DalTREC 2005 QA System Jellyfish: Mark-and-Match Approach to Question Answering

Tony Abou-Assaleh Nick Cercone Jon Doyle Vlado Kešelj
Chris Whidden
Faculty of Computer Science
Dalhousie University, Halifax, Canada
{taa,nick,doyle,vlado,whidden}@cs.dal.ca

February 5, 2006

Abstract

This is the second year that Dalhousie University actively participated in TREC. Three runs were submitted for the Question Answering track. Our results are below the median; however, they're signifantly larger than minimal, the lesson learnt will guide our future development of the system. Our approach was based on a mark-and-match methodology with regular expression rewriting.

1 Introduction

2005 is the second year that Dalhousie University participated actively in TREC. The project DalTREC[1] was initiated in 2003 and serves as an umbrella for different TREC-related activities at Dalhousie. The tracks of interest were Question Answering, Spam, HARD, Genomics, and Enterprise. We submitted runs for the Question Answering (QA) track and the Spam track. This paper discusses the QA track.

In our previous work, we explored the application of several approaches to question answering in the overlapping area of unification-based and stochastic NLP (Natural Language Processing) techniques [5, 1]. 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 was 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 was also observed in the context of lexical source-code transformations of arbitrary programming languages [2], where it is an alternative to manipulations of the abstract syntax tree. In this context, RegExp transformations are more robust in the face of missing header files, errors, usage of macros, templates, and other embedded programming language constructs. This is the bottom-up part of our system. The top-down part consists of unification-based parsing and matching and, while being developed, it has not been yet deployed in the system.

[1] http://www.cs.dal.ca/~trecacct — DalTREC URL

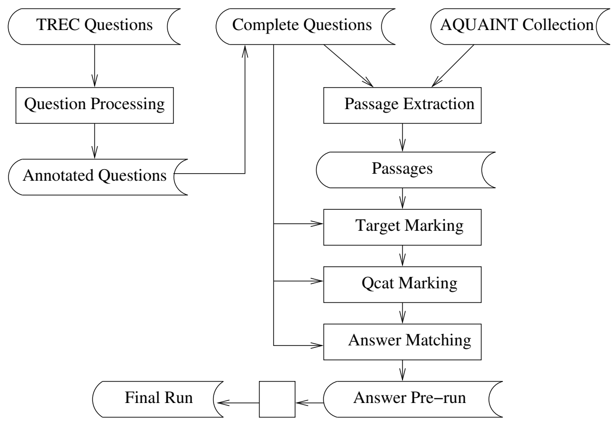

Figure 1: System Overview

2 Regular Expression Rewriting

The basic method used at various components of the QA system is *RegExp rewriting.* The open angle bracket (<) 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 `<TARGET>`. More commonly, a named entity e of type t is replaced with <t_e_s>, where e_s is the named entity e encoded as a string of printable characters that do not include <. The RegExp rewriting can be seen as a bottom-up deterministic parsing technique. For example, the rewriting in which "<NP_x> <VP_y>" is replaced with "<S_z>" corresponds to the context-free rule $S \to NP\ VP$. The value z is obtained by decoding x and y, concatenating them, and encoding the result again.

3 System Architecture

An overview of the system architecture is shown in figure 1 and it is described in more detail in the remaining of this section.

The current system could be described as a sequence of the following phases:

1. Question Processing
2. Passage Retrieval
3. Target Marking
4. Question Category Marking
5. Matching
6. Prerun-to-run
7. Document Ranking

3.1 Question Processing

The Question Processing component takes the original TREC questions as the input and produces an annotated list of questions. The process includes question parsing and generating complete questions.

3.1.1 Question Parsing

Questions are parsed using RegExp substitutions. During the parsing stage, based on RegExp matching, the system identifies the question category. In addition, the system may identify some category and question specific metadata. For instance, if the category is `SIZE`, the system may also identify the measurement units.

The question categories that system recognizes are `ACRONYM`, `ACRONYMEXP`, `ACTIVITY`, `AGE`, `CITY`, `CONTINUATION`, `COUNT`, `COUNTRY`, `DATE`, `DEF`, `DEPTH`, `DISTANCE`, `DURATION`, `HEIGHT`, `HOW`, `HOWDIE`, `HOWGETNAME`, `LENGTH`, `LENGTHORDURATION`, `LOCATION`, `MONEY`, `NAME`, `NATIONALITY`, `NICKNAME`, `OTHER`, `PERCENTAGE`, `PERSON`, `PRETEXT`, `PROVINCE`, `SEASON`, `SIZE`, `SPEED`, `SUB`, `TEMPERATURE`, `TIME`, `TRANSLATE`, `WHY`, and `YEAR`.

Some metadata extracted during parsing is analyzed using WordNet [3] to identify additional metadata and relationships between the target and what the question is asking about.

3.1.2 Generating Complete Questions

The TREC 2005 questions, like the TREC 2004 questions, are grouped by targets. A set of questions around a target may include just an anaphoric reference to the target and not the full target name; e.g., 'it' or 'he'. These are incomplete questions in the sense that they do not provide sufficient information for the answer to be retrieved, but frequently require the target information. We find it useful to generate a complete version of the question for several reasons: (1) we can rely on a continuation of the strategy used on the previous TREC's, and (2) we can approach the QA problem with a general question-processing methodology instead of requiring that our system handles context specified in certain way, even though it may mean simply an addition of a target.

The original TREC 2004 questions are grouped by targets. A set of questions around a target may include just an anaphoric reference to the target and not the full target name (e.g., 'it', 'he'). In order to be able to independently use each question in passage extraction, they are rewritten so that they include the target name. We call this form a "complete question" or "full question". Additionally, as the result of parsing the questions, we obtain question category (i.e., the expected answer type), and some other optional information, such as type of the relation between the target and the answer. Question parsing and generating full questions is based on regular expression rewriting rules.

Generating the full question was done in the following way: We start with the original question. If target appears in the question, it is not changed. For a question of type "other," we generate the question "`What is <TARGET> ?`". Otherwise, we attempt the following substitutions:

```
elsif ($Qtype eq 'OTHER') { $_ = " What is <TARGET> " }
elsif (/ it /)      { s/ it / <TARGET> / }
elsif (/ its /)     { s/ its / <TARGET>'s / }
elsif (/ he /)      { s/ he / <TARGET> / }
elsif (/ his /)     { s/ his / <TARGET>'s / }
elsif (/ she /)     { s/ she / <TARGET> / }
elsif (/ her /)     { s/ her / <TARGET>'s / }
```

```
elsif (/ they /)    { s/ they / <TARGET> / }
elsif (/ their /)   { s/ their / <TARGET>'s / }
elsif (/ theirs /)  { s/ theirs / <TARGET>'s / }
elsif (/^ It /)     { s/ It / <TARGET> / }
elsif (/^ Its /)    { s/ Its / <TARGET>'s / }
elsif (/^ He /)     { s/ He / <TARGET> / }
elsif (/^ His /)    { s/ His / <TARGET>'s / }
elsif (/^ She /)    { s/ She / <TARGET> / }
elsif (/^ Her /)    { s/ Her / <TARGET>'s / }
elsif (/^ They /)   { s/ They / <TARGET> / }
elsif (/^ Their /)  { s/ Their / <TARGET>'s / }
elsif (/^ Theirs /) { s/ Theirs / <TARGET>'s / }
```

If none of them succeeds, we prepend "`<TARGET>, `" to the original question.

We evaluated this approach on TREC-2004 questions, and came up with the following results about the anaphora rewritings:

- 1 difficult question (almost independent of the target)
- 143 correctly rewritten questions
- 2 acceptable but not considered strictly correct, namely

  ```
  How did Heaven's Gate commit suicide ?
  Why did Heaven's Gate commit suicide ?
  ```

- 40 incorrect but acceptable; hard to fix; e.g., "Gordon Gekko, what year was the movie released?" and "Gordon Gekko, who plays the role?"
- 15 incorrect byt acceptable; could be relatively easily fixed
- 76 correctly decided not apply a rewriting,
- 65 "other" questions
- 5 unnecessary target addition due to a
- 4 unneccessary target addition due to singular-plural variations: 3 Crips vs. Crip, and 1 Rhodes scholars vs. Rhodes scolar,.

3.2 Passage Retrieval

The passage retrieval from the AQUAINT data set is performed by an external search engine using the full questions generated in the question processing phase. We used two different sets of passages. The first set was the passages provided by NIST using the PRISE search engine. The second set was generate using our MySQL-based search engine that employed MySQL's full-text indexing functions. In both cases, the PRISE and MySQL, the results are treated in the same way—as passages relevant to the question.

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. Currently, our system does not handle intra-sentence references.

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 multi-RegExp expression. For example, if the question category is `COUNTRY`, then a regular expression that contains a predefined list of country names is fetched, and all RegExp rewriting is applied to matches. These keyword-list RegExps are compiled manually from various sources. Matching the `DATE` category, on the other hand, requires a combination of keyword-list RegExps for month names and abbreviations and RegExps to recognize number, which are combined in a larger RegExp to match various date formats.

We use Perl's autoloading feature to load the appropriate RegExps at runtime. Thus, the concept of sub-grammar extraction is applied to RegExp rules where only the RegExp rules that are useful for the current question category are loaded into memory and applied to the passages.

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 Answer Matching

At this point, the system has two sets of information: question metadata and annotated passages. These two sets of information are combined using special RegExp rules to generate candidate matches. The rules are applied sequentially. Every time a match is found, it is added to the list of matches. Currently, there is no separate ranking step for the extracted matches (although support for this step is built into the system); instead, the matches are ranked implicitly by the order in which the RegExp rules are applied. Hence, more specific RegExp rules are applied before the more general ones.

We were faced with two challenges when creating the RegExp rules. The first was related to determining how general should the most general rule be. Experimenting with questions from the TREC 2004 QA Track, we found that some of the general RegExp rules that we had in place captured erroneous matches for the questions that do not have an answer in the dataset. The second challenge was determining the order in which the RegExp rules should be applied. There were generic and question category specific RegExp rules. We were faced with the difficult tasks of determining the relative placement of generic and specific rules, as well as the placement of rules within a particular question category. For instance, our most specific generic rule is

```
/<_TARGET_>.*?\b$Qrel <${Qcat}([^>]*)>/
```

where `$Qrel` is typically a verb that appears in the question, and `$Qcat` is the question category. However, this rule will only work for the questions for which the metadata `$Qrel` has been identified. When `$Qrel` is not available, we found that the RegExp rule

```
/<TARGET>[^.]*?<${Qcat}_([^>]*)>/
```

is too general for most question categories and should not be used. However, it is the best matching rule if the question category is one of `AGE`, `CITY`, `COUNTRY`, `DATE`, `NATIONALITY`, `PROFESSION`, `PROVINCE`, or `YEAR`.

3.6 Pre-run to Run filtering

The result of matching is a list of answers for each question—several answers may appear for any question, or we may have no answers at all. The task of the pre-run-to-run filtering component is to prepare the final run using the following rules:

- only the first answer is passed for any factoid question (TREC requirement),
- a NIL answer is introduced for questions with no generated answers,
- not more than seven answers are allowed per list question (rule of thumb),
- all answers to a list or 'other' question have to be unique,
- additionally, answers to an 'other' question may not appear as an answer to any previous question about the same target, and
- any answer longer than 100 bytes is truncated.

Although not used in our runs for TREC 2005, our system can handle web-reinforced question answering. In the case where external resources are used to located candidate answers, this phase is used to locate the relevant documents to support the answers in the AQUAINT dataset.

3.7 Document Ranking

The ID of documents that support the answers are identified when the matches are found during the Answer Matching phase (section 3.5). The document ranking phase simply checks that the final output conforms to the output specifications of TREC 2005, and corrects any potential errors. For example, if external resources are used to find an answer, but no supporting document is found in the given dataset, then a default document is used, since it is not allowed to leave the document field blank.

4 Evaluation

We had two version of our system that differed in the way annotations are encoded within the text. In the first version, the tags are encoded into sequences of characters that can be matched in Perl using the RegExp `/\S+/`[2]; while in the second version, the encoded annotations match the RegExp `/\w+/`.[3]

Originally, our RegExp patterns were designed to work with `\w+` encoding. Later, we felt that the `\S+` encoding was more robust than the `\w+` encoding. During development, we evaluated our system using TREC 2004 questions. We found that the two tag encodings gave comparable overall accuracy for factoid questions, but the sets of correctly answered questions sometimes differed.

We used two sets of passages in the submitted runs. One set was retrieved using out MySQL-based search engine, and the other was provided by NIST from the PRISE search engine. Since we were limited

[2] Matches a sequence of non-space characters.
[3] Matches a sequence of alphanumeric characters, including the underscore.

to 3 runs, we chose the following combinations of encodings and search engines based on their performance on the TREC 2004 questions (see table 2):

Dal05m: MySQL-based search engine passage and `\w+` tag encoding.

Dal05p: PRISE search engine passage and `\w+` tag encoding.

Dal05s: PRISE search engine passage and `\S+` tag encoding.

4.1 TREC 2005 Evaluation Summary

Results of the three runs, Dal05m, Dal05p, and Dal05s, as returned from the NIST assessors, are summarized in table 1.

Run	Correct	Factual	Unsupported	Inexact	List	Other	Session
Dal05m	27	0.075	4	10	0.017	0.056	0.056
Dal05p	40	0.110	4	5	0.033	0.088	0.086
Dal05s	41	0.113	4	5	0.033	0.088	0.087
MAX		0.713			0.468	0.248	0.534
MEDIAN		0.152			0.053	0.156	0.123
MIN		0.014			0.000	0.000	0.008

Table 1: Jellyfish results from TREC 2005 QA Track

4.2 Evaluation on TREC 2004 Questions

During the development of Jellyfish, the system was constantly evaluated using the TREC 2004 questions. As a result, the TREC 2004 questions became the training set for our system, while the TREC 2005 question were the testing set. The results of evaluating Jellyfish on TREC 2004 questions are listed in table 2, where the suffix to Dal04 is interpreted as follows: *m* means MySQL-based search engine is used, *p* means the PRISE search engine is used, *w* means that `\w+` tag encoding is used, and *s* mean that `\S+` tag encoding is used.

Run	Correct	Factual	List	Combined
Dal04mw	40	0.174	0.093	0.147
Dal04pw	45	0.196	0.098	0.163
Dal04ms	41	0.178	0.083	0.146
Dal04ps	46	0.200	0.092	0.164

Table 2: Jellyfish evaluation using TREC 2004 Questions

4.3 Analysis

The results from our system in TREC 2005 (table 1) are within our expectations. It is evident from comparing table 1 and table 2 that our MySQL-based search engine has limited coverage. Although there is a noticeable reduction in accuracy on TREC 2005 questions due to the increase of the total number of questions from TREC 2004, the total number of correctly answered questions is comparable to that of the training data.

There is no significant difference in the accuracy when using either of the tag encoding schemes. We believe that using `\S+` tag encoding will facilitate improving the RegExprules, modularizing, and layering them.

4.4 Lessons Learned

1. RegExp rewriting is a simple and powerful parsing technique.
2. We have effectively used coarse-grained modularization of RegExp, and combined it with dynamic loading of RegExp.
3. Fine grained modularization of RegExp is possible, and would simplify the task of creating RegExp rules.
4. Using a macro system, such as Starfish, can greatly reduce the complexity of RegExp rules.
5. Just-in-time RegExp-based annotation can lower the computation requirements of deeper analysis.

5 Conclusions and Future Work

We present a relatively simple QA framework based on regular expression rewriting. We apply the concepts of modular grammar and just-in-time annotation to RegExprewrite rules. Although our system is still in the early stages of development, we have demonstrated that the mark-and-match approach with RegExprewriting is a robust method that can reduce the computation requirements of deeper analysis. We learned several lessons that will direct our further development of Jellyfish.

Future work includes fine-grained modularization of RegExprules, answer ranking, and better usage of syntactic and semantic information during the marking and the matching phases. In particular, we are working on incorporating shallow semantic parsing of the candidate answers in order to rank them.

References

[1] Nick Cercone, Lijun Hou, Vlado Kešelj, Aijun An, Kanlaya Naruedomkul, and Xiaohua Hu. From computational intelligence to web intelligence. *IEEE Computer*, 35(11):72–76, November 2002.

[2] Anthony Cox, Tony Abou-Assaleh, Wei Ai, and Vlado Kešelj. Lexical source-code transformation. In *Proceedings of the STS'04 Workshop at GPCE/OOPSLA*, Vancouver, Canada, October 2004.

[3] C. Fellbaum, editor. *WordNet: An Electronic Lexical Database*. MIT Press, 1998.

[4] Vlado Kešelj. Question answering using unification-based grammar. In Eleni Stroulia and Stan Matwin, editors, *Advances in Artificial Intelligence, AI 2001*, volume LNAI 2056 of *Lecture Notes in Computer Science, Springer*, pages 297–306, Ottawa, Canada, June 2001.

[5] Vlado Kešelj. Modular stochastic HPSGs for question answering. Technical Report CS-2002-28, School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, June 2002.