QAST: Question Answering System for Thai Wikipedia

Wittawat Jitkrittum† **Choochart Haruechaiyasak**‡ **Thanaruk Theeramunkong**†

†School of Information, Computer and Communication Technology (ICT)
Sirindhorn International Institute of Technology (SIIT)
131 Moo 5 Tiwanont Rd., Bangkadi, Muang, Phathumthani, Thailand, 12000
wittawatj@gmail.com, thanaruk@siit.tu.ac.th

‡Human Language Technology Laboratory (HLT)
National Electronics and Computer Technology Center (NECTEC)
Thailand Science Park, Klong Luang, Pathumthani 12120, Thailand
choochart.haruechaiyasak@nectec.or.th

Abstract

We propose an open-domain question answering system using Thai Wikipedia as the knowledge base. Two types of information are used for answering a question: (1) structured information extracted and stored in the form of Resource Description Framework (RDF), and (2) unstructured texts stored as a search index. For the structured information, SPARQL transformed query is applied to retrieve a short answer from the RDF base. For the unstructured information, keyword-based query is used to retrieve the shortest text span containing the questions's key terms. From the experimental results, the system which integrates both approaches could achieve an average MRR of 0.47 based on 215 test questions.

1 Introduction

Most keyword-based search engines available online do not support the retrieval of precise information. They only return a list of URLs, each referring to a web page, sorted by relevancy to the user's query. Users then have to manually scan those documents for needed information. Due to this limitation, many techniques for implementing QA systems have been proposed in the past decades.

From the literature reviews, previous and existing QA systems can be broadly categorized into two types:

1. **Knowledge Intensive:** Knowledge intensive systems focus on analyzing and understanding the input questions. The system knows exactly what to be answered, and also what type the answer should be. The analysis phase usually depends on an ontology or a semantic lexicon like WordNet. The answer is retrieved from a predefined organized knowledge base. Natural Language Processing (NLP) techniques are heavily used in a knowledge intensive system.
2. **Data Intensive:** Data intensive systems, which do not fully analyze the input questions, rely on the redundancy of huge amount of data (Dumais et al., 2002). The idea is that if we have a huge amount of data, a piece of information is likely to be stated more than once in different forms. As a result, the data-intensive QA systems are not required to perform many complex NLP techniques.

In this paper, we propose an open-domain QA system for Thai Wikipedia called QAST. The system supports five types of close-ended questions: *person*, *organization*, *place*, *quantity*, and *date/-time*. Our system can be classified as a data intensive type with an additional support of structured information. Structured information in Thai Wikipedia is extracted and represented in the form of RDF. We use SPARQL to retrieve specific information from the RDF base. If using SPARQL cannot answer a given question, the system will retrieve answer candidates from the pre-constructed search index using a technique based on Minimal Span Weighting (Monz, 2003).

2 System Architecture

Figure 1 shows the system architecture of QAST which consists of three main sub-systems: *Data Representation*, *Question Processor*, and *Answer*

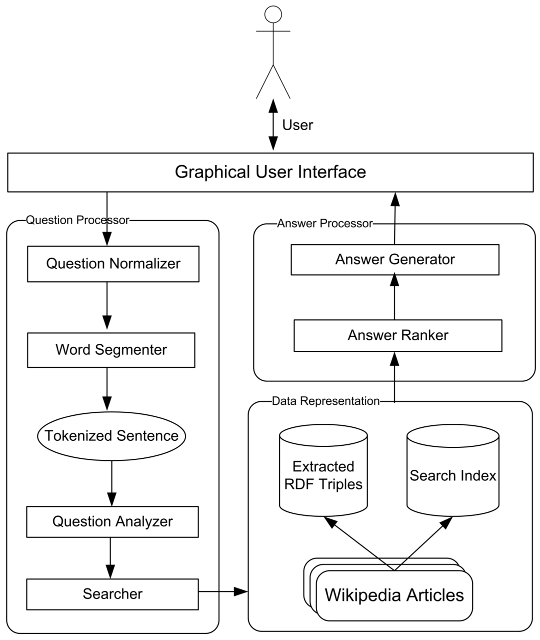

Figure 1: The system architecture of QAST

Processor.

2.1 Data Representation

The *Data Representation* part is a storage for all information contained in Thai Wikipedia. Two modules constitute this sub-system.

RDF Base: In QAST, RDF triples are generated from Wikipedia's infoboxes following similar approaches described in the works of Isbell and Butler (2007) and Auer and Lehmann (2007). To generate RDF triples from an infobox, we would have the article title as the subject. The predicates are the keys in the first column. The objects are the values in the second column. Altogether, the number of generated triples corresponds to the number of rows in the infobox.

In addition to the infoboxes, we also store synonyms in the form of RDF triples. The synonyms are extracted from redirect pages in Wikipedia. For example, a request for the Wikipedia article titled "Car" will result in another article titled "Automobile" to be shown up. The former page usually has no content and only acts as a pointer to another page which contains the full content. The relationship of these two pages implies that "Car" and "Automobile" are synonymous. Synonyms are useful in retrieving the same piece of information with different texual expressions.

Search Index: QAST stores the textual content as a search index. We used the well-known IR library, Lucene[1], for our search backend. We indexed 41,512 articles (as of February 5, 2009) from a Thai Wikipedia dump with full term positions. Firstly, all template constructs and the WikiText markups are removed, leaving with only the plain texts. A dictionary-based longest-matching word segmentation is then performed to tokenize the plain texts into series of terms. Finally, the resulted list of non-stopwords are passed to the Lucene indexing engine. The dictionary used for word segmentation is a combination of word list from the LEX*i*TRON[2] and all article titles from Thai Wikipedia. In total, there are 81,345 words in the dictionary.

2.2 Question Processor

Question processor sub-system consists of four modules as follows.

1. **Question Normalizer** – This first module is to change the way the question is formed into a normal form to ease the processing at latter stages. This includes correcting mistyped words or unusual spelling such as *f33t* for *feet*.
2. **Word Segmenter** – This module performs tokenizing on the normalized question to obtain a list of non-stopwords.
3. **Question Analyzer** – The question analyzer determines the expected type of answer (i.e., quantity, person, organization, location, date/time and unknown) and constructs an appropriate query. Normally, a SPARQL query is generated and used to retrieve a candidate answer from the RDF base. When the SPARQL fails to find an answer, the system will switch to the index search. In that case, the module also defines a set of hint terms to help in locating candidate answers.
4. **Searcher** – This module executes the query and retrieves candidate answers from the data representation part.

To generate a SPARQL query, the input question is compared against a set of predefined regular expression patterns. Currently, the system has two types of patterns: pattern for definitional questions, and pattern for questions asking for a prop-

[1]Apache Lucene, `http://lucene.apache.org`
[2]LEX*i*TRON, `http://lexitron.nectec.or.th`

erty of an entity. The pattern for definitional question is of the form a-rai-kue-X 'What is X ?' or X-kue-a-rai 'X is what ?'. After X is determined from a user's question, the first paragraph of the article titled X is retrieved and directly returned to the user. Since the first paragraph in any article is usually the summary, it is appropriate to use the first paragraph to answer a definitional question.

Questions asking for a property of an entity are of the form a-rai-kue-P-kong-X 'What is P of X ?' e.g., "When was SIIT established ?" which can be answered by looking for the right information in the RDF base. A simplified SPARQL query used to retrieve an answer for this type of question is as follows.

```
SELECT ?o
WHERE {
  ?tempPage hasInfobox ?tempBox .
  ?tempPage rdfs:label "X" .
  ?tempBox ?P ?o .
}
```

The query matches an object of a RDF triple with the predicate P (e.g., "date of establishment"), provided that the triple is generated from an infobox titled X (e.g.,"SIIT") . The object of the year 1992 is then correctly returned as the answer.

When SPARQL fails, i.e., the question does not match any known pattern or the answer does not exist in the RDF base, the system switches to the index search which performs the following the steps.

1. Word Segmenter tokenizes the question into a list of keywords q.
2. Question analyzer analyzes q, generates a basic Lucene' *TermQuery*, and defines a set of hint terms H.
3. Retrieve the most relevant c documents using Lucene's default search scoring function[3]. Denote D as the set of retrieved documents.
4. For each document d in D where $d = \{t_1, t_2, \ldots, t_{|d|}\}$ (t is a term),
   (a) Find in d the start term index $mmsStart$ and end term index $mmsEnd$ of the shortest term span containing all terms in q (Monz, 2003).
   (b) $spanLength \leftarrow 1 + mmsEnd - mmsStart$
   (c) If $spanLength > 30$, skip current d. Go to the next document.
   (d) Find minimal span weighting score msw (Monz, 2003). If $|q \cap d| = 1$ then, $msw = RSV_n(q,d)$. Otherwise, $msw = 0.4 \cdot RSV_n(q,d) + 0.6 \cdot (\frac{|q \cap d|}{spanLength})^{1/8} \cdot (\frac{|q \cap d|}{|q|})$ where $RSV_n(q,d) = lucene(q,d)/max_d lucene(q,d)$
   (e) $mmsStart \leftarrow max(mmsStart - s, 1)$
   (f) $mmsEnd \leftarrow min(mmsEnd + s, |d|)$
   (g) Find the weighting for hint terms hw ($0 \leq hw \leq 1$).
   (h) Calculate the span score $sp = msw \cdot (1 + hw)$
   (i) Add the text span to the span set P (Sort P by sp in descending order).
5. Return the top k spans in P as answers.

In the actual implementation, we set c equal to 500 so that only the top 500 documents are considered. Although retrieving more texts from the corpus would likely increase the chance of finding the answer (Moldovan et al., 2002), our trial-and-error showed that 500 documents seem to be a good trade-off between speed and content coverage. To look for an occurrence of hint terms, each span is stretched backward and forward for 10 terms (i.e., $s = 10$). Finally, we set k equal to 5 to return only the top five spans as the answers.

2.3 Answer Processor

This sub-system contains two modules: *Answer Ranker* and *Answer Generator*.

Answer Ranker concerns with how to rank the retrieved answer candidates. In the case where SPARQL query is used, this module is not required since most of the time there will be only one result returned.

In the case when the search index is used, all candidate answers are sorted by the heuristic span score (i.e., $sp = msw \cdot (1 + hw)$). The function mostly relies on regular expressions defining expected answer patterns. If a span has an occurrence of one of the defined patterns (i.e., $hw > 0$), it is directly proportional to the suitability of the occurrence with respect to the question, length and rareness of the pattern occurrence. For example, the hint terms of questions asking for a person would be personal titles such as Ms. and Dr.

As for the final step, the Answer Generator module formats the top five candidate answers into an HTML table and returns the results to the user.

[3]http://lucene.apache.org/java/2_3_0/scoring.html

Question Type	Index & RDF	Index
Person	0.47	0.37
Organization	0.56	0.46
Place/Location	0.43	0.36
Quantity	0.51	0.44
Date/Time	0.39	0.34
Average MRR	**0.47**	0.39

Table 1: QAST's performance comparison between (1) using both index and RDF and (2) using only the index.

3 Evaluation Metric

To evaluate the system, 215 test questions (43 questions for each question type) and their correct answers were constructed based on the contents of random articles in Thai Wikipedia. Mean Reciprocal Rank (MRR), the official measurement used for QA systems in TREC (Voorhees and Tice, 2000), is used as the performance measurement. To evaluate the system, a question is said to be correctly answered only when at least one of the produced five ranked candidates contained the true answer with the right context. Out-of-context candidate phrases which happen to contain the true answers are not counted. If there is no correct answer in any candidate, the score for that question is equal to zero.

4 Experimental Results and Discussion

Table 1 shows a comparison of the MRR values when using both index and RDF, and using only the index. The approach of using only the index, the overall MRR is equal to 0.39 which is fairly high with respect to the answer retrieval methodology. The index search approach simply relies on the fact that if the question keywords in a ranked candidate document occur close together and at least one occurrence of expected answer pattern exists, then there is a high chance that the term span contains an answer.

The MRR significantly increases to 0.47 (20.5% improvement) when RDF (structured information) is used together with the index. A thorough analysis showed that out of 215 questions, 21 questions triggered the RDF base. Among these, 18 questions were correctly answered. Therefore, using the additional structured information helps answer the definitional and factoid questions. We expect a higher improvement when more structured information is incorporated into the system.

5 Conclusions and Future Works

We proposed an open-domain QA system called QAST. The system uses Thai Wikipedia as the corpus and does not rely on any complex NLP technique in retrieving an answer.

As for future works, some possiblities for improving the current QAST are as follows.

- An information extraction module may be added to extract and generate RDF triples from unstructured text.
- Infoboxes, wikipedia categories and internal article links may be further explored to construct an ontology which will allow an automatic type inference of entities.
- More question patterns and the corresponding SPARQL queries can be added so that SPARQL is used more often.

Acknowledgement

The financial support from Young Scientist and Technologist Programme, NSTDA (YSTP : SP-51-NT-15) is gratefully acknowledged.

References

Soren Auer and Jens Lehmann. 2007. *What Have Innsbruck and Leipzig in Common? Extracting Semantics from Wiki Content.* In Proc. of the 4^{th} European conference on The Semantic Web: Research and Applications, pp. 503-517.

Susan Dumais, Michele Banko, Eric Brill, Jimmy Lin, and Andrew Ng. 2002. *Web Question Answering: Is More Always Better?.* In Proc. of the 25th ACM SIGIR, pp. 291-298.

Jonathan Isbell and Mark H. Butler. 2007. *Extracting and Re-using Structured Data from Wikis.* Technical Report HPL-2007-182, Hewlett-Packard.

Dan Moldovan, Marius Pasca, Sanda Harabagiu, and Mihai Surdeanu . 2002. *Performance Issues and Error Analysis in an Open-Domain Question Answering System.* Proc. of the 40^{th} ACL, pp. 33-40.

Christof Monz. 2003. *From Document Retrieval to Question Answering.* Ph.D. Thesis. University of Amsterdam.

Ellen M. Voorhees and Dawn Tice. 2000. *Building a Question Answering Test Collection.* In 23^{rd} ACM SIGIR, pp. 200-207.