Proceedings of the 2014 Federated Conference on Computer Science and Information Systems pp. 255–259
DOI: 10.15439/2014F343
ACSIS, Vol. 2

Semantic sentence structure search engine

Nikita Gerasimov
Nothern (Arctic) Federal University,
Severnaya Dvina Emb. 17,
Arkhangelsk, Russia; 163002;
Email: n.gerasimov@narfu.ru

Maxim Mozgovoy
The University of Aizu, Tsuruga,
Ikki-machi, Aizu-Wakamatsu,
Fukushima, 965-8580 Japan
Email: mozgovoy@u-aizu.ac.jp

Alexey Lagunov
Nothern (Arctic) Federal University,
Severnaya Dvina Emb. 17,
Arkhangelsk, Russia; 163002;
Email: a.lagunov@narfu.ru

***Abstract*—Many of current web search engines rely on inverted index-based data structures as document information store. Since and inverted index is a map from individual document words to their respective locations, such data structure destructs semantic links between the words, and thus does not support structural user queries. In other words, such systems can only find the documents that contain user-specified words. In this paper we propose to create semantic links between the terms contained in inverted index, and in such way create a semantic network. This network will preserve the internal structure of the stored documents, and will enable the users to perform structural queries. Both structural-saving indexation and structural user search query allow to save semantic speech meaning of the text while search process.**

I. Introduction

Today all popular search engines operate with inverted index [1],[2],[3], which provides the grounds for high-quality keyword-based search. The main idea is to create a mapping from every token to a list of its positions in the documents, indexed by the search engine. While both page ranking and linguistic algorithms can offer rather acceptable results for the users, the very idea of processing non-linked keywords, extracted from texts, imply non-semantic search only.

Thus, today's semantic networks, implemented both by commercial companies and and open communities are not fully utilized by the search engines. Powerful linguistic and statistic functions, implemented in modern search engines, are not used to their full extent.

A. Preserving semantic links

The main idea of the present work is to store not inverted index, but sentence structure with link to it's source page position. The base sentence structure consists of three elements: predicate, subject and object, called a triplet. This idea is presented in the Figure 1. Each page is parsed to get linked tokens, constituting the elements to be saved to the database with sentence links and source page positions. The tokens form an oriented graph or a semantic network. Subjects in such graph serve as objects for other subjects and vice versa. This structure is similar to RDF [5] (Resource Description Framework), which describes knowledge using a directed graph.

Search process is implemented with RDF queries over the semantic network. The user enters a triplet in form of three words, which is searched in database of linked documents (a semantic network). In the future, the user will be able to use natural language as a query language. In this case, the system will be able to process not only triplet words, but also other syntactic forms. This means that the indexer will have to process the source documents using an extended RDF scheme, which would contain also adjectives, adverbs, and other parts of speech (POS).

A query triplet can be searched in the database using a simple straightforward comparison or with the methods used in many popular search engines, such as synonyms dictionary and TF-IDF[1][4].

TF-IDF can be implemented as a coefficient of relevance, which influences the document position in the resulting list.

Our search algorithm is not intended to replace traditional inverted index search engines, and can be implemented within an additional module, or serve as a basis for a specialized fact search engine in a knowledge graph.

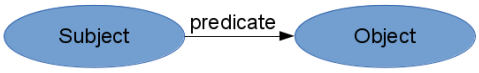

Fig 1: RDF

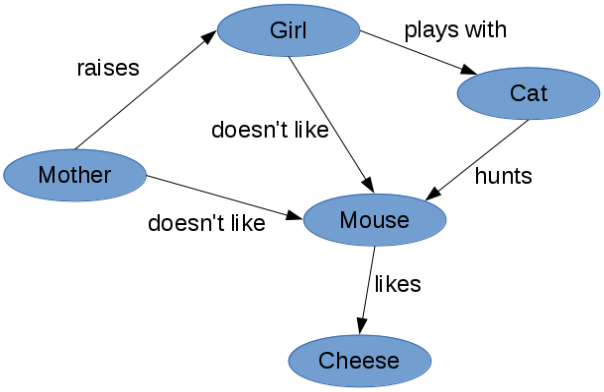

Fig 2: Semantic network

[1]TF-IDF is numerical statistic dimension defining document relevancy in document collection. Result depends on word frequency in current document and inverse frequency in other documents.

B. Basic search engine functions

Our search engine indexing mechanism (robot or spider) solves the following problems:

1. Detecting document external links
2. Useful content detection
3. Semantic structure parsing

C. Useful content detection

For useful content detection we used an artificial neural network, as suggested in [6]. Many of web-pages sources are divided into separate strings containing HTML markup and text, despite it doesn't influence to page rendering. Also we empirically devided HTML tags into two groups: simple and special. Simple tags collection contains text decorating tags like "<b><i><s>". Special tags set contains the others one. Our neural network detects whether a given string contains meaningful text or non-meaningful webpage elements.

While exploring HTML documents we found such regularities:

1. Useful content usually is absent at the beginning and ending of the article.
2. A string is probably useful if the presence of HTML tags inside the string is low.
3. Longer strings are most probably useful.

We used neural network with thr following input parameters:

1. Document string number expressed in percents.
2. A string length expressed in percents. 100% is the longest document string.
3. Relation between simple HTML tags and text chars.
4. Relation between special HTML tags and text chars.

Every parameter except the first one is repeated two more times: for the previous string, and for the next one. The characteristics of our neural networks are shown in Table I.

We trained the neural network using 50 English Wikipedia pages. This method allowed us to quickly get a content parser, having 83% decision accuracy. As a neural network engine we used Encog Java library.

D. Database

Our system uses two DBMS: a NoSQL graph-oriented Neo4J DBMS, and a NoSQL document-based MongoDB. In order to store tokens with minimal overhead, we employ tries (see Figure 3).

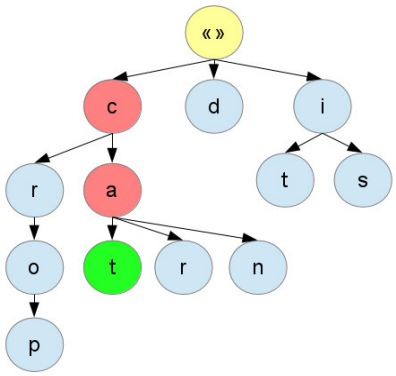

Fig 3: Trie

Tries are supported by Neo4J DBMS that stores all data as a graph, and provides handy ways to traverse graphs, and search and retrieve individual vertices.

To store RDF-like links we used a NoSQL document-based DBMS MongoDB to achieve structureless storage organization, and high speed. In our case, the web spider saves parsed sentences into the documents, containing document index of a predicate, an object, subject, and a link to a word trailing letter in a trie. This structureless organization allows us to add new part-of-speech elements without restructuring the database. Also this allows us to model any sentence structure with optional adjectives or participles.

II. Structure parsing algorithm

A. Common work algorithm

To get a parsed sentence, the system performs the following steps:

1. Anaphora resolution
2. Sentence segmentation
3. Token boundaries identification.
4. Part-of-speech tagging of the tokens array.
5. Syntactic parsing of the POS-tagged sequences.

During components selection we tried to use the subsystems, containing English language and preferentially Russian language model.

Table I.
Neural network characteristics

# Layer	Neurons count	Function
1 Output	1	TanH
2	4	TanH
3	7	TanH
4	11	TanH
5	12	TanH
6 Input	9	Linear

B. Anaphora resolution

Anaphora (coreference) resolution systems are less developed, but there are some systems available:

1. OpenNLP
2. CherryPicker
3. JavaRAP (pronoun coreference system)
4. BART
5. ARKref (rule-based)
6. ARS

According to the recommendations provided in [7], we have chosen ARKref as a main anaphora resolution module. ARKref is a deterministic, rule-based system that uses rich syntactic and semantic information to make antecedent selection decisions.

C. Identifying sentence and token borders; POS tagging

This spider system is implemented as a separate unit with a separate API. For sentence and token borders identification, there are many ready solutions available, and this topic is widely covered in the literature. For now, our system works mainly with the English language, but as we might want to extend the list of supported languages in the future, we so we selected an extensible open source Java TreeTagger system. TreeTagger is fast, and has low RAM and CPU consumption with availability of various language models. TreeTagger can be quickly replaced with any other tokenizer.

D. Dependency parsing

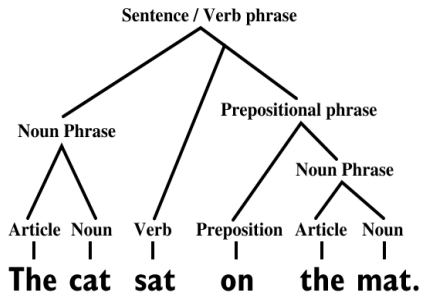

Fig 4: Constituency parsing

Firstly we tried to use Standford NLP Parser as a main sentence processing instrument, but we faced high RAM and CPU consumption. Furthermore, Stanford Parser uses constituency[2] grammars that do not reflect well the structure of languages with relaxed word order, such as Russian. For such languages dependency grammars are usually considered more appropriate.

Following the recommendations in [8], we have chosen MaltParser for best parsing quality from the list of available parsers.

To train the parsing system to recognize any particular language, a deeply annotated text corpus (a treebank) is needed. For each word in a Treebank, the following data is required:

1. Word position in the sentence
2. Word
3. Grammatical attributes
4. Head word position
5. Dependency type

MaltParser contains pre-trained models for English, French, and Swedish. For other languages, it is necessary to create a malttab training set. For the Russian language, the treebank is available as a part of "National Russian language corpus"

III. Implementation

A. The platform

Our search engine consists of two main parts: the search indexer (spider) and the web interface. As most of the NLP software is written in Java, the spider is also written in Java. Since some of the NLP systems operate with space-consuming language models, some heavy weight modules were separated from the base system and made available via RPC API. Thanks to this approach, the system has an ability to use several servers that process different languages (i.e., it is horizontally scalable). Such RPC-available modules are: the anaphora resolution system, the POS tagger, and the dependency parser. For easier development, we have chosen Apache Thrift RPC framework for every isolated component.

As mentioned above, the application stores data in two databases: graph-based Neo4J and document – based MongoDB. The web interface is written in JavaScript/JQuery and operates using Java Spring-based REST API. The component diagram is shown in Figure 5.

All components are implemented in similar ways, and each of them uses a multi-threaded RPC framework, and thus performs multi-threaded text processing.

B. Indexer component

The Indexer component's ("Spider" in the components diagram) aim is to get the next page from the list of links, to process it by calling other components' RPC API and to save the results into the database. Furthermore, this component extracts the links to the new documents to be analyzed, and adds them to the general links list.

This component works with other modules via RPC framework Apache Thrift, that is used due to the simplicity of cross-platform code generation, its lightweight protocol (as opposed to XML-RPC or SOAP), simple implementation and multithreading. At the present time, the system does not support language detection, but the system can operate via RPC with several other processing servers, handling different natural languages. To test the system, we used English Wikipedia as the data source.

C. Anaphora resolution

The anaphora resolution module operates with raw text (cleaned from HTML markup), and replaces pronoun or noun anaphors with their antecedents. As a result, the spi-

[2]Constituency grammar is based on Chomsky's generative grammar. Parsers based on constituency grammars try to divide sentences into smaller word groups until the individual tokens are identified. The example of phrase-structure (constituency) parsing is shown in Figure 4.

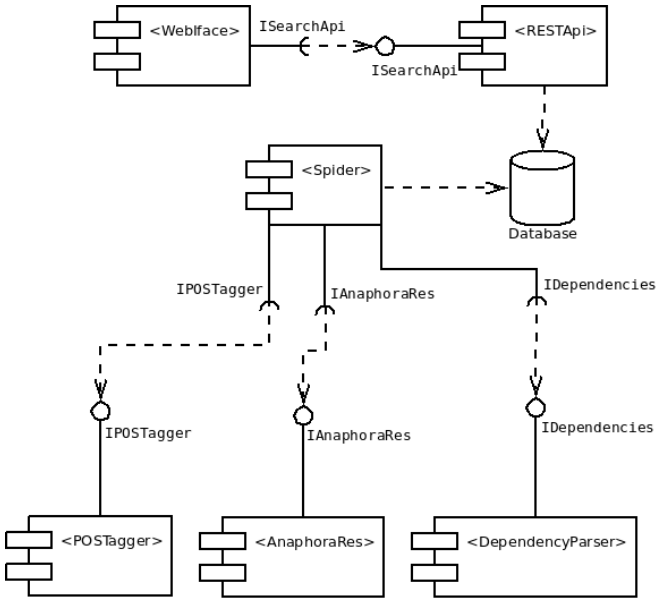

Fig 5: Components diagram

der gets two text versions: the raw text and the text with resolved coreferences. The latter document is being processed in other modules, but both are saved to the database. The anaphora resolution module is a multithreaded server. The number of threads is set up in the server configuration.

D. POS tagging and Dependency finder

These components are marked as "POSTagger" and "DependencyParser" in the components diagram.

Component class diagram for these modules is similar to the coreference resolution component it uses RPC-server classes, singleton configuration classes and other.

MaltParser makes output data in the CoNLL format, similar to the malttab format.

E. Web interface

The web interface is a web application, written in JavaScipt/JQuery. The current web interface allows the user to input three words: subject, predicate and object, to be sent to the server via the REST API. The system processes the query and finds the list of suitable sentences in the database.

REST API is implemented with Java Spring framework.

Using separate processing modules leads to ability of search query NLP processing. This would allow users to make queries as usual sentences.

IV. Related work

[9] also proposed similar semantic network storing approach. Author offers to store RDF structures like a graph using object-oriented databases.

[10] describes a system that processes automatic text sentences tagging for further text managing analyzing or searching.

Our system novelty essence is the approach to process, store and search text data. The method novelty lies in transformation text into RDF-like semantic network and following triplet search over the prepared semantic network index.

V. Conclusion

Our research aim was to try to create a semantic-powered search engine that uses NLP technologies. During the development we have analyzed different information retrieval and NLP instruments and methods, such as syntactic parsing, POS tagging, and coreference analysis.

As the result, we got a semantic sentence-structure search engine prototype. Currently, the system has the following limitations:

1. System processes English-language documents only.
2. The useful content extraction module reliably parses Wikipedia documents only.
3. The current system operates only with triplets. It cannot process adjectives or adverbs.
4. The system does not use any synonyms dictionary.

Finally, system searches triplet like three English words contacted with "AND" boolean operator and it is unusable in current state as providing search service is very poor concerning to internal Wikipedia search (cause Wikipedia is used as testing data source). Search engine returns result with given wittingly right query i.e. known triplet from known document.

Also sentence parsing system is very poor at the moment. We parsed part of Wikipedia for system test. The result is presented in Table 2.

Using synonyms dictionary, a more diverse knowledge-base as a data source, and coreference resolution improvements should make results better.

Table II.
Experiment results

Comment	Value
Total indexed documents	1401
Total parsed sentences	17226
Average parsed sentences in document	12
Average sentences in document	26
Result	54% of document information is lost

References

[1] S. Ilyinsky, M. Kuzmin, A. Melkov and I. Segalovich, “An efficient method to detect duplicates of Web documents with the use of inverted index”, WWW Conference, 2002

[2] S. Brin and L. Page, “The anatomy of a large-scale hypertextual Web search engine”, Proceedings of the seventh international conference on World Wide Web 7, 1998, pp. 107-117

[3] C. D. Manning, P. Raghavan and H. Schütze, “Introduction to Information Retrieval” Cambridge University Press. 2008 p. 6.

[4] A. Gulin, P. Karpovich, D. Raskovalov and I. Segalovich, “Ranking algorithms optimisation using machine-learning methods”, Romip proceedings, 2009

[5] A. Harth and S. Decker, “Optimized Index Structures for Querying RDF from the Web”, Digital Enterprise Research Institute (DERI), National University of Galway, Ireland, 2005, p. 2.

[6] S. Edunov, “How to extract useful content from HTML”, “http://www.algorithmist.ru/2010/11/html-2.html”

[7] Benjamin Chu Min Xian, F. Zahari and D. Lukose, “Benchmarking ARS: Anaphora Resolution System”, Proceedings of the 11th International Conference on Knowledge Management and Knowledge Technologies, 2011, p. 39

[8] A. Gareyshina, M. Ionov, O. Lyashevskaya, D. Privoznov, E. Sokolova and S. Toldova, “RU-EVAL-2012: Evaluating dependency parsers for Russian” Proceedings of COLING 2012: Posters, IIT Bombay, Mumbai, India, pp. 349-360

[9] V. Bonstrom, A. Hinze, H. Schweppe, "Storing RDF as a graph", Web Congress, 2003. Proceedings. First Latin American , vol., no., pp.27,36, 10-12 Nov. 2003

[10] M. Kalender, Jiangbo Dang, "SKMT: A Semantic Knowledge Management Tool for Content Tagging, Search and Management," Semantics, Knowledge and Grids (SKG), 2012 Eighth International Conference on , vol., no., pp.112,119, 22-24 Oct. 2012