

DalTREC 2007 QA System Jellyfish: Experiments with Integration of Lucene and GATE, and Improved Usage of WordNet and Qrel

Tony Abou-Assaleh,^{*} Chris Whidden, Vlado Kešelj,
Hathai Tanta-ngai, and Nick Cercone[†]
Faculty of Computer Science
Dalhousie University, Halifax, Canada
{vlado,taa,whidden,hathai,nick}@cs.dal.ca

16 October 2007

Abstract

We present a question-answering system Jellyfish. Our approach is based on marking and matching steps that are implemented using the methodology of cascaded regular-expression rewriting. In addition to bug fixes and minor tweaks, Jellyfish 2007 includes Apache Lucene as a passage retrieval system, coreference resolution using GATE, and question relation marking assisted by WordNet. We present the system architecture and evaluate the system using the TREC 2006 and 2007 datasets.

1 Introduction

Previous work has explored the application of several approaches to question answering in the overlapping area of unification-based and stochastic NLP (Natural Language Processing) techniques [1, 6]. Two novel methods that were explored relied on the notions of *modularity* and *just-in-time sub-grammar extraction*.

One of the learned lessons of the previous experiments is that the regular expression (RegExp) substitutions are a very succinct, efficient, maintainable, and scalable method to model many NL subtasks of the QA task. This is also observed in the context of lexical source-code transformations of arbitrary programming languages [2], where RegExp substitutions are an alternative to

^{*}Abou-Assaleh is also the Director of Research and Development at GenieKnows.com, Halifax, Nova Scotia, Canada.

[†]Cercone is also the Dean of the Faculty of Science and Engineering, York University, Toronto, Ontario, Canada.

manipulating the abstract syntax tree, and proved to be more robust in the face of missing header files, errors, usage of macros, templates, and other embedded programming language constructs.

We employ RegExp rewriting as a primary technique in our question-answering (QA) system Jellyfish. We use RegExp matching and rewriting at various stages of the system, whose architecture is described in section 3.

We evaluate our system using the datasets from TREC 2006 and TREC 2007. Our system was originally developed with respect to the TREC 2004 dataset and has been updated based on the analysis of its performance on the TREC 2006 dataset. The TREC 2007 dataset is used for testing.

Our goal is to apply a unification-based approach as a high-level answer-extraction step on top of the low-level RegExp processing.

2 Regular Expression Rewriting

The basic method used at various components of the QA system is *RegExp rewriting*. The open angle bracket (\langle) is used as a special escape character, hence we make sure that it does not appear in the source text, which is either a question or a passage. The basic text substrings, such as the target or named entities, are recognized using regular expressions and replaced with an angle-bracket-delimited expression. For example, the target is marked as $\langle \text{TARGET} \rangle$. More commonly, a named entity e of type t is replaced with $\langle t_{e_s} \rangle$, where e_s is the named entity e encoded as a string of printable characters that do not include \langle . The RegExp rewriting can be seen as a bottom-up deterministic parsing technique. For example, the rewriting in which “ $\langle \text{NP}_x \rangle \langle \text{VP}_y \rangle$ ” is replaced with “ $\langle \text{S}_z \rangle$ ” corresponds to the context-free rule $S \rightarrow NP VP$. The value z is obtained by decoding x and y , concatenating them, and encoding the result again.

3 System Architecture

The current system consists of the following phases: 1) Question Processing, 2) Passage Retrieval, 3) Target Marking, 4) Question Category Marking, 5) Question Relation Marking, 6) Matching, 7) Answer selection.

3.1 Question Processing

The Question Processing phase takes the original questions as input, parses them, and generates complete questions as output. Questions are parsed using RegExp matching and substitution to identify the question category and extract some related metadata. Some metadata extracted during parsing is analyzed using WordNet [4] to identify additional metadata and relationships between the target and what the question is asking about.

In TREC datasets, questions are grouped by targets. Replacing the anaphoric references in questions with the target results in self-contained questions. These *complete* questions are used in passage retrieval.

3.2 Passage Retrieval

The passage retrieval from the AQUAINT dataset is performed by an external search engine using the full questions generated in the question processing phase. We use two different sets of passages. The first set is the passages provided by NIST using the PRISE search engine. The second set is retrieved using the Apache Lucene [7] open-source search engine. In both cases, the PRISE and Lucene, the results are treated in the same way—as passages relevant to the question.

For the Blog dataset we used the passages provided by NIST, which were a result of the PRISE search engine. We have not obtained the full Blog dataset.

3.3 Target Marking

The string in the question that constitutes that target is identified during the question processing step (section 3.1). Using simple RegExp rewriting rules, the target is identified in the passages and replaced with the <TARGET> tag. In effect, this phase annotates sentences in the passages that may contain an answer.

Target marking in the previous versions of Jellyfish only used an exact match for the target. This meant that any references to the target by last name, type, or via pronouns were ignored by Jellyfish. In Jellyfish 2007, we added coreference resolution by incorporating GATE [3]. Using GATE’s coreference resolution module, we improved target marking when the target type is person, since GATE is good at resolving names and pronouns. However, GATE was less suitable for handling location and event coreference resolutions.

3.4 Question Category Marking

During this step, the system scans all the relevant passages and uses RegExp rewriting to mark entities that belong to the question category. The type of RegExp used depends on the question category and may be a simple keyword-based RegExp or a sophisticated multipart RegExp. Question category marking acts as a just-in-time annotation phase where only the passages that may be relevant to the current question are annotated, and the annotation data is customized based on the question category.

3.5 Question Relation Marking

One of the new features in Jellyfish 2007 is question relation marking. A question relation is a relation involving one or more entities in the question, usually the target. For example, the question “How did Beethoven die?” has the relation

die involving Beethoven. The question relation marking component of Jellyfish uses WordNet [4] to identify synonyms of question relations. First the question relation is reduced to a base form by WordNet. Then each word of the passages is examined to see if it has the same base form, is a synonym, or has a subtype of the same base form as the question relation. If so, it is marked as a question relation. For the example question above, occurrences of *die*, *died*, and *killed* would all be marked as question relations.

3.6 Answer Matching

During answer matching, question metadata and annotated passages are combined using special RegEx rules to generate candidate matches. The rules are applied sequentially. Every time a match is found, it is added to the list of matches. The relative order of the rules imposes an implicit ranking of matches. Consequently, more specific rules are placed before the more general ones. Since some questions may have no answers in the dataset, one must avoid using rules that are too general. Appropriate rule generality and ordering depends on the question category. Typically, more specific question categories permit the usage of more general rules, and vice versa.

3.7 Answer Selection

This phase is a filtering step that takes the list of answer for each question from the matching phase and selects the answers that are to be included in the output. For TREC, we set the number of answers for factoid questions to 1 and for list questions to 7. Presently, we simply select the first answers in the list.

The selected answers are formatted to be used either for evaluation, inspection, or integration in other systems.

4 Evaluation

Run	Correct	Factual	List	Other	Other per series
Dal06l	43	0.107	0.061	-	-
Dal07n	34 (4/2/1)	0.094	0.029	0.033	0.051
Dal07p	28 (2/6/0)	0.078	0.028	0.034	0.046
Dal07t	38 (3/6/2)	0.106	0.034	0.048	0.063
Best 07	-	0.706	0.479	0.329	0.484
Median 07	-	0.131	0.085	0.118	0.108
Worst 07	-	0.019	0.000	0.000	0.015

Table 1: DalTREC 2007 Results. The numbers in parentheses refer to the unsupported, inexact, and locally correct, respectively.

The TREC 2004 QA dataset was used for deriving the RegEx rules; further improvements and fine tuning of the system followed the TREC 2007 QA

dataset. Testing of the system used TREC 2007 QA datasets. On the training data (TREC 2006 dataset), our system was able to correctly answer 43 out of 403 factoid questions in the TREC 2006 dataset yielding an accuracy of 10.7%. In the TREC 2007 dataset, the number of factoid question was 360. The formulation of the questions in TREC 2007 relied more heavily on expanding the questions to incorporate the context information provided in the target of each series. We have not used any external resources, other than WordNet. Beside the AQUAINT corpus, we use the top 50 Blog documents per target, provide by NIST based on the PRISE search engine results. Our system answered 38 questions globally correctly, and 2 locally correctly, yielding factoid correctness accuracy of 10.6%. The results of evaluation are presented in table 1. The following runs are presented:

- **Dal06l:** 100 top Lucene passages; GATE for coreference resolution; WordNet for question relation marking; TREC 2006 questions
- **Dal07n:** 100 top PRISE passages; no blog data; GATE for coreference resolution; no question relation marking; TREC 2007 questions
- **Dal07p:** 100 top PRISE passages; no blog data; GATE for coreference resolution on first passage only; WordNet used for question relation marking; TREC 2007 questions
- **Dal07t:** 100 top Lucene passages; 50 top PRISE blog passages; no coreference resolution; no question relation marking; TREC 2007 questions

5 Lessons Learned

As a general lesson, we are continuing on building our system using the RegExp rewriting technique, and our experience is that RegExp rewriting is a simple and powerful parsing technique. We have effectively used coarse-grained modularization of RegExp, and combined it with dynamic loading of RegExp. Fine grained modularization of RegExp is possible, and would simplify the task of creating RegExp rules. Just-in-time RegExp-based annotation can lower the computation requirements of deeper analysis. Our system is fairly robust. The performance of Jellyfish on TREC 2006 and TREC 2007 questions is comparable. The technique itself is not used to its full potential due to limited number of man-hours that we were able to devote to the project.

Regarding the particular evaluation results returned by evaluators, the run Dal07t performed significantly better. More experiments need to be performed with the evaluation script, when it becomes ready, since there are several factors that may have caused this difference. Those are the use of Lucene, use of Blog passages, or not using coreference resolution or question relation marking. A positive influence of not using coreference resolution or question relation marking is unlikely since our earlier experiments indicated a positive effect of these techniques. Hence, it needs to be determined whether Lucene is a search engine that boosted our performance or it was the use of Blog data.

References

- [1] N. Cercone, L. Hou, V. Kešelj, A. An, K. Naruedomkul, and X. Hu. From computational intelligence to web intelligence. *IEEE Computer*, 35(11):72–76, November 2002.
- [2] A. Cox, T. Abou-Assaleh, W. Ai, and V. Kešelj. Lexical source-code transformation. In *Proceedings of the STS'04 Workshop at GPCE/OOPSLA*, Vancouver, Canada, October 2004.
- [3] H. Cunningham, D. Maynard, K. Bontcheva, and V. Tablan. Gate: A framework and graphical development environment for robust nlp tools and applications. In *Proceedings of the 40th Anniversary Meeting of the Association for Computational Linguistics (ACL'02)*, Philadelphia, PA, July 2002.
- [4] C. Fellbaum, editor. *WordNet: An Electronic Lexical Database*. MIT Press, 1998.
- [5] V. Kešelj. Question answering using unification-based grammar. In E. Stroulia and S. Matwin, editors, *Advances in Artificial Intelligence, AI 2001*, volume LNAI 2056 of *L.N. in Comp.Sci.*, Springer, pages 297–306, Ottawa, 2001.
- [6] V. Kešelj. Modular stochastic HPSGs for question answering. Technical Report CS-2002-28, School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, June 2002.
- [7] The Apache Software Foundation. Apache Lucene. <http://lucene.apache.org/java/docs/index.html>, October 2007.