Learning) for the ML engine because IBL makes it easy to determine whether an operation is needed for an arbitrary vector in the above algorithm (line 6). IBL can return a distance value along with the classification result. If the distance value is too large, it can be determined that no operation is needed for the vector because it is far from being similar to existing training vectors and may be deemed as noise. For other learning methods, this determination may not be easily achieved.

In the following section, we will explain Algorithm.1 and the three operations by taking an example sentence "The polo with an edge is refined enough for work" , which is excerpted from a corpora of clothes descriptions collected from many clothes shops on the Web. In the sentence, " Polo" is a brand and represents a certain kind of shirts and "edge" actually means collar. The link structure for this sentence is shown in Fig.4.

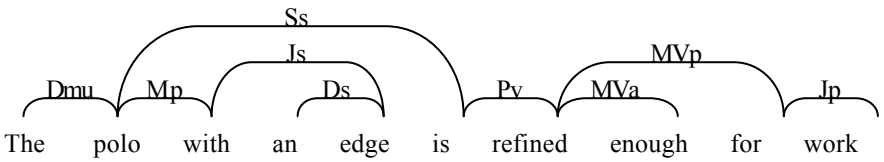

Fig. 4. The link structure for the example sentence

3 Learning to generate RDF

In this section we will introduce the three operations that map a link structure to RDF: word-conceptualization, link-folding and relationalization. These three kinds of operations must be performed exactly in the right order in both the training phase and the generating phase because a later operation may use information generated in the previous operations. In section 3.4, the integration on RDF graph (line 16 of Algorithm.1) is explained.

3.1 Word-Conceptualization

I. Function

Word-conceptualization is the first operation to be performed. Its function is to annotate open words as concepts in the sentence to form the skeleton of the initial empty RDF graph and mark close words for further operation. This operation can be seen as a word sense disambiguation operation.

II. Training

In the training phase, domain experts select all the open words in the link structure one by one. Once an open word is selected, the training interface can provide the expert a list of possible concepts or schemata retrieved from the domain ontology. The expert then chooses the appropriate one from the list.

This operation is then encoded by the vector generator into a context vector and its category. For example, the context vector for the open word "polo" in the example sentence may be <polo, NN, Dmu, Mp>[2] in which " NN" is the POS (part-of-speech) tag[3] and " Dmu" and " Mp" are the innermost left and right link types of "polo" (see Fig.4). All the context information is obtained from the link structure.

The category for the context vector is encoded as the result of the operation – the ID of the selected concept or schemata in the domain ontology. The encoding is something like "WN16-2-330153" which can be used later as a key to retrieve concept (in WordNet terminology, word sense) from the WordNet database.

Since WordNet is not specific for any domain, some words in a certain domain may not exactly match any sense in the list. For those words the experts are asked to choose the most similar sense instead of adding a new sense to WordNet so as to preserve the hierarchy in WordNet for further research.

III. Generating

Generating all possible context vectors (line 4 of Algorithm 1.) is actually to generate one context vector for each open word in the link structure of the sentence. The generated context vector is then sent to the ML engine as to do a classification. The returned category is an encoding of concept or schema ID. In line 9 of Algorithm 1, the RDF generator retrieves the concept or schema from the domain ontology according to the decoded ID and creates a concept node in the RDF graph.

Because word usage has patterns in domain specific sentences, we expect that similar context vectors appear

[2] The vector is just an example. For brevity, we are not trying to make the vector encoding perfect in this paper. Actually, what context information is encoded into the vector is a separate problem. This problem is isolated into the vector generator component. In the current implementation, we defined a configuration file for the vector generator to address the issue.

[3] We can augment the link parser with a POS tagger so that the accurate POS tag information can be added to the link structure and be obtained from it later.

for a given open word on a specific word sense. Based on these similar context vectors, we expect the ML engine can return correct classification with a high possibility since the semantic ambiguity is also rare in domain specific sentences.

After this step, all concept nodes of the RDF graph should be created. The RDF graph for the example sentence is shown in Fig.5. For convenience, we use simple concept names in Fig.5. The "S-WORK" is the " SUITABLE-FOR-WORK" schema in domain ontology.

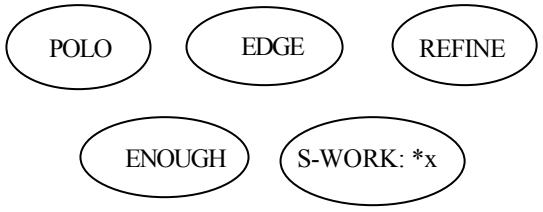

Fig. 5. RDF graph after word-conceptualization

3.2 Link-Folding

I. Function

The following two operations focus on creating the semantic relations between the concepts. Closed words (especially prepositions) with their links imply relationships between the concepts of the words they connect. In the example sentence, "… polo --- with --- edge …" fragment implies a PART relation between [POLO:#] and [EDGE]. We then " fold" the 'with' and its left and right links and replace them with a PART relation. This is just what the link-folding operation does.

Closed words with their links representing semantic relations can be seen as word usage patterns. In domain specific sentences, such patterns are expected to occur frequently. This actually enables the machine to learn the patterns from training corpora. In addition, since semantic ambiguities are rare in domain specific sentences, it can be expected that the result of the learning converge on the correct relation. Similar analysis also applies to the next operation – relationalization in section 3.3.

II. Training

In the training phase, the domain expert can select any closed word that connects two concepts and implies a semantic relation and map it to the responding semantic relation from the relation ontology[4].

The context vector for this operation may encode context information such as the POS tag of the closed word, the left and right link types and the two concepts. The category is an encoding of the relation ID in the domain ontology. For the "…polo --- with --- edge …" case, the context vector may be <with, IN, Mp, Js, POLO, EDGE>[5].

[4] For brevity, we omitted the direction of a relation here.

[5] The POLO and EDGE in the vector are actually the concept IDs in the domain ontology. We will use the same convention in the following vector examples.

And the category is the encoding of the ID of the PART relation in the domain ontology.

III. Generating

In the generating phase, generating all possible context vectors for this operation (line 4 of Algorithm 1.) is actually generating one context vector for every possible case in which a closed word connects two concepts. This needs consult the concept information generated in the word-conceptualization operations. If an operation is needed for the vector, it is sent to the ML engine to do a classification. The returned category is an encoding of the relation ID in domain ontology. In line 9 of Algorithm.1, the RDF generator retrieves the relation from domain ontology according to the ID and creates the relation between the two concepts.

For the example sentence, there are three closed words that need link-folding operation: 'with', 'is' and 'for', as shown in Fig.4. Among them, the word 'is' is an auxiliary verb and 'with' and 'for' are prepositions.

The relation implied by the auxiliary verb 'is' is THEME and the 'for' between 'refined' and 'work' implies a RESULT relation. The RDF graph after this step has relations added between concepts. As to our example sentence, the RDF graph has grown to Fig.6.

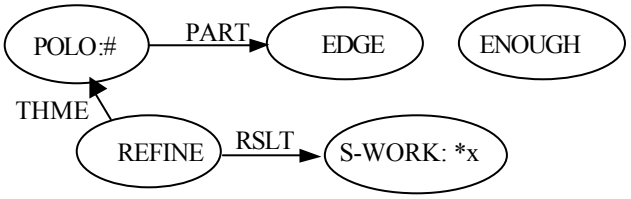

Fig. 6. RDF graph after link-folding

3.3 Relationalization

I. Function

Semantic relation can also be implied by a link that directly connects two concepts in the link structure. For example, the 'MVa' link between 'refined' and 'enough' in the link structure of example sentence implies a MANNER relation. The relationalization operation translates this kind of links into corresponding semantic relations.

II. Training

In the training phase, domain knowledge expert can select any link that implies a semantic relation between concepts it connects. The expert then selects the semantic relation from the domain ontology for the connected two concepts.

The context vector for this operation can include information such as the link type and the concepts. For the "…refined –MVa – enough …" , the context vector may be <MVa, REFINE, ENOUGH>. The category for the context vector can be encoded as the relation ID in the domain ontology. For the above vector, it is the ID of the MANNER relation.

III. Generating

In the generating phase, generating all possible context vectors for this operation (line 4 of Algorithm 1.) is actually generating one context vector for every link that connects two concepts. If an operation is needed for the vector, it is sent to the ML engine to do a classification. The returned category is an encoding of the relation ID in domain ontology. In line 9 of Algorithm.1, the RDF generator retrieves the relation from domain ontology according to the ID and creates the relation between the two concepts.

After this step, more relations may be created in the RDF graph. As to the example sentence, the MANNER relation will be created to connect the [REFINE] concept and the [ENOUGH] concept and the whole graph grows to Fig.7.

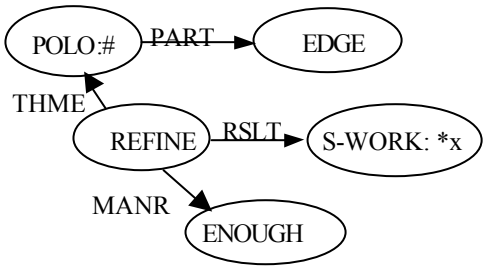

Fig.7.RDF graph after relationalization

3.4 Integration

Integration is the last step (line 16) in Algorithm.1. This step is not a part of the training phase. It only appears in the generating phase and it is the only step that uses manually constructed heuristics. What it does includes simple co-reference detection and nested graph creation.

In the discussion of the previous three operations, we don't involve lambda expressions for brevity. In fact, they may appear when words for concepts are missed in the sentence. They may also be introduced when schema is selected in word-conceptualization phase. In order to complete the RDF graph, we need to draw co-reference lines between the variables in these lambda expressions.

Although there is machine-learning based approach for co-reference detection [9], in our work we mainly focus on the generation of RDF graph for a single sentence. Discourse analysis and co-reference detection is left for a separate research work. For different domains, we may construct different heuristics for them. In our current wok we simply make all undetermined references to point to the topic currently under discussion.

Nested graph (context) may be introduced by expanding schema definition or removing modal/tense modifiers of a concept. Although RDF specification lacks a clear semantics about RDF reification, we are now using RDF reification mechanism to represent nested graph (context). In our example, we have mentioned in section 3.1 that the concept type SWORK is actually a " SUITALBE-FOR-WORK" schema from the domain ontology. We can do an expansion on it. Fig.8 is the definition for the " SUITALBE-FOR-WORK" schema. SUTB represents the relation SUITABLE.

After the expansion, we can do a simple co-reference detection that draws a co-reference line between the

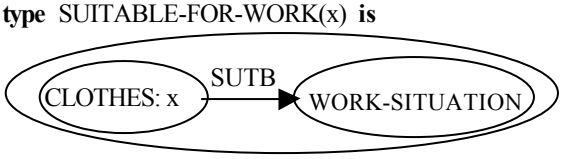

Fig. 8. The definition for SUITABLE-FOR-WORK

undetermined variable *x* and the current topic [POLO:#]. After this step, the final graph is generated. Fig.9 is the result for our example sentence "The polo with an edge is refined enough for work".

3.5 Summary

Through the sections from 3.1 to 3.4, we have explained how we map link structure to RDF graph and convert the

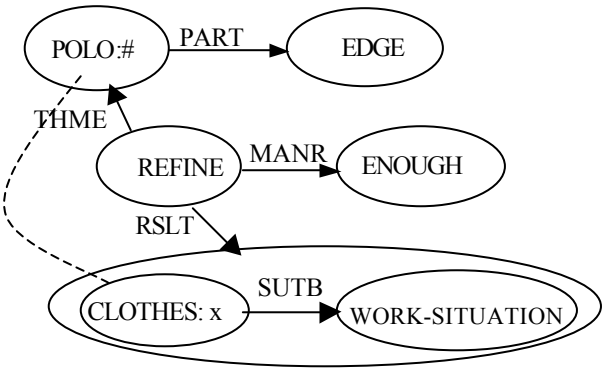

Fig. 9. The final RDF graph of the example sentence

mapping to machine learning area. Word-conceptualization builds concepts in the RDF graph. Link-folding and relationalization connect concepts with semantic relations. In the last step, we use manually constructed heuristics to do simple co-reference detection and nested graph creation.

4 Results

We have developed a prototype called ALPHA system written in C. ALPHA system is now running on Solaris. It can be trained for different domains. Currently in our work, clothes domain is chosen as the sample domain. Nearly 300 clothes descriptions, 500 sentences have been collected from clothes shops on the Web[6] and are trained in ALPHA system. Among them, 34 descriptions and 93 sentences are reserved for testing. The test result is shown in Fig.10. Using different IBL algorithms, the accuracy[7] of concepts varies from 60% ~ 80%, and that of relation varies from 45% ~ 60%.

The result demonstrated the feasibility of our approach.

---

[6] Those online shops include www.brooksbrothers.com and www.gap.com, etc.

[7] Here the accuracy of concepts = concepts annotated correctly / total concepts, and the accuracy of relations = links annotated correctly / links that should be annotated.

Link Grammar has a strong impact on the accuracy of ALPHA system. Although its characteristics make it relatively easily to add domain grammar, it has some trouble in disambiguating the syntactic structure of over-abridged sentences in clothes domain, such as " Back vent." , which causes a serious failure in ALPHA system. Though we are aware of the problem, we will let it be at present because we want to pay more attention to semantic disambiguation.

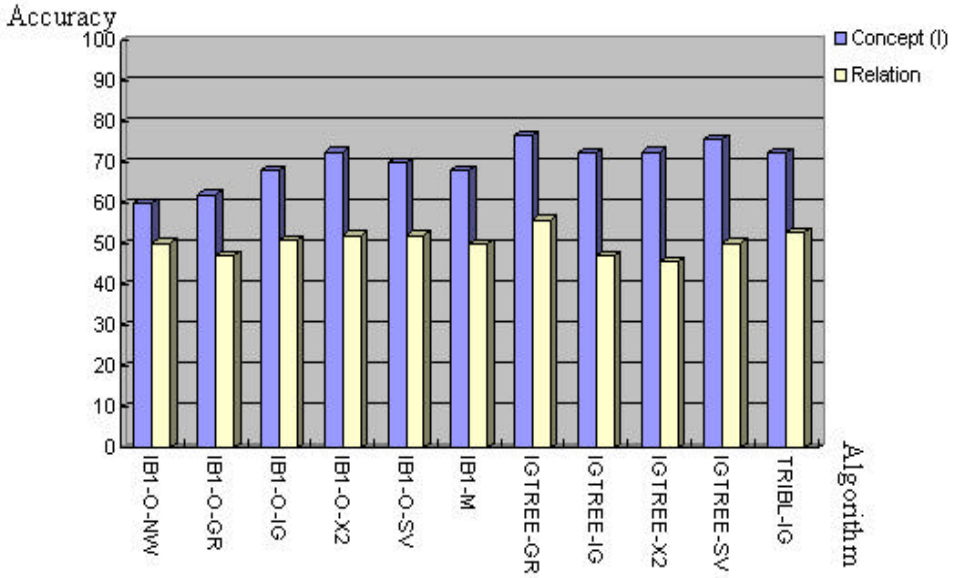

Fig. 10. The accuracy of concepts and relations about different algorithm

To improve the accuracy of ALPHA system, we are considering developing new algorithms that can compute the distances of vectors more accurately. We are also considering making changes in the architecture so as to support the analysis of clauses and idioms. Further more, other application domains will be selected to test our approach.

5 Related works

Ontology-based annotation is most studied such as [15], [16]. [15] extends HTML with semantic extensions and builds an interactive and graphic tool to help annotate web pages manually. What it does is to associate an object in HTML with a concept from their ontology. After gaining experiences from manually annotation, they also conceive an information extraction-based approach for semi-automatic annotation of natural language texts by mapping terms to ontological concepts. Different from it, our approach is fully automatic after the training phase. Our approach also generates the semantic markup in standard RDF format.

In natural language annotation, grammar-based approach is often used. They can roughly be divided into slot-filling and structure-mapping categories according to their generating techniques.

Slot-filling techniques such as [12] fill template semantic graphs with thematic roles identified in the parse tree. Often the graph of one tree node in the syntactic parsing tree is constructed using the graph of its child nodes according to construction rules on how to fill the slots. Although this approach has been successfully applied in many applications, it heavily depends on manually created construction rules on the parse tree.

Another kind of technique advanced in previous work is to directly map between syntactic structure and semantic structure such as [13]. We call them structure-mapping. In this respect, they are more similar to our work. To map to more flat structures of conceptual graphs, [13] uses syntactic predicates to represent the grammatical relations in the parse tree. Instead, in our work, Link Grammar is employed to directly obtain a more flat structure. Different from [13]'s approach, our work doesn't use manual rules. Moreover, we separate the semantic mapping into several steps that greatly reduce the total number of possibilities. In another work in [14], parse tree is first mapped to a "syntactic conceptual graph" . The " syntactic conceptual graph " is then mapped to a real conceptual graph. This approach again heavily uses manually constructed mapping rules.

Up to now most methods for annotation are by hand or heavily depend on rules created manually. These methods will have difficulty in applying to the Web because of the tremendously large amounts of pages. Our approach provides an automatic way to annotate them in a faster and robust way. Research on machine learning in natural language processing using corpora data [6] has increased significantly and there are growing numbers of successful applications of symbolic machine learning techniques[10,11]. Our work presents a preliminary inquest into the use of traditional machine learning techniques to automatically generate semantic markups for domain specific sentences. We expect that many improvements are possible and our work may be selectively adopted or enhanced.

References

1 Walter Daelemans, Jakub Zavrel, Kovan der Sloot, and Antal van den Bosch, *TiMBL: Tilburg Memory Based Learner version 3.0 Reference Guide*, March 8, 2000

2. Daniel D.Sleator and Davy Temperley, Parsing English with a Link Grammar, in the *Third International Workshop on Parsing Technologies*, August 1993.
3. Sager Naomi, "Sublanguage: Linguistic Phenomenon, Computational Tool," In R. Grishman and R. Kittredge (eds.), *Analyzing Language in Restricted Domains: Sublanguage Description and Processing*, Lawrence Erlbaum, Hillsdale, NJ, 1986
4. R. Kittredge and J.Lehrberger, " Sublanguage: Study of language in restricted semantic domain", Walter de Gruyter, Berlin and New York, 1982.
5. The information about the link parser from Carnegie Mellon University is available at: http://link.cs.cmu.edu/link/index.html
6. Raymond J.Mooney and Claire Cardie, Symbolic Machine Learning for Natural Language Processing, in the tutorial of *ACL'99*, 1999. Available at http://www.cs.cornell.edu/Info/People/cardie/tutorial/tutorial.html
7. George A.Miller, WordNet: An On-line Lexical Database, in the *International Journal of Lexicography*, Vol.3, No.4, 1990.
8. Mitchell P.Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz, Building a large annotated corpus of English: the Penn Treebank, *Computational Linguistics*, 19:313-330, 1993.
9. McCarthy,J., and Lehnert,W., Using Decision Trees for Coreference Resolution. In Mellish, C. (Ed.), *Proceedings of the Fourteenth International Conference on Artificial Intelligence*, pp. 1050-1055. 1995.
10. Claire Cardie and Raymond J.Mooney, Machine learning and natural language (introduction to special issue on natural language learning). *Machine Learning*, 34, 5-9, 1999.
11. Brill, E. and Mooney, R.J. An overview of empirical natural language processing, *AI Magazine*, 18(4), 13-24, 1997.
12. Cyre,W.R., Armstrong J.R., and Honcharik,A.J., Generating Simulation Models from Natural Language Specifications, in *Simulation* 65:239-251, 1995.
13 Paola Velardi, et.,all, Conceptual Graphs for the analysis and generation of sentences, in *IBM Journal of Research and Development*, 32(2), pp.251-267, 1988.
14 Caroline Barrière, From a Children's First Dictionary to a Lexical Knowledge Base of Conceptual Graphs, Ph.D thesis, School of Computing Science, Simon Fraser University, 1997. Available at ftp://www.cs.sfu.ca/pub/cs/nl/BarrierePhD.ps.gz
15 M.Erdmann, A.Maedche, H.-P.Schnurr, and Steffen Staab. From manual to semi-automatic semantic annotaiton: about ontology-based text annotation tool, In P. Buitelaar & K. Hasida (eds). Proceedings of the COLING 2000 Workshop on Semantic Annotation and Intelligent Content, August 2000
16 Michael Schenk. *Ontology-based semantical annotation of XML*. Master's thesis, Univeritat (TH) Karlshruhe, 1999
17. James Allen, " Natural Language Understanding", 2nd edition, pp.24-25, the Benjamin/Cummings Publishing, 1995.
18 John F. Sowa, *Knowledge Representation: Logical, Philosophical, and Computational Foundations*, Brooks Cole Publishing Co., Pacific Grove, CA, 2000.
19 Carol Liu, Towards A Link Grammar for Chinese, Submitted for publication in *Computer Processing of Chinese and Oriental Languages - the Journal of the Chinese Language Computer Society*.