HDSKG: Harvesting Domain Specific Knowledge Graph from Content of Webpages

Xuejiao Zhao[1,2,3], Zhenchang Xing[4], Muhammad Ashad Kabir[5], Naoya Sawada[6], Jing Li[3], Shang-Wei Lin[1,3]
[1]Rolls-Royce@NTU Corporate Lab, Nanyang Technological University (NTU), Singapore
[2]Joint NTU-UBC Research Centre of Excellence in Active Living for the Elderly (LILY), NTU, Singapore
[3]School of Computer Science and Engineering, NTU, Singapore
[4]Research School of Computer Science, Australian National University, Australia
[5]School of Computing and Mathematics, Charles Stuart University, Bathurst, NSW, Australia
[6]Cloud Service Division, NTT Communications Corporation, Japan
{xjzhao, shang-wei.lin, jli030}@ntu.edu.sg; zhenchang.xing@anu.edu.au; akabir@csu.edu.au; naoya.sawada@ntt.com

***Abstract*—Knowledge graph is useful for many different domains like search result ranking, recommendation, exploratory search, etc. It integrates structural information of concepts across multiple information sources, and links these concepts together. The extraction of domain specific relation triples (subject, verb phrase, object) is one of the important techniques for domain specific knowledge graph construction. In this research, an automatic method named *HDSKG* is proposed to discover domain specific concepts and their relation triples from the content of webpages. We incorporate the dependency parser with rule-based method to chunk the relations triple candidates, then we extract advanced features of these candidate relation triples to estimate the domain relevance by a machine learning algorithm. For the evaluation of our method, we apply *HDSKG* to Stack Overflow (a Q&A website about computer programming). As a result, we construct a knowledge graph of software engineering domain with 35279 relation triples, 44800 concepts, and 9660 unique verb phrases. The experimental results show that both the precision and recall of *HDSKG* (0.78 and 0.7 respectively) is much higher than the openIE (0.11 and 0.6 respectively). The performance is particularly efficient in the case of complex sentences. Further more, with the self-training technique we used in the classifier, *HDSKG* can be applied to other domain easily with less training data.**

***Index Terms*—Knowledge Graph, Structural Information Extraction, openIE, Stack Overflow, Dependency Parse**

I. Introduction

Recently, the trends and development of information retrieval and mining have transferred from the document-centric to the entity-centric [1]. So knowledge graph which integrates the structural information of concepts across multiple information sources, and links these concepts together [2] is useful for many different domains, such as search result ranking, recommendation, exploratory search, etc [3], [4], [5], [6]. The webpages such as Wikipedia, Quora are potential repositories for building up knowledge graph. Thus, automatic knowledge graph construction techniques for the content of webpages become a widely research topic. [4], [7], [8].

Open information extraction refers to the extraction of relation triples without specifying the fixed relation schema, typically the relation is a triple with (subject, verb phrase, object) structure [9], [10] (Subjects and objects are collectively called concepts). The relation triples are one of the important components of the knowledge graph [11]. There are many researches on open information extraction including NELL [12], OpenIE [13], and Google [14]. The key challenge in construction of the knowledge graph is the extraction of relation triples with high precision and recall.

Knowledge Graph is also popularly used in software engineering domain [6], [15], [16], [17] to solve various problems such as software requirements analysis and design [18], coding support, documentation [6], maintenance and testing [19]. But there are few studies focus on structural knowledge-base construction in software engineering domain. Large scale knowledge-base construction (KBC) in other domains were studied intensely over the last decade [12], [20], [21], [22], [23], [24], [25]. Such as relational database which allow users to semantically query relationships and properties of natural language resources, including links to other related datasets [7]. It can be used to straightforward search engine algorithms or other search operations and improve the accuracy of information retrieval. Even though there are some research focusing on ontology construction in software engineering domain [26], [27]. They only use the domain key words as the concepts and explore the taxonomic relations of the key words. However, there are still many domain concepts indicating richer semantic and more useful relations between the concepts needed to be mined.

Stack Overflow is the most popular Q&A website about computer programming [27], [28], every question requires the asker to give 1-5 tags. The *tagWiki* is the content used to describe the definition and some related resource of every tag in Stack Overflow. Such a knowledge repository contains huge number of sentences with affluent concepts and concepts relations descriptions of software engineer domain. These relations are essential for construction of software engineer domain knowledge graph. There are a substantial amount of work that explore the knowledge of *tagWiki* [1], [29], but none of them focus on extraction of the relation triples.

In this paper, an automatic method named *HDSKG* (Harvesting Domain Specific Knowledge Graph) is proposed to discover domain specific concepts and their relation triples from the domain specific webpages. We incorporate the dependency parser with the rule-based method to chunk the relations triple candidates, then extract novel features of those triples to estimate domain relevance by self-training SVM (Support Vector Machine) classifier.

For the evaluation, we conduct a case study which extracts the knowledge graph from Stack Overflow. Our experimental results show that both the precision and recall of *HDSKG* (0.78 and 0.7 respectively) is much higher than the openIE (0.11 and 0.6 respectively) – the state of the art tool for relation triples extraction. Furthermore, self-training SVM classifier makes *HDSKG* easy to be applied to other domains with less training data.

This paper makes following four major contributions:

- We propose *HDSKG* that extracts the dependencies of the NP (Noun Phrases) and VP (Verb Phrases) from sentences to generate relation triples by incorporating the dependency parser with rule-based chunking.
- We extract advanced features from the relation triples, then leverage a self training SVM classifier and domain lexicon to estimate the domain relevance of the relation triples with a small number of training data.
- We apply *HDSKG* to Stack Overflow tagWiki, and harvest a knowledge graph of software engineering domain with 44800 concepts, 9660 unique verb phrase and 35279 relation triples, and it is available online [1].

II. Related Work

Automatic knowledge graph construction is a popular research topic. According to the relations property, we divide them into 3 categories.

The first category extract fixed relations using information extraction techniques to construct knowledge graphs. The typical techniques such as CiteSeerX [30], DIG [31], Pujara et al. [32], and NELL [12], etc. focus on small-scale fixed relations extraction with high precision. Deepdive [3] extract relations by Markov Logic Networks (MLNs) and improve the extraction by the bootstrapping system [33]. Bootstrapping technique is similar to self-training in our system.

The second category uses open information extraction technique. It doesn't need to appoint some fixed relation in advance. Representative technique are proposed by Prismatic [5], Schmitz Michael et al. [34], and Fader et al. [35]. Abebe and Tonella et al. [36] presented a semi-automated approach to construct domain concept ontology from source code identifiers. Our system belongs to this category and we further add the function for domain relevance estimation.

The third category leverage structured data sources to build up the knowledge graph. YAGO [4] and YAGO2s [37] is a structured knowledge base which automatically extracted from Wikipedia with the data e.g., categories, redirects, infoboxes, etc. The knowledge graph of YAGO [4] and YAGO2s [37] is built by defining entity classes from the conceptual Wikipedia categories. As of 2012, YAGO2s has more than 10 million entities and 120 million relations about these entities, the accuracy is above 95% which is checked manually.

Knowledge graph is also widely researched in software engineering domain. Padhye et al. [6] propose a technique to extract the profiles of the usage of API to connect people, projects and libraries in a network. Subramanian et al. [17] link source code examples to API documentation by extraction of entity linking. To the best of our knowledge, our work is the first attempt for mining knowledge graphs which contain relation description for each entity from Q&A website.

III. The Approach

In this section, we describe the approach of *HDSKG* to show how *HDSKG* harvest domain specific knowledge graph from content of webpages. In particular, we present how *HDSKG* works on natural language texts sources to extract relation triples of domain specific knowledge graph.

A. Architecture of HDSKG

Fig. 1 is the framework of *HDSKG*. The input of *HDSKG* is the natural language text materials from content of webpages (e.g., healthcare webpages, computer programming webpages, etc.). The output of *HDSKG* is the knowledge graph of target domain .

The *HDSKG* contains two main parts as shown in Fig. 1 named *HDSKG Chunking* and *HDSKG Domain Relevance Estimation*. *HDSKG Chunking* incorporates the rule-based method with dependency parser to chunk candidate relation triples. This part comprises five main steps, namely preprocess text, split sentences, add NLP (Natural Language Processing) makeup, chunk NP and VP, and chunk relation triples. The second part is used to estimate domain relevance of the candidate relation triples, which comprises two main components, namely extract features of candidate relation triples, estimate domain relevance of each relation triples by self-training SVM classifier.

B. Pre-process Text

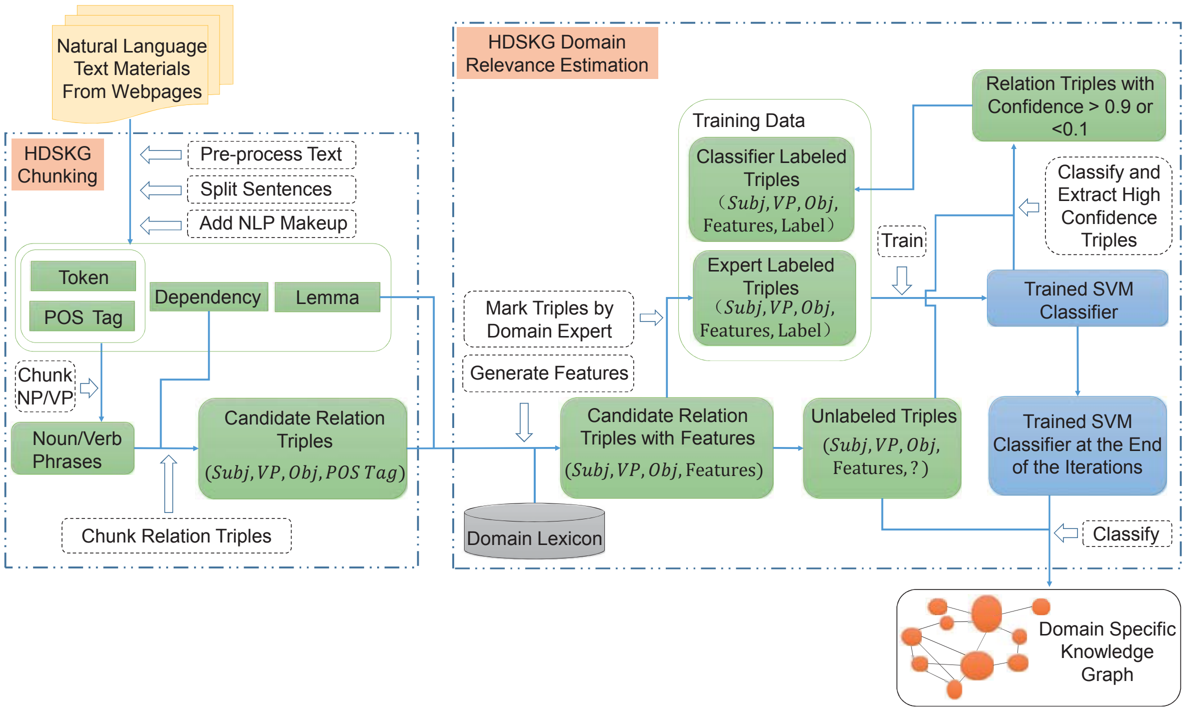
In this step, we extract all the text content from the html files of webpages and filter some useless information which don't contain concepts and relations like hyperlink lists, code snippets, etc. Then we split the text content into sentences.

After that, we enrich the sentences by two steps:

- Recording the source information for every sentence, e.g., the title of the source webpage from which the sentence are extracted, and the sequence of sentence in the source webpage.
- Making the sentences readable without the context: replacing the pronouns at beginning of the sentences (e.g., it, she, he) with the title of source webpage [9].

Take the sentence *"It is written in C++, and is ultimately derived from the Borland InterBase 6.0 source code."* for example. It is the third sentence in the *tagWiki* of *Firebird*, so we record the title of the source webpage – *Firebird* and the sentence sequence – three. Then we replace the *"it"* to *"Firebird"*, so the sentence become *"Firebird-3, Firebird is*

[1]http://neo4j.tuntunkun.org/xuejiao/



Natural Language Text Materials From Webpages
HDSKG Chunking
Pre-process Text
Split Sentences
Add NLP Makeup
Token
POS Tag
Dependency
Lemma
Chunk NP/VP
Noun/Verb Phrases
Candidate Relation Triples $(Subj, VP, Obj, POS\ Tag)$
Chunk Relation Triples
HDSKG Domain Relevance Estimation
Mark Triples by Domain Expert
Generate Features
Domain Lexicon
Candidate Relation Triples with Features $(Subj, VP, Obj, Features)$
Training Data
Classifier Labeled Triples $(Subj, VP, Obj, Features, Label)$
Expert Labeled Triples $(Subj, VP, Obj, Features, Label)$
Train
Unlabeled Triples $(Subj, VP, Obj, Features, ?)$
Relation Triples with Confidence > 0.9 or <0.1
Classify and Extract High Confidence Triples
Trained SVM Classifier
Trained SVM Classifier at the End of the Iterations
Classify
Domain Specific Knowledge Graph

Fig. 1. The Framework of *HDSKG*

written in C++, and is ultimately derived from the Borland InterBase 6.0 source code."

C. Add NLP Makeup

NLP is widely used for natural language pre-processing and semantic analysis [38], [39], [40]. The NLP makeup component in *HDSKG* system adopts the tool named *coreNLP* of Stanford for Tokenization, POS (Part of Speech) Tagging, Dependency Parsing and Lemmatization [41], [42], [43], [44].

Tokenization is used to break the text into words, phrases, or symbols.

The POS is the category of words (or, more generally, of lexical items) which has similar grammatical properties. e.g., NNP:Proper noun, singular, VBZ:Verb, third person singular present, RB:Adverb. Fig. 2 shows the POS tagging result of definition sentence of *Firebird* generated by *coreNLP*.

The goal of lemmatization is to reduce inflectional forms and derivational related forms of a word to a common base form.

D. Chunk Candidate Relations Triples

There are twice chunking to get the candidate relation triples. Firstly, *HDSKG Chunking* chunks the tokens in the sentence to NP and VP according to the POS, then utilizes the dependency of the tokens to chunk the VP and NP to relation triples.

1) Chunk NP and VP by Rule-Based Chunking: In this step, we adapt a fulltext parsing techniques called "rule-based chunking" to extract NP and VP [45], [46].

In our system, the NP and VP are identified by the regular expressions as shown in Tab. I. Where (MD) is modal; (VB.) stands for different categories of verb such as VB - verb base form, VBG - verb gerund or present participle, VBN - verb past participle, VBP - verb non-3rd person singular present, VBZ - verb 3rd person singular present. (NN.*) stands for different categories of noun such as NN - singular or mass noun, NNS - plural noun, NNP - singular proper noun, NNPS - plural proper noun. (JJ) represents an adjective; (RB) is adverb; (DT) presents an article; and (IN*) means any preposition or subordinating conjunction. The "VVP" is the VP with open clausal complement.

TABLE I
REGULAR EXPRESSION OF DIFFERENT CHUNKS

Name	Regular Expression
VVP	(MD)*(VB.*)+(JJ)*(RB)*(JJ)*(VB.*)?(DT)?(TO*)+(VB)+ (MD)*(VB.*)+(JJ)*(RB)*(JJ)*(VB.*)?(DT)?(IN*)+(VBG)+
VP	(MD)*(VB.*)+(CD)*(JJ)*(RB)*(JJ)*(VB.*)?(DT)?(IN*|TO*)+ (MD)*(VB.*)+(JJ)*(RB)*(JJ)*(VB.*)?(DT)?(IN*|TO*)+ (MD)*(VB.*)+(JJ)*(RB)*(JJ)*(VB.*)+ (MD)*(VB.*)+
NP	(CD)*(DT)?(CD)*(JJ)*(CD)*(VBD|VBG)*(NN.*)*- (POS)*(CD)*(VBD|VBG)*(NN.*)*- (VBD|VBG)*(NN.*)*(POS)*(CD)*(NN.*)+

In the regular expressions, "?" stands for whether or not there is such a determinant; "*" means zero or more determinant; "+" means must have such a determinant; "-" means continue to next row.

Using the rule-based chunking, we chunk the tokens of sentence to VP and NP. For example, the sentence *"PyTables is built on top of the HDF5 library, using the Python language and the NumPy package."*, we chunk VP *"is built on"* and *"using"*; NP *"PyTables"*, *"HDF5 library"*, *"Python language"*, and *"NumPy package"* as shown in Fig. 3.

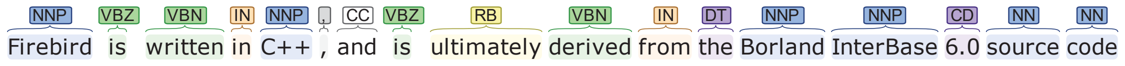

NNP
VBZ
VBN
IN
NNP
,
CC
VBZ
RB
VBN
IN
DT
NNP
NNP
CD
NN
NN
Firebird is written in C++ , and is ultimately derived from the Borland InterBase 6.0 source code

Fig. 2. POS tagging results of the definition sentence of the Firebird

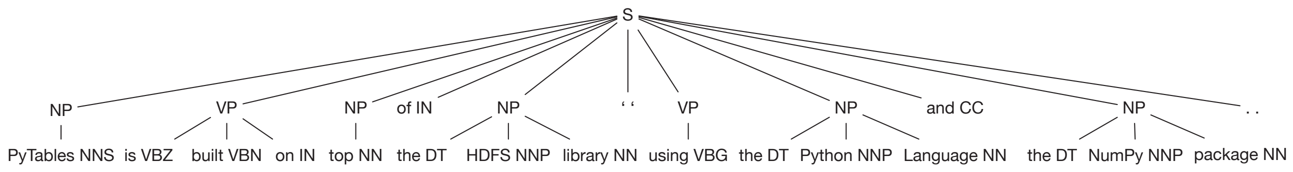


Fig. 3. Result of NP and VP Chunking

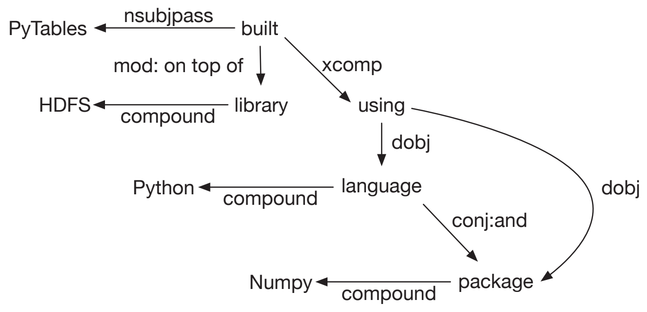


Fig. 4. Graphical representation of the Dependencies for the sentence: *"PyTables is built on top of the HDF5 library, using the Python language and the NumPy package."*

2) Dependency Parsing: Dependency Parsing extracts the grammatical structure of the sentence, then derive relationships between "head" words (also known as Governor) and words which modify the heads (also known as dependent). Fig. 4 shows a directed graph representation of the dependencies for the sentence: *"PyTables is built on top of the HDF5 library, using the Python language and the NumPy package."*. The nodes of Fig. 4 are the words of the sentence, and the edges in Fig. 4 are grammatical relations.

Tab. II presents some important dependencies of sentence *"PyTables is built on top of the HDF5 library, using the Python language and the NumPy package."* used in our system.

As Tab. II shows, "nsubjpass" is a NP and the syntactic subject of a passive clause [47]. For our illustrative sentence, nsubjpass(PyTables-1, built-3) means that "PyTables-1" is the syntactic subject of the verb "built-3".

The "nmod" used between two content words, as the preposition of one content word is now viewed as a case depending on its complement. In general, the "nmod" indicates some further adjunct relation specified by the case [48]. The nmod (library-9, built-3) means that "built-3" is nominal modifiers of "library-9". The "library-9" expresses further "on top of" relation to "built-3". Notice that some times there will appear a preposition after "nmod" like "nmod:at". This "at" means the complement use the preposition "at" to modify the Dependent.

The "xcomp" means open clausal complement of a verb or an adjective. These complements are always non-finite rather than adjuncts/modifiers. It is a predicative or clausal complement without its own subject. So the "xcomp" always expresses a new meaning or new relation. The xcomp means that "using-11" is open clausal complement of "built-3".

The "dobj" is the direct object of a VP, and is also a NP which is the accusative object of the verb.

3) Chunk Relation Triples: After the VP and NP chunking, we can chunk semantic relation like (concept, relation verb phrase, concept). Traditional rule-based chunking detects the chunk with morphological structure like (NP, VP, NP). This method not only misses many information, but also adds noise to the relation triples. For example, from sentence *"Firebird is written in C++, and is ultimately derived from the Borland InterBase 6.0 source code."*, traditional rule-based chunking can only extract relation triple (Firebird; is_written_in; C++). From sentence *Eclipse on your system can be used as a Java editor and a C++ editor*, relation triple (system; can_be_used_as; Java_editor) are extracted by traditional rule-based chunking using the morphological structure (NP, VP, NP). However, it is semantically wrong.

To tackle the above problems, we propose a method which incorporates the dependency parser with rule-based chunking. By analyzing the dependency generated by the dependency parse technique, the real subject of the verb in a complex sentence is determined. We assuming having a sentence S, Firstly we use the rule-based chunking to get all the VP and NP as below:

$$S = (NP_1, NP_2, ..., NP_i, ..., NP_m, VP_1, VP_2, ..., VP_j, ..., VP_n) \tag{1}$$

Where NP_i is the ith NP in S, VP_j is the jth VP in S, there are m NP and n VP in S.

$$NP_j = (n_{j1}, n_{j2}, ..., n_{jr}, ..., n_{jq}) \tag{2}$$

$$VP_i = (v_{i1}, v_{i2}, ..., v_{ik}, ..., v_{ip}) \tag{3}$$

Where n_{jr} is the rth term of NP_j, v_{ik} is the kth term of VP_i, we assume that NP_j contains q terms and VP_i contains p terms.

Secondly we use dependency parsing to get the dependency of all terms v and n. Due to the dynamic of natural language, there are too many different expressions for the same meaning. After we analyzing the dataset deeply, we propose many scenarios to deal with the sentences. We choose six scenarios in our system to show how *HDSKG Chunking* chunks the candidate relation triples.

Scenario 1: If we extract the nsubjpass (n_{12}, v_{13}), dobj(v_{13}, n_{22}) from S by dependency parsing, we can chunk (NP_1, VP_1, NP_2) as a relation triple. For example, in the sentence *"OpenRPT provides WYSIWYG editor"*, the "openRPT" is NP_1

TABLE II

DEPENDENCIES DESCRIPTION FOR SENTENCE: *"PyTables is built on top of the HDF5 library, using the Python language and the NumPy package."*

Dependency	Dependent	Governor	Semantic relationship between the words depicted by denpendency
nsubjpass	PyTables-1	built-3	"PyTables-1" is passive nominal subject of "built-3"
auxpass	is-2	built-3	"is-2" is passive auxiliary of built-3
det	the-7	library-9	"the-7" is determiner of "library-9"
nmod	library-9	built-3	"built-3" is nominal modifiers of "library-9"
xcomp	using-11	built-3	"using-11" is open clausal complement of "built-3"
dobj	language-14 package-18	using-11	"language-14" and "package-18" is direct object of "using-11"

and n_{11} , the "WYSIWYG editor" is NP_2 and "editor" is n_{22}, the "provides" is VP_1 and v_{11}. From dependency parsing we get nsubjpass (openRPT-1, provides-2) and dobj (provides-2, editor-4), so we can chunk relation triple (OpenRPT; provides; WYSIWYG_editor).

Scenario 2: If we extract the dependency like nsubjpass (n_{11}, v_{12}), nmod (v_{12}, n_{23}) in sentence *"HSQLDB is supported by many Java frameworks."*, where n_{11} is "HSQLDB", v_{12} is "supported", n_{23} is "frameworks". We should chunk relation triple (HSQLDB; is supported by; many_Java_frameworks). Here we assume adjective "many" can't be ignored like DT (article), because "many Java frameworks" is not equal to "all Java frameworks".

Scenario 3: In some cases, for example "webkit is developed by Intel at the Intel Open Source Technology Center.", we get dependency like nsubjpass(webkit-1, developed-3), nmod:agent(developed-3, Intel-5) and nmod:at(developed-3, Center-12), we chunk two relation triples (webkit; is_developed_by; Intel) and (webkit; is_developed_at; Intel_Open_Source_Technology_Center). Notice that with the preposition of dependency "nmod:at", the preposition of VP_1 "is_developed_by" can be changed to "at" while chunking the second relation triples.

Scenario 4: If there is an "and" between two NPs, we will chunk 2 relation triples with different subjects or objects. For example, from sentence "flashcanvas renders shapes and images.", we will chunk relation triples as:

- (flashcanvas; renders; shapes)
- (flashcanvas; renders; images)

Scenario 5: If there is a dependency "xcomp" extracted from S, we will give an additional dependency "nsubjpass" to the dependencies of the sentence. For example, the sentence *"PyTables is built on top of the HDF5 library, using the Python language and the NumPy package."* as shown in section. III-D2, we extract two dependencies – xcomp(using-11, built-3) and nsubjpass(PyTables-1, built-3) from this sentence, we will add an additional dependency nsubjpass(PyTables-1, using-11) to this sentence. It because that as a open clausal complement of a verb or an adjective, the "xcomp" always expresses a new meaning or new relation. Thereby we can chunk relation triples as:

- (PyTables; is_built_on_top_of; HDF5_library)
- (PyTables; using; Python_language)
- (PyTables; using; NumPy_package)

Scenarios 6: If S contains VVP as mentioned in section. III-D1, we don't chunk two relation triples like **Scenarios 5** even there is a "xcomp" extracted from S. This is because the two verbs of VVP share the same object. Instead, we chunk only one relation triple and leave the two verbs together in VVP as the relation verb of the relation triple. For example "Joone is used to build neural networks.", we will chunk relation triple as:

- (Joone; is_used_to_build; neural_networks)

By the above method, from "Firebird is written in C++, and is ultimately derived from the Borland InterBase 6.0 source code.", we can get dependency nsubjpass(Firebird-1, derived-10), so we will chunk relation triples as:

- (Firebird; is_ultimately_derived_from; Borland_InterBase_6.0_source_code)
- (Firebird; is_written_in; C++)

From sentence *"Eclipse on your system can be used as a Java editor."* we can get nsubjpass(Eclipse-1, used-7). Thus we can chunk relation triples as:

- (Eclipse; can_be_used_as; Java_editor)
- (Eclipse; can_be_used_as; C++_editor)

E. Domain Relevance Estimation of Candidate Relations Triples

From *HDSKG Chunking* we present in section. III-D, we harvest many candidate relation triples (subject, verb phrase, object). But some of the relation triples are not related to our target domain. For example, the relation triple (Values; may be enclosed in; quotes), this is a correct chunking but meaningless for the software engineering domain. In this section, we use a machine learning method with advanced features to estimate the domain relevance of the candidate relation triples generated by *HDSKG Chunking*.

1) Feature Engineering: There are many properties beyond simple term statistics which can be extracted from the candidate relation triples to estimate the domain relevance of them. We use the features list in Tab. III to represent each candidate relation triples.

For the complex features we provide a brief description below.

Text Features: This feature type regards the basic text properties of the cr (candidate relation triples) at the term level. The POS fraction features capture the distribution of POS tags in the cr and uses following definition:

TABLE III
THE FEATURES FOR CANDIDATE TRIPLES CLASSIFICATION

#	Name	Gloss
		Text features
1	#terms_cr	Number of,terms of cr (candidate relation triples)
2-5	POS fractions_subj	Fraction of verbs, nouns, adjectives, others,of subject
6-9	POS fractions_obj	Fraction of verbs, nouns, adjectives, others,of object
10-13	POS fractions_vp	Fraction of verbs, nouns, adjectives, others,of verb phrase
14-17	POS fractions_cr	Fraction of verbs, nouns, adjectives, others,of cr
		Corpus features
18	#mention_subj	Number of subject mentioned in whole cr corpus
19	#mention_obj	Number of object mentioned in whole cr,corpus
20	#mention_vp	Number of verb phrase mentioned in whole cr corpus
21	#mention_cr	Number of cr mentioned in whole cr corpus
22	suport_subj_obj	The proportion that subject and object occur simultaneously in whole cr corpus
23	conf_subj	The proportion of the occurrence of object when subject occurs in whole cr corpus
24	conf_obj	The proportion of the occurrence of subject when object occurs in whole cr corpus
		Concept features
25	subj_tfidf,	TF-IDF of subject in tagWiki,in whole text materials
26	obj_tfidf	TF-IDF of object in tagWiki,in whole text materials
27	sum_tfidf	Sum of TF-IDF of subject and object in whole text materials
28	average_tfidf	Average TF-IDF of subject,and object in whole text materials
29	%domain_keyword_subj	The proportion,of number of domain keywords in subject
30	%domain_keyword_obj	The proportion,of number of domain keywords in object
31	%domain_keyword_subj_obj	The proportion,of number of domain keywords in subject and object
		Source features
32	ss_position	Position of ss (source sentence) in stm (source text material)
33	cr_position	Position of cr in ss
34	From_subj_obj_stm	If this cr extract from stm which the title involve the subject or object
35	#terms_ss	Number of terms in ss which cr extracted from
36	start_index_cr	Start index of the first term of cr in ss
37	end_index_cr	End index of the first term of cr in ss

- noun : the tokens which POS tags are NN, NNS, NNP, or NNPS.
- verb : the tokens which POS tags are VB, VBD, VBG, VBN, VBP, or VBZ.
- adj : the tokens which POS tags are JJ, JJR, or JJS.

Corpus Features: These features present the statistic relation between current cr and whole cr corpus. If the cr is extracted from source webpages many times, this cr appear to be relevant to target domain with higher potential. The $conf_{subj}$ denotes the percentage of the occurrence of object when subject occurs, the $conf_{obj}$ denotes the percentage of the occurrence of subject when object occurs. And the support indicates the percentage that subject and object occur simultaneously in the corpus. These association rule features present relation between the subject and object. If these two concepts can be found frequently from stm (source text material) simultaneously, the relation triples of these two concepts are high potential relevant to target domain.

Concept Features: These features regard the quality of the concepts. TF-IDF is a numerical statistic approach which can determine the words relevance in a corpus of documents [49], [50]. TF (Term Frequency) means the raw frequency of a term in a document. IDF (Inverse document frequency) is the logarithmically scaled inverse fraction of the documents that contain the word, obtained by dividing the total number of documents by the number of documents containing the term, and then taking the logarithm of that quotient. Here is the formula for unnormalized weight of $term_i$ in $document_j$ in a corpus of D (documents):

$$TF{-}IDF_{weight_{i,j}} = frequency_{i,j} * log_2(D/document_{freq_i}) \quad (4)$$

Thus the features $subj\_tfidf$, obj_tfidf, sum_tfidf and $average\_tfidf$ can indicate the domain relevance of subjects and objects. If most of the terms in a subject are very general like “values”, “process”, etc., the value of the $subj\_tfidf$ will be very low. Otherwise if the subject contains unique and distinctive terms like “c++ library”, “MySQL relational database”, the $subj\_tfidf$ value will be high.

We also prepare the domain lexicon which contains the specialized vocabularies of the target domain for estimating the domain relevance of the candidate relation triples. Obviously the candidate relation triples which contain more specialized vocabularies of the target domain show higher domain relevance [51].

Source Features: Here, we refer to features of the cr depended on the stm (source text material) and ss (source sentence) [52]. For the $crposition$ and $ss_{position}$, these two features mean that the cr extracted from the front part of the stm and the front part of the ss is high relevance to target domain. The position refer to the sequence of a cr in the stm or the ss. On the other hand, if a cr is

TABLE IV
EXAMPLE OF RELATION TRIPLES LABEL

Candidate Relation Triples	Label
(Java; supports; features)	False
(row; is_represented_as; list)	False
(Values; may_be_enclosed_in; quotes)	False
(Java; was_originally_developed_by; James_Gosling)	True
(Java; was_originally_developed_at; Sun_Microsystems)	True
(Eclipse; is; open-source_ide_platform)	True

extracted from the webpage in which the topic contains the terms of concepts, it shows high domain relevance. For the long sentence, $cr\_position$, $start\_index_{cr}$ and end_index_{cr} indicate that the cr extracted from the front part of a long ss shows high domain relevance.

2) Label Data: We design the estimation of domain relevance as a binary-class classification problem to solve. The supervised learning of classification requires the labeled data for training. We manually mark candidate relation triples by domain experts as training data to boost the learning process. Tab. IV shows a example about how to label the positive and negative relation triples.

3) Semi-Supervised SVM Classifier Learning: The workload while using a system to a new domain determines the applicability and the portability of a system. Obviously, labeling new training data is the most labor consuming part. To reduce the labor force and improve the prediction accuracy, we select the best single classifer – SVM classifer with a self learning structure to do the classification [27]. In the first iteration, we use the expert labeled relation triples as the training data and get a trained SVM classifier. Then, we input the unlabel relation triples to the trained SVM classifier and get the labels (or classes) of them. The classifier not only classify the relation triples, but also provide a confidence level to every classified relation triple to show the probability of current classification. From the second iteration, self-training enrich training data with the labeled relation triples which $confidence > 0.9$ or $confidence < 0.1$ to train a new classifier for the next iteration. The iteration will terminate if the difference of accuracy lower than a threshold or achieved the preseted iteration time. Then we use the trained SVM classifier at the end of the iterations to classify all the unlabeled relation triples, and use the relation triples in the positive classes to construct the knowledge graph of target domain.

IV. EXPERIMENT

In this section, a case study is proposed to illustrate the *HDSKG* system process for acquiring the domain specific relation triples and constructing a knowledge graph. *tagWiki* of Stack Overflow is used to extract the knowledge graph and some *tagWiki* documents are chose to evaluate our system.

A. Experiment Setup

1) Data Acquisition: We download the data dump of Stack Overflow until Mar, 2015 from Stack Overflow official data dump[1].

Source Text Materials Acquisition: We use the tagWiki of Stack Overflow provided in a html format as our source text materials and delete some content which contain few relation triples such as "frequent question", "Useful links", etc. Like the tagWiki of tag "Firebird", the input is the html file of the webpage of "Firebird"[2]. As a result, we acquire 20534 documents and split 97454 sentences from the documents.

Domain Lexicon: We use the tags of Stack Overflow as the domain lexicon. Finally, 27620 tags is acquired, these tags cover the popular programming languages(e.g., java, python, php), frameworks(e.g., GWTP, Pallet, AnyEvent), libraries(e.g., OmniFaces, PHPPowerPoint), tools(e.g., opengl, SubGit, MvcSiteMapProvider) and some frequently used terminology(e.g., cookies, ip) of software engineering domain.

Ground Truth of the Relation Triples: From the tagWiki of Stack Overflow, we choose 47 tagWikis by random as our evaluation materials. Four experts are asked to extract the relation triples from the webpages of the 47 tagWikis. The four experts have the background in software engineering and three of them have ever published knowledge graph or ontology construction related papers. To ensure the precision, we give the hyperlinks to the experts rather than the pre-processed documents. Only the relation triples which extract by at least 2 experts are involved in our ground truth. Finally we get 96 relation triples from the webpages of tagWiki, but we ignore 5 of them due to their source sentences from webpages are different from the source sentences from data dump. We use the rest 91 relation triples as the ground truth to evaluate the performance of *HDSKG*.

Involved Tools: The *Stanford CoreNLP* is used to do the NLP make up and dependency parse [53]. The tf-idf library of *Gensim* is used to compute tf-idf [54]. The *nltk* is used to do the VP, NP chunking [55]. *neo4j* is used to store the finally relation triples for supporting the knowledge graph searching and result visualization [56].

B. Comparison Method openIE

We leverage the popular open information extraction tool - *open IE* of Stanford as the comparison method [10], [13] due to it is close to *HDSKG*.

But there are also some differences between our tool and openIE. First is the method to extract relation triples from stm. openIE only fucus on extraction of relation triples, but ignore the subject and object should be entities. For example, from sentence "Born in a small town, she took the midnight train going anywhere.", openIE will extract relation triple (she; took; midnight_train). Here "she" is not an entity. *HDSKG* identifies the pronouns in a sentence and replace by suitable nouns. If some pronouns can't be replaced, we will not extract the

[1]https://archive.org/details/stackexchange
[2]http://stackoverflow.com/tags/firebird/info

corresponding relations, because the relation triples between the pronouns and the nouns is useless to knowledge graph. The other diffidence is that openIE only extract the relation triples but can't estimate the domain reverence of them. So there are many redundancy relation triples will be extracted by openIE.

C. Evaluation and Results Analysis

1) Classifier Comparison and Accuracy Evaluation of Iterations: We do 10-folder cross-validation to evaluation the accuracy in each iteration. The 1000 labeled data (contain 467 positive data and 523 negative data) are separated in two parts, 900 of them are used as training data and 100 of them are used as testing data. In every cross-validation, the prediction confidences of every prediction data are recorded. After 10 times cross-validation, only the prediction data which $confidence > 0.9$ or $confidence < 0.1$ in all the 10 times cross-validation are chose to append to training data for the next iteration. Fig.5 is the accuracy of different classifiers in each iteration. It shows the performance of the DNN is the most sensitive classifier to the number of training data, so if we can get large number of training data, the best classifier should be DNN. SVM linear classifier achieves the highest accuracy with the 6 times self-training iterations. The accuracy increase from 0.71 to 0.76, which better than random forest (estimators number = 500) and 4 layers DNN with the $hidden\_units = [200, 400, 100, 50]$.

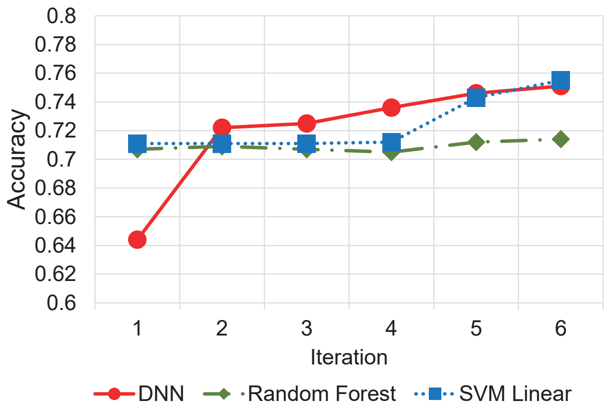


Fig. 5. Accuracy of Classifiers in each Iteration

2) Features Contribution Analysis: For exploration the effect of different features set, we train 5 SVM classifiers using different features groups. Each classifier only uses text features, corpus features, concept features, source features and all features respectively. As shown in Fig.5, the classifier with all features performs best. From Fig.5 we also can infer that all these features we design are useful in estimation the domain relevance of relation triples.

3) Performance Comparison: We further explore the performance of *HDSKG*. Tab. V shows the extractions of traditional Rule Based Chunking, openIE, *HDSKG Chunking* and *HDSKG Domain Relevance Estimation*. This example indicates that compare with traditional Rule Based Chunking, *HDSKG Chunking* can get more relation triples which

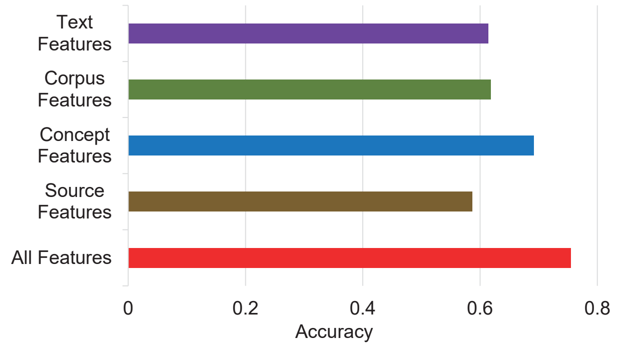


Fig. 6. Accuracy of Classifiers with Different Features

can't extract by simple morphology matching. The traditional Rule Based Chunking can only chunk the relation triples (PyTables; is_built_on_top_of; HDF5_library) from sentence *"PyTables is built on top of the HDF5 library, using the Python language and the NumPy package."*, but can't chunk (PyTables; using; NumPy_package) because the morphology of this relation triple is not continuous (NP,VP,NP). With the using of dependency parsing, we can get the xcomp (using-11, built-3), dobj (using-11, language-14), and dobj (using-11, package-18). These dependencies mean that "using-11" is open clausal complement of "built-3", and the object of "using-1" is "language-14" and "package-18". From scenario 5 of *HDSKG Chunking*, we can chunk (PyTables; using; NumPy_package) and (PyTables; using; Python_language).

openIE derives the latent relation triple (PyTables; using; Python_language), but ignores the paralleled relation triple (PyTables; using; NumPy_package). But this relation triple captures by *HDSKG* accurately by the scenario 4 designed in our system. In addition to above shortcoming, openIE also generates many redundant relation triples like (PyTables; is; built). *HDSKG* doesn't make this mistake due to our VP and NP chunking can identify the verb phrase "is built on top of".

For the sentence "Jansi is a small java library that allows you to use ANSI escape codes to format your console output which works even on windows.", all the above tree methods extract the wrong relation triple (ANSI; escape; codes). Then we can use the *HDSKG Domain Relevance Estimation* to filter this irrelevant relation triple. From the Tab. V we can see the *HDSKG Domain Relevance Estimation* can reserve the correct relation triples and filter out the irrelevant relation triples accurately.

Compare with the above 2 methods, *HDSKG Chunking* extracts both apparent and latent relation triples, and the domain relevance estimation of *HDSKG* filters the relation triples which less relevant to the target domain.

Precision, recall and f-measure are used as evaluation metrics of the extractions of *HDSKG*. Precision is the ratio of the correct relation triples to all extractions, this metric is seen as a measure of quality. Recall is the ratio of the correct relation triples to all ground truth relation triples, which is a measure of completeness. F-measure is computed by:

$$F = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \qquad (5)$$

TABLE V
RELATION TRIPLES EXTRACTED FROM DIFFERENT EXTRACTION METHODS

Extraction Method \ Content of Sentence	PyTables is built on top of the HDF5 library, using the Python language and the NumPy package.	Jansi is a small java library that allows you to use ANSI escape codes to format your console output which works even on windows.
Rule Based Chunking Extraction	(PyTables; is_built_on_top_of; HDF5_library)	(Jansi; is; small_java_library) (ANSI; escape; codes)
openIE Extraction	(PyTables; using;,Python_language) (PyTables; is; built) (PyTables; is_built_on_top_of; HDF5_library)	(Jansi; is; small) (ANSI; escape; codes)
HDSKG Chunking Extraction	(PyTables; using; Python_language) (PyTables; using; NumPy_package) (PyTables; is_built_on_top_of; HDF5_library)	(Jansi; is; small_java_library) (ANSI; escape; codes)
HDSKG Domain Relevance Estimation Extraction	(PyTables; using; Python_language) (PyTables; using; NumPy_package) (PyTables; is_built_on_top_of; HDF5_library)	(Jansi; is; small_java_library)
Ground Truth	(PyTables; using; Python_language) (PyTables; using; NumPy_package) (PyTables; is_built_on_top_of; HDF5_library)	(Jansi; is; small_java_library)

Fig. 7 shows the performance of different extracting methods, compare with openIE, *HDSKG domain relevance estimation* achieves the highest precision 0.77 and *HDSKG Chunking* get the highest recall 0.74.

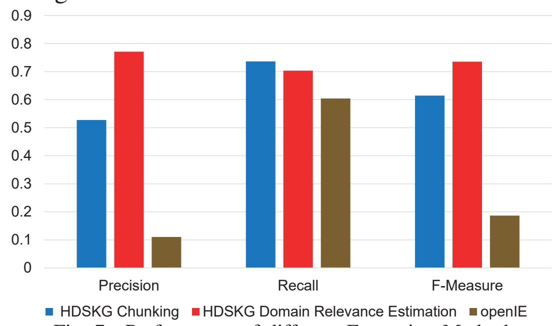


Fig. 7. Performance of different Extraction Methods

According to the result, even we lose 0.04 recall by estimation of domain relevance , the precision is be improved 0.25. The recall performance of the openIE is 0.6, which is close to the recall of *HDSKG Domain Relevance Estimation*, but this recall of openIE is at the expense of reduction of the precision by generating more irrelevance relation triples.

Compare with the openIE, *HDSKG Chunking* produces less relation triples with higher precision. *HDSKG Domain Relevance Estimation* estimates the domain relevance with small loss of recall but improves the precision conspicuously. Fig. 8 is part of the knowledge graph generate by *HDSKG* in the software engineering domain.

V. DISCUSSION

So far, we have extracted a knowledge graph of software engineering domain from the tagWiki of Stack Overflow. In this section, we discuss the implications of our research for knowledge graph extraction, information retrieval of software engineering domain, and knowledge representation.

A. Implications for Knowledge Graph Extraction

As shown in section III-D, *HDSKG Chunking* incorporates the dependency parser with rule-based method and extracts the candidate relation triples in the natural language materials with high precision and low recall. The 6 scenarios we present in section. III-D3 cover most of the occasions in natural language expressions. It ensures the *HDSKG* can easy to be used in other natural language materials, e.g., healthcare, industrial design, etc.

For the estimation of domain relevance part of *HDSKG*, the features we designed are efficient to distinguish domain relevant relation triples as shown in section. IV-C2. *HDSKG* just needs to change the domain lexicon to the domain you want and label few training data, with the self-training SVM classifier, *HDSKG* will construct the knowledge graph of your target domain automatically.

B. Implications for Information Retrieval in Software Engineering Domain

With the knowledge graph of software engineering domain, we can change the traditional information retrieval way.

Firstly, a direct answer search engine can be constructed [57], [58]. As it shown in Fig. 8, there are 2 kinds of relations, they are “is a” and “object property”. “is a” is the taxonomic relations which indicate a containment relation between 2 concepts [59]. For example, the taxonomic relation like “pi-db” -> “relational_database” -> “database” in Fig. 8. “pi-db” is subclass of “relational_database”, and “relational_database” is subclass of “database”. So we can derive if we have a sentence like "The employees’ information is put in firebird", this sentence "The employees’ information is put in database" is also right. The “object property” is the non-taxonomic relations, these relations present some properties of the concepts. So if there is a query like: *“Which database can run on Unix and be written in C++”* From the query we can extract the VP “run on” and “be written in”, and the concepts “database”, “Unix” and “C++”. Then the “firebird” will be recommended directly. This direct answer search engine can answer many questions which ask about a specific concept and

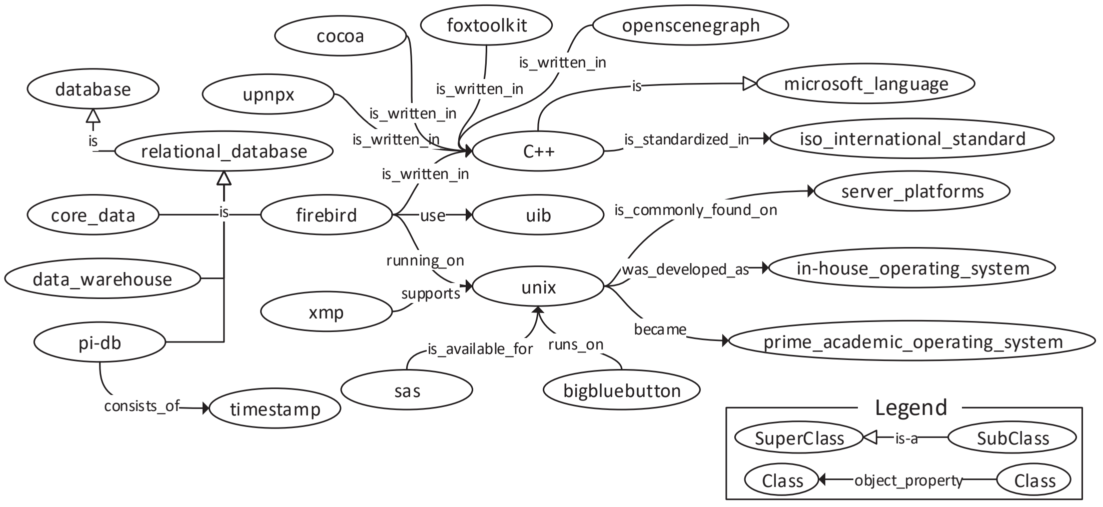


Fig. 8. An Example of Knowledge Graph Generated by *HDSKG*

help developers to find available technologies without reading many webpages.

We can also use this knowledge graph to the semantic search engine because it links the concepts across query key words, documents, databases, webpages, etc. via semantic modeling.. For example, the phrase “firebird is written in C++” and the phrase “C++ is microsoft language” are extracted from different webpages. There is no link between the concept “firebird” and the concept “microsoft language”. But in the knowledge graph, these two concepts connect to “C++”. So in the search results, the ranking algorithms should concern the weight to “microsoft language” related webpages when search “firebird”.

C. Implications for Knowledge Representation

Our knowledge graph is also a structural knowledge representation which can be used to do data documentation and enrich search result pages. Actually Stack Overflow paid effort on it already. There is a new online function which is still under beta test named documentation[1]. In this part, Stack Overflow organizes the information of each tag structural like “different version”, “important functions” and “useful code snippets”, etc. Compare to the traditional tagWiki, developer can easy to locate the information they want. But this documentation is contributed by the users of Stack Overflow manually, it’s low efficient and subjective. Our knowledge graph can assist to enrich the documentation. Because it is extracted from the materials which is contributed by many developers, with the “corpus features” we used to the classifier in section. III-E1, the subjective relation triples can be removed. And with the result like the 5 “is written in” relations in Fig. 8, we can enrich the documentation of “C++” with the tools or libraries which are written by “C++”.

Our knowledge graph also can be used to enrich the search result. For example, if you search “iso standard”, as the result in Fig. 8, we will recommend “C++” which is standardized by “iso standard”.

[1]http://stackoverflow.com/documentation

VI. Conclusion and Future Work

Knowledge graph integrates structural information of concepts across multiple information sources, and links these concepts together. It is useful in many domains such as search result ranking, recommendation, exploratory search, etc. In this paper, an automatic method named *HDSKG* is proposed to discover domain specific concepts and their relation triples from the content of webpages. By incorporation of the dependency parser with rule-based method, we get the candidate relations triple candidates with high precision and recall. And by utilizing the self-training SVM classifier, we can estimate the domain relevance of the candidate relation triples.

However, even we already proposed six different particular scenarios, due to the dynamic of natural language, there are still some scenarios *HDSKG* can’t deal with very well. For example, the “of” in NP sometimes make the concept very long like “Middleware is a framework of hooks into Django’s request or response processing.”. But if we delete the content after “of”, some mistakes will be caused like “Windows is a family of graphical operating-systems" will extract (Windows; is; family), it is obviously incorrect. There are also some concepts which refer to same meaning but different expressions, like “python” and “python_language”, “operation_system” and “operating_system”, etc). In the future, we plan to do the concepts and verb phrase clustering to unify the expressions and optimize our system to deal with more particular scenarios.

VII. Acknowledgements

This work was conducted within the Rolls-Royce@NTU Corporate Lab with support from the National Research Foundation (NRF) Singapore under the Corp Lab@University Scheme. This research is partially supported by the National Research Foundation, Prime Minister’s Office, Singapore under its IDM Futures Funding Initiative.

References

[1] C. Chen, Z. Xing, and L. Han, "Techland: Assisting technology landscape inquiries with insights from stack overflow," *32nd ICSME. IEEE*, 2016.

[2] K. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor, "Freebase: a collaboratively created graph database for structuring human knowledge," in *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*. ACM, 2008, pp. 1247–1250.

[3] F. Niu, C. Zhang, C. Ré, and J. W. Shavlik, "Deepdive: Web-scale knowledge-base construction using statistical learning and inference." *VLDS*, vol. 12, pp. 25–28, 2012.

[4] F. M. Suchanek, G. Kasneci, and G. Weikum, "Yago: a core of semantic knowledge," in *Proceedings of the 16th international conference on World Wide Web*. ACM, 2007, pp. 697–706.

[5] J. Fan, D. Ferrucci, D. Gondek, and A. Kalyanpur, "Prismatic: Inducing knowledge from a large scale lexicalized relation resource," in *Proceedings of the NAACL HLT 2010 first international workshop on formalisms and methodology for learning by reading*. Association for Computational Linguistics, 2010, pp. 122–127.

[6] R. Padhye, D. Mukherjee, and V. S. Sinha, "Api as a social glue," in *Companion Proceedings of the 36th International Conference on Software Engineering*. ACM, 2014, pp. 516–519.

[7] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. Ives, "Dbpedia: A nucleus for a web of open data," in *The semantic web*. Springer, 2007, pp. 722–735.

[8] J. Zhu, Z. Nie, X. Liu, B. Zhang, and J.-R. Wen, "Statsnowball: a statistical approach to extracting entity relationships," in *Proceedings of the 18th international conference on World wide web*. ACM, 2009, pp. 101–110.

[9] F. Wu and D. S. Weld, "Open information extraction using wikipedia," in *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, 2010, pp. 118–127.

[10] O. Etzioni, M. Banko, S. Soderland, and D. S. Weld, "Open information extraction from the web," *Communications of the ACM*, vol. 51, no. 12, pp. 68–74, 2008.

[11] Y. Lin, Z. Liu, M. Sun, Y. Liu, and X. Zhu, "Learning entity and relation embeddings for knowledge graph completion." in *AAAI*, 2015, pp. 2181–2187.

[12] A. Carlson, J. Betteridge, B. Kisiel, B. Settles, E. R. Hruschka Jr, and T. M. Mitchell, "Toward an architecture for never-ending language learning." in *AAAI*, vol. 5, 2010, p. 3.

[13] G. Angeli, M. J. Premkumar, and C. D. Manning, "Leveraging linguistic structure for open domain information extraction," *Linguistics*, no. 1/24, 2015.

[14] M. Pasca, D. Lin, J. Bigham, A. Lifchits, and A. Jain, "Organizing and searching the world wide web of facts-step one: the one-million fact extraction challenge," in *AAAI*, vol. 6, 2006, pp. 1400–1405.

[15] J. C. de Almeida Biolchini, P. G. Mian, A. C. C. Natali, T. U. Conte, and G. H. Travassos, "Scientific research ontology to support systematic review in software engineering," *Advanced Engineering Informatics*, vol. 21, no. 2, pp. 133–151, 2007.

[16] P. Wongthongtham, E. Chang, T. Dillon, and I. Sommerville, "Development of a software engineering ontology for multisite software development," *IEEE Transactions on Knowledge and Data Engineering*, vol. 21, no. 8, pp. 1205–1217, 2009.

[17] S. Subramanian, L. Inozemtseva, and R. Holmes, "Live api documentation," in *Proceedings of the 36th International Conference on Software Engineering*. ACM, 2014, pp. 643–652.

[18] H. S. Delugach, "Specifying multiple-viewed software requirements with conceptual graphs," *Journal of Systems and Software*, vol. 19, no. 3, pp. 207–224, 1992.

[19] H.-J. Happel and S. Seedorf, "Applications of ontologies in software engineering," in *Proc. of Workshop on Sematic Web Enabled Software Engineering"(SWESE) on the ISWC*. Citeseer, 2006, pp. 5–9.

[20] M. Banko, M. J. Cafarella, S. Soderland, M. Broadhead, and O. Etzioni, "Open information extraction from the web." in *IJCAI*, vol. 7, 2007, pp. 2670–2676.

[21] J. Betteridge, A. Carlson, S. A. Hong, E. R. Hruschka Jr, E. L. Law, T. M. Mitchell, and S. H. Wang, "Toward never ending language learning." in *AAAI Spring Symposium: Learning by Reading and Learning to Read*, 2009, pp. 1–2.

[22] O. Etzioni, M. Cafarella, D. Downey, S. Kok, A.-M. Popescu, T. Shaked, S. Soderland, D. S. Weld, and A. Yates, "Web-scale information extraction in knowitall:(preliminary results)," in *Proceedings of the 13th international conference on World Wide Web*. ACM, 2004, pp. 100–110.

[23] G. Kasneci, M. Ramanath, F. Suchanek, and G. Weikum, "The yago-naga approach to knowledge discovery," *ACM SIGMOD Record*, vol. 37, no. 4, pp. 41–47, 2009.

[24] R. Krishnamurthy, Y. Li, S. Raghavan, F. Reiss, S. Vaithyanathan, and H. Zhu, "Systemt: a system for declarative information extraction," *ACM SIGMOD Record*, vol. 37, no. 4, pp. 7–13, 2009.

[25] N. Nakashole, M. Theobald, and G. Weikum, "Scalable knowledge harvesting with high precision and high recall," in *Proceedings of the fourth ACM international conference on Web search and data mining*. ACM, 2011, pp. 227–236.

[26] J. Zhu, H. Wang, and B. Shen, "Software. zhishi. schema: A software programming taxonomy derived from stackoverflow."

[27] J. Zhu, B. Shen, X. Cai, and H. Wang, "Building a large-scale software programming taxonomy from stackoverflow," in *SEKEąŕ2015: 27th International Conference on Software Engineering and Knowledge Engineering*, pp. 391–396.

[28] A. Barua, S. W. Thomas, and A. E. Hassan, "What are developers talking about? an analysis of topics and trends in stack overflow," *Empirical Software Engineering*, vol. 19, no. 3, pp. 619–654, 2014.

[29] C. Chen and Z. Xing, "Mining technology landscape from stack overflow," *10th ESEM. IEEE/ACM*, 2016.

[30] H. Li, I. Councill, W.-C. Lee, and C. L. Giles, "Citeseerx: an architecture and web service design for an academic document search engine," in *Proceedings of the 15th international conference on World Wide Web*. ACM, 2006, pp. 883–884.

[31] P. Szekely, C. A. Knoblock, J. Slepicka, A. Philpot, A. Singh, C. Yin, D. Kapoor, P. Natarajan, D. Marcu, K. Knight *et al.*, "Building and using a knowledge graph to combat human trafficking," in *International Semantic Web Conference*. Springer, 2015, pp. 205–221.

[32] J. Pujara, H. Miao, L. Getoor, and W. Cohen, "Knowledge graph identification," in *International Semantic Web Conference*. Springer, 2013, pp. 542–557.

[33] C. Z. Mooney, R. D. Duval, and R. Duval, *Bootstrapping: A nonparametric approach to statistical inference*. Sage, 1993, no. 94-95.

[34] M. Schmitz, R. Bart, S. Soderland, O. Etzioni *et al.*, "Open language learning for information extraction," in *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*. Association for Computational Linguistics, 2012, pp. 523–534.

[35] A. Fader, S. Soderland, and O. Etzioni, "Identifying relations for open information extraction," in *Proceedings of the Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 2011, pp. 1535–1545.

[36] S. L. Abebe and P. Tonella, "Towards the extraction of domain concepts from the identifiers," in *2011 18th Working Conference on Reverse Engineering*. IEEE, 2011, pp. 77–86.

[37] J. Hoffart, F. M. Suchanek, K. Berberich, and G. Weikum, "Yago2: A spatially and temporally enhanced knowledge base from wikipedia," *Artificial Intelligence*, vol. 194, pp. 28–61, 2013.

[38] J. Hirschberg and C. D. Manning, "Advances in natural language processing," *Science*, vol. 349, no. 6245, pp. 261–266, 2015.

[39] A. M. Rush, S. Chopra, and J. Weston, "A neural attention model for abstractive sentence summarization," *arXiv preprint arXiv:1509.00685*, 2015.

[40] H. He, K. Gimpel, and J. Lin, "Multi-perspective sentence similarity modeling with convolutional neural networks," in *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, 2015, pp. 1576–1586.

[41] C. D. Manning, M. Surdeanu, J. Bauer, J. R. Finkel, S. Bethard, and D. McClosky, "The stanford corenlp natural language processing toolkit." in *ACL (System Demonstrations)*, 2014, pp. 55–60.

[42] K. Toutanova and C. D. Manning, "Enriching the knowledge sources used in a maximum entropy part-of-speech tagger," in *Proceedings of the 2000 Joint SIGDAT conference on Empirical methods in natural language processing and very large corpora: held in conjunction with the 38th Annual Meeting of the Association for Computational Linguistics-Volume 13*. Association for Computational Linguistics, 2000, pp. 63–70.

[43] J. Nivre, M.-C. de Marneffe, F. Ginter, Y. Goldberg, J. Hajic, C. D. Manning, R. McDonald, S. Petrov, S. Pyysalo, N. Silveira *et al.*, "Universal dependencies v1: A multilingual treebank collection," in *Pro-*

ceedings of the 10th International Conference on Language Resources and Evaluation (LREC 2016), 2016.

[44] S. Schuster and C. D. Manning, "Enhanced english universal dependencies: An improved representation for natural language understanding tasks," in *Proceedings of the 10th International Conference on Language Resources and Evaluation*, 2016.

[45] C. Grover and R. Tobin, "Rule-based chunking and reusability," in *Proceedings of the Fifth International Conference on Language Resources and Evaluation (LREC 2006)*, 2006.

[46] T. Zhang, F. Damerau, and D. Johnson, "Text chunking based on a generalization of winnow," *Journal of Machine Learning Research*, vol. 2, no. Mar, pp. 615–637, 2002.

[47] M.-C. De Marneffe and C. D. Manning, "Stanford typed dependencies manual," Technical report, Stanford University, Tech. Rep., 2008.

[48] M.-C. De Marneffe, T. Dozat, N. Silveira, K. Haverinen, F. Ginter, J. Nivre, and C. D. Manning, "Universal stanford dependencies: A cross-linguistic typology." in *LREC*, vol. 14, 2014, pp. 4585–92.

[49] J. Leskovec, A. Rajaraman, and J. D. Ullman, *Mining of massive datasets*. Cambridge University Press, 2014.

[50] J. Ramos, "Using tf-idf to determine word relevance in document queries," in *Proceedings of the first instructional conference on machine learning*, 2003.

[51] P. Velardi, M. Missikoff, and R. Basili, "Identification of relevant terms to support the construction of domain ontologies," in *Proceedings of the workshop on Human Language Technology and Knowledge Management-Volume 2001*. Association for Computational Linguistics, 2001, p. 5.

[52] N. Voskarides, E. Meij, M. Tsagkias, M. de Rijke, and W. Weerkamp, "Learning to explain entity relationships in knowledge graphs," *In Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and The 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing (ACL-IJCNLP 2015)*, p. 11, 2015.

[53] C. D. Manning, M. Surdeanu, J. Bauer, J. Finkel, S. J. Bethard, and D. McClosky, "The Stanford CoreNLP natural language processing toolkit," in *Association for Computational Linguistics (ACL) System Demonstrations*, 2014, pp. 55–60. [Online]. Available: http://www.aclweb.org/anthology/P/P14/P14-5010

[54] R. Řehůřek and P. Sojka, "Software Framework for Topic Modelling with Large Corpora," in *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*. Valletta, Malta: ELRA, May 2010, pp. 45–50, http://is.muni.cz/publication/884893/en.

[55] S. Bird, "Nltk: the natural language toolkit," in *Proceedings of the COLING/ACL on Interactive presentation sessions*. Association for Computational Linguistics, 2006, pp. 69–72.

[56] J. Webber, "A programmatic introduction to neo4j," in *Proceedings of the 3rd annual conference on Systems, programming, and applications: software for humanity*. ACM, 2012, pp. 217–218.

[57] Y. WHAN KIM and J. H. Kim, "A model of knowledge based information retrieval with hierarchical concept graph," *Journal of Documentation*, vol. 46, no. 2, pp. 113–136, 1990.

[58] M. Joshi, U. Sawant, and S. Chakrabarti, "Knowledge graph and corpus driven segmentation and answer inference for telegraphic entity-seeking queries." in *EMNLP*, 2014, pp. 1104–1114.

[59] M. A. Hearst, "Automatic acquisition of hyponyms from large text corpora," in *Proceedings of the 14th conference on Computational linguistics-Volume 2*. Association for Computational Linguistics, 1992, pp. 539–545.