A Flexible Architecture for Open-Domain Complex Question-Answering

Abstract

This paper describes the initial implementation and evaluation of a flexible, modular architecture for open domain question answering. The system includes an explicit planning component, and supports multiple global strategies and/or multiple implementations of individual processing modules which are selected at run time. The system also includes a global memory to store intermediate processing results for each module. The memory supports comparative evaluation of individual components, and provides a basis for future application of machine learning.

1 Introduction

Simple factoid questions can now be answered reasonably well using pattern matching. Some systems (Soubbotin and Soubbotin, 2002) use surface patterns enhanced with semantic categories and question types in order to model the likelihood of answers given the question. Furthermore, Hovy et al. (Hovy et al., 2002) have obtained good results using only surface patterns pre-extracted from the web. However, pattern-based approaches don’t represent the meaning of the patterns they use, and it is not clear whether they can be generalized for more difficult, non-factoid questions.

Open domain question answering is a complex, multi-faceted task, where question type, information availability, user needs, and a combination of text processing techniques (statistical, NLP, etc.) must be combined dynamically to determine the optimal answer. For more complex questions, a more flexible and powerful control mechanism is required. For example, LCC (D. Moldovan and Surdeanu, 2002) has implemented feedback loops which ensure that processing constraints are met by retrieving more documents or expanding question terms. The LCC system includes a passage retrieval loop, a lexico-semantic loop and a logic proving loop. The IBM PIQUANT system (Carroll et al., 2002) combines knowledge-based agents using predictive annotation with a statistical approach based on a maximum entropy model (Ittycheriah et al., 2001).

Both the LCC and IBM systems represent a departure from the standard pipelined approach to QA architecture, and both work well for straightforward factoid questions. Nevertheless, both approaches incorporate a pre-determined set of processing steps or strategies, and have limited ability to reason about new types of questions not previously encountered. Practically useful question answering in non-factoid domains (e.g., intelligence analysis) requires more sophisticated question decomposition, reasoning, and answer synthesis. For these hard questions, QA architectures must define relationships among entities, gather information from multiple sources, and reason over the data to produce an effective answer. As QA functionality becomes more sophisticated, the set of decisions made by a system will not be captured by pipelined architectures or multi-pass constraint relaxation, but must be modeled as a step-by-step decision flow, where the set of processing steps is determined at run time for each question.

In this paper, we describe the JAVELIN QA architecture, which includes a general, modular infrastructure controlled by a step-by-step planning component. JAVELIN combines analysis modules, information sources, user discourse and answer synthesis as required for each question-answering interaction. JAVELIN also incorporates a global memory, or repository, which maintains a linked set of object dependencies for each question answering session. The repository can be used to provide a processing summary or answer justification for the user. The repository also provides a straightforward way to compare the results of different versions of individual processing modules running on the same question. The modularity and flexibility of the architecture provide a good platform for component-based (glass box) evaluation.

In Section 2, we provide an overview of the JAVELIN architecture and processing modules. In Section 3, we present some initial results from run-
ning an early version of the system on the TREC QA task. In Section 4, we conclude with a summary of ongoing and future work.

2 Architecture Overview

JAVELIN is a flexible, extensible, object-oriented architecture that separates the details of individual operations (e.g., taggers, parsers) from the context(s) in which they are used. The architecture is designed to support both system and component-level evaluation, so that competing strategies and components can be compared in terms of various performance criteria.

![Diagram of JAVELIN architecture](image)

Figure 1: The JAVELIN architecture. The Planner controls execution of the individual components via the Execution Manager.

The current system is shown in Figure 1. It includes a variety of modular components which perform different question answering functions, including question analysis, document retrieval, answer candidate extraction, and answer selection. A centralized Planner module, supplied with a model of the question-answering task, selects and sequences execution of individual components, enabling the system to generate multiple processing strategies and replan when necessary. The actual execution details are handled by the Execution Manager, which also stores session data (i.e. process steps, intermediate and final results) in the Repository. User interaction is coordinated by the graphical user interface (GUI) and the Answer Justification module, which provides a browsable view of the process history.

All of the modules in the JAVELIN architecture are completely separate from the rest of the system. Their internal representation and algorithms are opaque and they interact with the rest of the system through the Execution Manager using XML. In our current implementation each module can be a server or a set of servers depending on whether a component is multilingual or uses multiple strategies to achieve its goal. Feedback loops and multiple sessions are simultaneously possible through this implementation since each server can spawn a copy to handle new requests.

The JAVELIN architecture does not limit the number or type of modules it can support. The following subsections present the current implementation of the JAVELIN architecture at a component level, and illustrate its flexibility and extensibility.

2.1 QA Components

2.1.1 Question Analysis

The Question Analyzer (QA) module analyzes questions and produces a request object - a structured representation which includes: a) an assignment of question and answer types from a predefined taxonomy; b) a list of keywords, their types (e.g. word, phrase, proper-name), and alternate forms; c) constraints and features specific to each answer type; and d) a syntactic f-structure for the question.

Our question and answer type taxonomies are based on (Graesser et al., 1992), (Lehnert, 1978), and (Hiyakumoto, 2001). In JAVELIN, the question type is used to guide the overall planning strategy, and the answer type specifies the semantic category of the desired answer. For example, the question *When did the Titanic sink?* would be assigned a question type of *event-completion* and an answer type of *temporal*.

The Question Analyzer relies on word-based linguistic information as well as on sentence-level syntactic analysis in order to construct the request object. A word-level analysis of the question tokens is produced by combining information from several external resources, such as Wordnet (Fellbaum, 1998) for word morphology and semantic categorization, the Brill part-of-speech tagger (Brill, 1995), the BBN IdentiFinder (BBN, 2000), and the KANTOO Lexifier (Mitamura and Nyberg, 2000). Once the word-level analysis is complete, pattern-matching is used to assign question and answer types. The results of the word-level analysis and classification are then passed to the KANTOO GLR
Syntaxifier (Mitamura and Nyberg, 2000) to create the f-structure.

2.1.2 Document Retrieval

The main function of the Retrieval Strategist (RS) module is to identify and retrieve documents that are likely to contain an answer to the given question. Stemming is performed at indexing time using the Wordnet morphology library. Prior to indexing, source documents are preprocessed with the BBN IdentiFinder named-entity tagger (BBN, 2000), which analyzes the source text for the following named-entity types: Organization, Time, Date, Person, Place, Name, Currency, Amount, Number, Percentage. This analysis attempts to focus subsequent retrieval on documents containing not only the relevant keywords, but relevant entity types. At indexing time, terms within a span of text identified as a named entity are stored in the index using a corresponding set of special fields. We also use a special extension that recognizes numeric expressions.

Document retrieval requests sent to the RS consist of two main parts: the request object produced by the Question Analyzer, and a set of processing constraints such as available time for retrieval, upper and lower limits on the number of documents to be retrieved, etc. The Retrieval Strategist uses the keywords and answer type information supplied in the request object to construct an initial query. Keywords are included verbatim, and although the RS module does not currently perform any keyword expansion internally, it treats any alternate forms specified for the keywords as synonyms for retrieval.

The document retrieval algorithm proceeds using an incremental query relaxation technique. The initial query is highly constrained, looking for all the keyword terms and data types in close proximity to each other. At each subsequent iteration, the algorithm relaxes one or more parameters in the query, such as the word proximity window. This assumes that documents containing answers will contain clusters of keywords and data types in closer proximity. The algorithm terminates once the number of documents retrieved is equal to the upper limit, or no additional relaxation steps are possible.

The Retrieval Strategist differs in some important ways from IR systems used in other open-domain QA systems. Instead of having direct implementation dependencies on other modules of the system, the RS is defined as a separate component reachable only through a well-defined XML interface. The RS is the only module having direct access to document databases and this allows for the complexities of dealing with multiple transport protocols, query formats, or document encodings to be hidden from the rest of the system. This makes it easier to support flexible search scenarios, with heterogenous and/or distributed collections of document databases.

Many QA systems use an IR scheme that retrieves and assigns scores to all documents in one step. The incremental query relaxation technique we use is more general. It introduces some additional overhead because of the increased number of queries, but also allows us to operate adaptively. For example, we might vary the query expansion depending on the nature of the results at each step. The iterative method also gives an implicit ranking based on the relaxation step at which a document was found, with fewer relaxations implying higher rank. This gives us a way to combine results from different search engines for documents found at different relaxation levels.

2.1.3 Answer Candidate Extraction

The Information Extractor (IX) is responsible for identifying and scoring answers from a set of potentially relevant documents. The goal of the IX is to find small relevant passages, identify the candidate answer in each such passage, and score each \{passage, answer\} pair by estimating the probability that it answers the original question.

We have implemented two strategies for answer extraction, each based on a different classifier: K-Nearest-Neighbor (KNN) implementing KD-trees, and a decision tree using the \textit{c4.5} algorithm. For the KNN method, a parameter optimization was performed in Matlab yielding the number of nearest neighbors per query \(n_m = 25\), positive neighbor emphasis \(\alpha = 1.7\), and exponent for the nearest neighbors \(\beta = 1.5\). The IX shows the benefit of a multi-strategy approach since different classifiers have different behavior for different kinds of text, different
answer types, and different relaxation degrees.

The IX requires a set of relevant documents from which candidates will be extracted, the request object, and processing constraints such as the minimum number of passages to be extracted and the time limit for the task. The IX output consists of \{\text{passage, answer, score}\} tuples, where the score is a measure of the degree to which the passages and answers solve the original question.

The first step in the information extraction process is a loose passage filter which considers all passages that meet a minimum requirement based on a relaxed version of the task. This step produces a collection of potentially good \{\text{passage, answer}\} pairs from the document set. Then the IX computes a set of features for each such pair. These features are subsequently used by the classifier to assign relevance scores to each answer candidate.

Currently, the features supplied to the classifier are based on the surface form and surface statistics of the passages, and make use of part-of-speech analysis, named-entity tagging, and morphological normalization. These features identify patterns and check for the existence of various cues that indicate whether the \{\text{passage, answer}\} under scrutiny is relevant to the original question. For example, features may include patterns such as “ONOUN was QVERB in | on | at ANOUN”, surface statistics such as the number of query terms in a given passage, or a measure of punctuation occurring between the query terms and the answer term. Given such features, the classifiers are trained off-line and parameters are tuned for better generalization.

2.1.4 Answer Selection

The Answer Generator (AG) module produces a list of answer candidates sorted according to their combined confidence score. The AG receives potential answer candidates, their individual confidence scores, and the passages they were extracted from. The current implementation of the AG combines answer clustering, normalization, and formatting.

During the normalization step, the potential answer candidates are put into canonical form, and their scores are normalized to [0, 1]. Then all but the highest ranked candidate from each document is removed. The canonical form for a given answer depends on the answer type.

Confidence score normalization is accomplished by taking the input confidence score range, [0, 2200], using the frequency of answer confidence scores to fit a normal distribution over this range, and directly mapping all values in \([-2\sigma, 2\sigma]\) to [0, 1]. All outlying confidence values are set to 0 and 1 respectively.

The answers are then grouped into clusters, where each member of the cluster supports the most specific member of the same cluster. As with canonicalization, the definition of “supporting” depends on the answer type. For example, a numeric answer \(A\) would support numeric answer \(B\) if \(A\) and \(B\) have the same units and \(|A-B|/B < 0.05\), while a person-name answer would support another if they had the same last name.

The confidence scores for an answer cluster are then computed as the probability that at least one member of the cluster is correct given that all answers in the cluster are independent and equally weighted. For a cluster \(C\) containing answers \(A_1, A_2, ..., A_n\) with scores \(S_1, S_2, ..., S_n\), the confidence for the entire cluster \(T_C\) are computed with the following formula:

\[
T_C = 1 - \prod_{i=1}^{n} (1 - S_i)
\]  

Due to the modular nature of the system, the planner can dynamically select an arbitrary answer extraction module and the Answer Generator will transparently deal with combining similar answers and normalizing the output. More importantly, the Answer Generator can take input from several different modules and, using techniques similar to boosting, select the most probable answer across all answer candidates, which tends to be more accurate than using any extraction module individually.

2.2 Planning

The Planner is responsible for selecting and issuing module command sequences to maximize the expected utility of the information JAVELIN produces, taking into account the available system resources and constraints such as total execution time. The primary advantage planning offers is the flexibility to dynamically select between different versions of the system components, enabling JAVELIN to generate different QA strategies at run-time, rather than relying on a fixed pipeline architecture.
Upon receiving a new question, the Planner calls the Question Analyzer via the Execution Manager to perform the initial question analysis, from which it generates a planning problem describing the initial state and information goal. The Planner then begins the planning and execution process, continuing until it has met the goal criteria (has found an answer or set of answers with sufficiently high expected utility), or has exhausted its available resources. At this point, it returns the answer or a failure message. The Planner module also provides an “interactive” mode of operation, which enables it to request user feedback during the planning process.

Internally, the Planner uses a forward-chaining utility-based planning and execution algorithm that performs a best-first search across the set of possible information states (Hiyakumoto and Veloso, 2002). It is supplied with a domain model describing the features of the information state on which the planning process will be based, and the actions the Planner can select between, namely the various modules that comprise the system, the preconditions under which they are applicable, and their possible effect on the information state. It is also given a problem statement consisting of: an initial state, a predefined utility-function, a utility success threshold, and a value specifying a confidence threshold for termination. Beginning with the initial state and an empty plan, the Planner evaluates the successor states of each candidate action, selecting the one with the highest expected utility to add to the partial plan. The internal planning state is updated to reflect the projected outcome, and the action selection process repeats. At each step, the algorithm considers the tradeoff between executing the first unexecuted action in the plan, and continuing to plan with the uncertainty of the projected states. If an execution step is carried out, it is followed by an assessment of the need for replanning. The algorithm terminates when all steps in the plan have been executed and the confidence and utility thresholds for goal satisfaction are met, or there are no additional actions the Planner can take.

2.3 Execution Coordination

The Execution Manager (EM) performs central coordination of batch and interactive question processing. Under the Planner’s control, the EM can execute feedback loops or simulate a simple pipelined architecture. When feedback loops are enabled, the EM relies on the Planner module to determine module sequencing, and acts as a broker between the Planner and the other modules in the system. The EM translates individual Planner requests into the input XML required by the requested module, runs that modules, and saves all results and process history in the Repository.

2.4 Modular Integration

Each JAVELIN component is implemented as an independent server and communicates with the Execution Manager through XML. This design choice provides a flexible encapsulation of input and output which supports integration of modules running on different platforms. Appropriate DTDs are defined for each XML object, and data objects are verified via XML parsing. Each XML object passed in the run-time system corresponds to a static data object (e.g. table row) in the Repository. Extending the existing protocols can be accomplished through straightforward revision of the XML DTD and corresponding entity definition in the Repository.

2.5 Data Storage

The Repository is the information backbone of the system. It stores the information objects produced at each stage of the question-answering process (e.g., question text, documents, answer candidates) as well as process information used at the infrastructure level to maintain system coherence (e.g., batch test results, servlet links for the GUI, processing time statistics). The Repository module consists of a relational database and a file system database. The relational database was developed using Microsoft SQL 2000, and currently holds 45 tables and over 3 million records.

2.6 Answer Justification

The Answer Justification (AJ) component produces an audit trail of the processing performed by the system during the course of answering a question. The purpose of this audit trail is twofold: it supplies evidence supporting an answer’s correctness, and documents the processing decisions made by the system. In the future, the audit trail could enable a user to interactively provide feedback to the system, help
to guide processing choices, or correct errors in the system’s knowledge base.

The current implementation of the Answer Justification module provides the user with a concise trace and a summary of the information produced during processing, as well as a detailed trace of each step of the execution. Question summaries are generated automatically from the repository data, and include: the question and answer type assigned by the system, the number of documents and answer candidates generated, the highest ranked answer and its associated confidence score, and when applicable, the associated TREC relevance judgement. The summary also includes hyperlinks which reveal additional levels of detail (e.g., full text of individual documents, answer passage text, etc.).

2.7 Run-Time Strategy Selection

Run-time strategy selection in JAVELIN is controlled by both the domain model and the utility function provided to the Planner. Each operator in the domain model corresponds to module and system functions the planner can control, including requests for user feedback. Operator preconditions specify information state characteristics that must hold in order for the operator to be considered by the planner when selecting the next action to add to the plan. Consequently, run-time characteristics of the information state influence the sequence of actions selected for execution. For example, an operator requiring user interaction, such as \texttt{ASK\_USER\_FOR\_MORE\_KEYWORDS}, is only considered when the information state contains the ‘interactive session’ predicate, which only holds if the interaction flag is set when a new question is posed.

In addition to the domain model, strategy selection is guided by the utility function supplied to the planner at run-time. This function is a linear weighted combination of information quality estimates and resources (such as execution time) which the planner uses to estimate the value of the information states it expects to reach by executing an action. During the planning process, the planner selects an action sequence to maximize the expected value of this function. Changing the relative weights or constituents of this function changes the relative value placed on the projected states and their associated operators. For example, if greater weight is given to minimizing execution time, actions with longer average execution times will lead to states with lower expected utility, and are less likely to be selected.

3 Evaluation

A strength of JAVELIN’s modular architecture is that different module versions can be tested simply by changing the server port and host information in the configuration file used by the Execution Manager. This makes it easier to determine the relative contributions of each module, as well as to test incremental improvements to the system. The use of a planner also makes it possible to incorporate multiple modules performing the same function (e.g., the KNN and DT versions of the IX module). Each new module is integrated by updating the EM configuration file, adding a new operator to the planner domain with appropriate preconditions, and adding execution support (if necessary) to create and interpret any module-specific input and output.

Our standard procedure for evaluating changes to individual question-answering modules consists of: running a simulated “pipeline” baseline with the existing system components, replacing a single module with a revised version, and rerunning the pipeline test. To emulate a simple single-pass pipeline architecture, the planner is disabled and the EM calls each of the four primary system modules in sequence (i.e., QuestionAnalyzer, RetrievalStrategist, InformationExtractor, and AnswerGenerator). The relative contributions of the Planner module can be evaluated in an analogous manner by comparing the system’s performance with the planner against the simulated baseline without it.

Table 1 summarizes our results for the 500 questions of the TREC-11 QA track in July 2002. It is important to note that these results are based on an early version of the system which did not include the Planner module and the user interface, and supported only these answer types: locations, temporal-expressions, proper-names, and numeric-expressions. All other questions were assigned a default “object” category. In each run, the system used only the top 15 documents returned by the Retrieval Strategist module. Two TREC runs were submitted, one using the decision tree version of the Information Extractor, the other using the KNN version.
We are currently in the process of evaluating the system with the Planner module using non-interactive variations of the domain presented in the previous section. Because of the challenges entailed in including a user in an evaluation of more than a few questions, our efforts are focused on automated operators for answer-type reclassification, additional document retrieval, and query-expansion. Our goal is to identify contexts where such operators are useful, as illustrated by the example execution sequences of Figure 2. Here, the question “Where is bile produced?” is initially classified as having a ‘location’ answer type, which, because of the IX modules’ limited answer type coverage, produces very low-confidence answers focused on geographic locations. By introducing an answer type reclassification operator that provides a back-off strategy to an ‘object’ answer type, applicable only when the highest answer confidence score is below some threshold (0.05 in this example), the Planner enables the system to produce more reasonable results.

### 4 Ongoing and Future Work

Following the TREC evaluation, we utilized the answer justification capabilities of the system to isolate the cause of each wrong answer (question misclassified, wrong documents retrieved, wrong passages extracted, wrong answer(s) receive highest score). We have implemented and are currently testing an updated set of processing modules which incorporate corresponding improvements. We are also in the process of testing the baseline system with the Planner module fully integrated, to determine how the current planning operators can improve performance over a simple pipelined strategy. If this paper is accepted for presentation at HLT-NAACL 2003, the final version of the paper will include both sets of test results. In addition to this ongoing work, we are

<table>
<thead>
<tr>
<th>ATYPE</th>
<th>TOTAL</th>
<th>C&lt;sub&gt;DT&lt;/sub&gt;</th>
<th>S&lt;sub&gt;DT&lt;/sub&gt;</th>
<th>C&lt;sub&gt;KNN&lt;/sub&gt;</th>
<th>S&lt;sub&gt;KNN&lt;/sub&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>lexicon</td>
<td>28</td>
<td>2</td>
<td>0.07</td>
<td>2</td>
<td>0.07</td>
</tr>
<tr>
<td>location</td>
<td>109</td>
<td>26</td>
<td>0.24</td>
<td>22</td>
<td>0.20</td>
</tr>
<tr>
<td>nameexp</td>
<td>66</td>
<td>11</td>
<td>0.17</td>
<td>11</td>
<td>0.17</td>
</tr>
<tr>
<td>object</td>
<td>92</td>
<td>3</td>
<td>0.03</td>
<td>6</td>
<td>0.07</td>
</tr>
<tr>
<td>org-name</td>
<td>18</td>
<td>3</td>
<td>0.17</td>
<td>1</td>
<td>0.06</td>
</tr>
<tr>
<td>pers-name</td>
<td>46</td>
<td>7</td>
<td>0.15</td>
<td>8</td>
<td>0.17</td>
</tr>
<tr>
<td>prop-name</td>
<td>32</td>
<td>3</td>
<td>0.09</td>
<td>6</td>
<td>0.19</td>
</tr>
<tr>
<td>temporal</td>
<td>101</td>
<td>19</td>
<td>0.19</td>
<td>29</td>
<td>0.29</td>
</tr>
</tbody>
</table>

**Figure 2:** Example question illustrating a context where Planning can improve performance.

The results for these runs were quite similar. Only about 20-25% of the questions were answered correctly, due to the limited coverage of answer types and the omission of the Planner module. A breakdown of our performance by assigned answer-type is provided in Table 2.

<table>
<thead>
<tr>
<th>Submitted Run</th>
<th>DT</th>
<th>KNN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Correct (R)</td>
<td>75</td>
<td>86</td>
</tr>
<tr>
<td>Inexact (X)</td>
<td>13</td>
<td>12</td>
</tr>
<tr>
<td>Unsupported (U)</td>
<td>10</td>
<td>8</td>
</tr>
<tr>
<td>Wrong (W)</td>
<td>402</td>
<td>394</td>
</tr>
<tr>
<td>Confidence-weighted score</td>
<td>0.251</td>
<td>0.209</td>
</tr>
<tr>
<td>No-answer precision</td>
<td>(12/49) 0.152</td>
<td>(10/61) 0.164</td>
</tr>
<tr>
<td>No-answer recall</td>
<td>(12/46) 0.261</td>
<td>(10/46) 0.217</td>
</tr>
</tbody>
</table>
focusing on other key improvements to JAVELIN:

- **Shallow Parsing for Answer Extraction.** We are in the process of incorporating a natural language parser to extract the predicate-argument structure of possible answer passages, to improve the accuracy of the system beyond the current feature-based approach. The new extractor will be integrated and tested as an alternative implementation of the IX module.

- **Planning for Question Decomposition.** Some complex questions require decomposition into more than one sub-question before retrieval. For example, *What is the occupation of Bill Clinton’s wife?* might require an initial question to retrieve the appropriate named entity, following by a secondary question to retrieve the desired attribute. We are extending JAVELIN to handle this type of compositional processing.

- **Interactive Question Answering.** The most recent version of the system includes planner operators which can interact with the user to determine whether the system has assigned the correct answer type to the question, or to adjust the set of keywords associated with the question. One challenge we face is to determine how to evaluate the system when such user interaction is enabled.

- **Multilingual Data Sources.** We are in the process of extending the system to work with Japanese and Chinese corpora. The initial version will perform keyword query translation (as done in cross-lingual information retrieval) and return target-language documents to the user.

Although each of these additional capabilities represents a significant extension to the baseline system, all of these extensions can be implemented within the existing architecture, due to the flexible way that modules are integrated in the JAVELIN architecture.

**References**


