Pauchok: A Modular Framework for Question Answering

Stefanie Tellex & Boris Katz

Artificial Intelligence Laboratory
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139

http://www.ai.mit.edu

**Motivation:** The Question Answering tracks at recent Text REtrieval Conferences (TREC) provide a forum for evaluating and comparing question answering systems. Many systems entered in TREC have the same general architecture. An implementation of some of the basic abstractions shared across these systems would allow us to more easily create and evaluate new components for question answering systems. In addition, it could act as a test bed that gives feedback on what each component contributes to the overall performance of the system.

**Previous Work:** Hirschman [1] and Light [2] describe generic architectures for question answering systems. In the TREC proceedings [4, 3] Voorhees gives an overview of the systems submitted to TREC. Most systems include modules for some form of document retrieval, passage retrieval, and answer extraction.

**Approach:** Pauchok is a modular framework that captures some of the abstractions common to many question answering systems. It provides an interface between the major components of those abstractions, so that we can implement new modules with minimal effort. Figure 1 shows the interface used for casual interaction with Pauchok.

Pauchok’s design has three major components:

- **Document Retrieval:** The document retrieval module retrieves documents from the corpus likely to contain answers to the user’s question. It consists of a query generation algorithm and an query retrieval interface. The query generation algorithm takes as input the user’s question and creates a query containing terms likely to appear in documents containing an answer. This query is passed to the retriever, which returns a set of documents. The document retrieval module also supports a query back-off mechanism; if a given query does not produce enough documents, the system tries more general queries using a variety of strategies.

- **Passage Retrieval:** Passage retrieval algorithms take a document and a question, and try to return passages from the document most likely to contain an answer. Typical passage retrieval algorithms work by breaking the document into passages and then finding the passage with the highest score. The system abstracts this mechanism so that passage tokenizing algorithms and passage scoring algorithms can be modularized.

- **Answer Extraction:** Finally, the system must extract an answer to the question from the passage. This module takes as input a passage from the passage retrieval component and tries to retrieve the exact phrase representing an answer. This functionality typically requires parsing and detailed question analysis.

A statistics gathering interface is wrapped around all the modules. This interface allows regression tests to be created and and the performance of each individual module to be measured. With this information, we can identify the worst performing modules and apply concentrated effort to improve them.

Using this architecture, we can implement and compare a variety of question answering components. We hope that the application of more linguistically-sophisticated techniques will improve the performance of our system. Relational parsing and indexing, word stemming, pronoun resolution, and synonymy analysis could improve the performance of the passage retrieval and answer extraction modules. For example, existing passage scoring algorithms may benefit by replacing all pronouns in their input text with their referents. We could also create entirely new algorithms based on these techniques, such as a passage retrieval algorithm that looks for certain patterns of syntactic relations in the question and the document.

**Difficulty:** It is hard to find good abstractions to use in realizing a generic question answering system because we have to generalize over many existing systems with a wide variety of algorithms and techniques. Implementing a
Figure 1: Screenshot of the Pauchok System, showing how it can be configured on the fly.

new algorithm often involves architectural changes to the system to refine the set of abstractions used. For example, in the initial implementation of Pauchok, the system generated a single query for both document retrieval and passage retrieval. However, the query used for document retrieval may not be the best one to use for passage retrieval. Through implementing additional passage retrieval algorithms we discovered that it made more sense to have two distinct queries, one for document retrieval and one for passage retrieval.

In theory, giving the system more linguistic information should improve its performance, but it often is not clear how to apply that information. In some cases, misapplying linguistically-sophisticated techniques could actually hurt performance. For example, applying relational parsing to a query generation algorithm might produce an overly specific list of query terms, which would reduce the recall of the system.

**Impact:** Pauchok’s architecture streamlines the implementation of new question answering techniques. In addition, it provides an infrastructure that allows us to easily plug in new algorithms, enabling us to get quick feedback on the performance of new components and to better understand and evaluate existing question answering techniques.

**Future Work:** Although Pauchok’s design is complete, the answer extraction and statistics gathering portions remain to be implemented. Additional indexing methods, query generation and passage retrieval algorithms will be created. The idea of backing off and trying different strategies will be extended to other parts of the system; it is currently only part of document retrieval. We plan to compare various modules and configurations.

**Research Support:** This research is funded by DARPA under contract number F30602-00-1-0545 and administered by the Air Force Research Laboratory.

**References:**


