The JAVELIN Question-Answering System at TREC 2002

E. Nyberg, T. Mitamura, J. Carbonell, J. Callan,
K. Collins-Thompson, K. Czuba, M. Duggan, L. Hiyakumoto,
N. Hu, Y. Huang, J. Ko, L. Lita, S. Murtagh, V. Pedro, D. Svoboda
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213-3891

1 Introduction

This paper describes the JAVELIN approach for open-domain question answering (Justification-based Answer Valuation through Language Interpretation), and our participation in the TREC 2002 question-answering track. The main scientific tenets underlying JAVELIN are:

- **QA as Planning**. Question Answering is a complex multi-faceted task, where question type, information availability, user needs, and NLP sophistication of QA modules all combine dynamically to determine the optimal QA strategy. We propose a general modular infrastructure controlled by a planner, which combines analysis modules, information sources, user discourse and answer synthesis as needed for each question-answering interaction. Multilinguality and multi-information-source-type (text, HTML, DBs, KBs, etc.) will be integrally supported.
- **Universal Auditability**. Every step, from question analysis to answer generation, creates and updates a detailed set of labeled dependencies to form a traceable network of reasoning steps. This dependency network will provide the basis for refinement dialogs, efficient recomputation and refocusing, reason maintenance, user-driven knowledge auditability, and machine learning for question answering subtasks and control strategies.
- **Utility-based Information Selection and Fusion**. The expected utility of information is used to guide planning decisions and produce confidence scores for the resulting answers, taking into consideration critical facets including the task context, value of the information to the user, resource constraints, and the accuracy of individual analysis modules.

Past work on open-domain question answering has focused on particular aspects of the problem which provide convenient idealizations:

- *QA as Information Retrieval.* Systems that rely primarily on traditional IR techniques attempt to find the most relevant documents for a query. If shorter answers are required, some method is used to identify the most relevant passage in the document, typically via some passage ranking scheme based on query terms [12, 18]. IR-based approaches can fail to answer a question when the answer is not directly found in the text, but must be inferred from the content.
- *QA as Information Extraction.* Systems that rely primarily on traditional IE techniques use a template-based approach widely explored by participants in the TIPSTER and MUC competitions. Questions and answers are associated with slot-filler structures, which are fully or partially matched by the contents of searched documents. Answers are provided by extracting the contents of filled templates. When used for QA, this approach can be linked with special forms of indexing, which tag named entities (people, organizations, locations, etc.) in the original corpus [17, 16]. IE techniques alone have difficulty with broader, vaguer questions, where the answer must be derived by fusing facts from several passages, and cannot be located in one particular template fill.

- *QA as Natural Language Processing.* More recent systems have begun to improve on the performance of purely IR or IE based system through the use of NLP techniques that include deeper semantic analysis. A common use of NLP is in question analysis, where the original question is parsed to some degree, in order to understand it well enough to: a) perform selective query expansion based on meaning, rather than a "bag of words"; b) focus the search for the answer on precisely the unit of meaning which is sought (e.g., the particular named entity that answers a "who is" or "where is" question) [4, 6, 11, 8]. To date, systems have focused on NLP and semantic processing as a way of "boosting" or filtering IR-based or IE-based methods. The limited scope, content and availability of large-scale ontological resources (such as WordNet) is a challenge.

The most successful recent systems, such as the Falcon system developed at SMU [8], acknowledge that a hybrid approach, with several levels of sophistication, can produce better results than systems that rely purely on IR or IE approaches with a bit of parsing thrown in. Perhaps the most efficient way to approach QA is to deploy the simplest technique that achieves a high-quality answer for a given question, on the assumption that simplicity implies a more rapid response. Significant progress on open-domain question answering could be achieved by systems that incorporate the most sophisticated language processing technologies, when appropriate, and fall back to simpler, pattern-driven techniques where more sophisticated techniques fail. A general framework for multi-strategy QA will require advances in how QA systems are configured (using a modular architecture) and how they are controlled at run time (by incorporating explicit planning and reasoning components).

The remainder of this paper is organized as follows: Section 2 describes the JAVELIN system as it was implemented at the time of the TREC question-answering evaluation, and briefly presents several key extensions to the system completed shortly afterwards; Section 3 presents our results for the TREC QA task; Section 4 summarizes our post-TREC analysis; and Section 5 closes with a discussion of lessons learned and future directions.

2 System Architecture

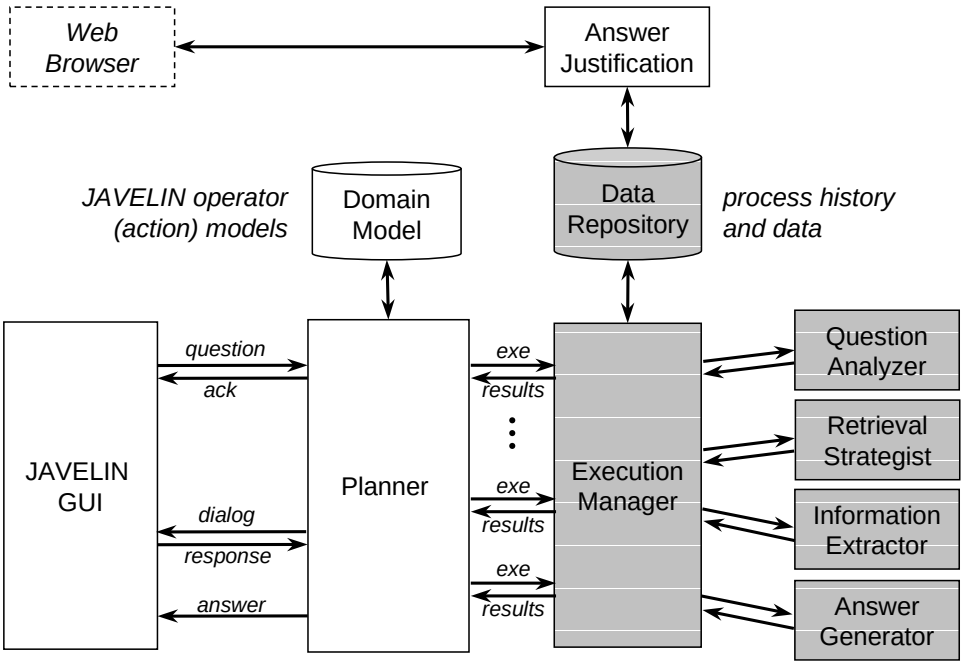

Figure 1: The JAVELIN architecture. The Planner operates as a service to the user interface, and controls execution of the individual components via the Execution Manager. Components included in the TREC test are shaded. Those shown in white were added subsequently.

JAVELIN is based on 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 component-level evaluation, so that competing strategies and operators can be compared in terms of various performance criteria.

The current system, depicted in Figure 1, integrates a variety of modular components that perform individual 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 takes care of storing of session data (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.

It is important to note that only the shaded modules of Figure 1 were used in our TREC QA submission. The remaining components, shown in white, were integrated after TREC. The next subsections describe the implementation of the modules comprising the system, focusing first on those which were part of the TREC evaluation, followed by a description of our post-TREC extensions.

2.1 Components Used in the TREC Evaluation

2.1.1 Question Analysis

The Question Analyzer (QA) is responsible for producing an initial analysis of the question text on which the rest of the system, including the Planner, bases its processing. Given a question sentence as its input, the QA generates a *request object*, a semi-structured representation of the question containing:

- the classification of the question according to a predefined taxonomy of question and answer types
- a list of keywords, their types (either word, phrase, or proper-name), and alternate forms
- answer-type specific constraints and sequential features used during answer candidate extraction
- an f-structure representation of the question

Central to the entire system is our question type and answer type taxonomy, based on [7] [13] [9]. In JAVELIN, the question type is used to guide the overall planning strategy and the answer type specifies a semantic category for the desired answer. Table 1 shows two examples of this classification, and Table 2 lists the top-level answer types recognized by the Question Analyzer at the time of TREC. These represent a small fraction of the answer types the system will eventually include.

The Question Analyzer relies on both word-based linguistic information and sentence-level syntactic analysis to construct the request object. A word-level analysis of the question tokens is produced by combining information from several external resources, namely Wordnet [5] (for word morphology and semantic categorization), the Brill part-of-speech tagger [2], BBN IdentiFinder [1] , and the KANTOO Lexifier [15]. Once the word-level analysis is complete, pattern-matching is used to assign a question and answer type (with the default classification of "object"), and to identify answer-type specific constraints. The results of the word-level analysis and classification are then passed to the KANTOO GLR Syntaxifier [15] to create the f-structure.

Table 1: Examples of JAVELIN'S question and answer type classification.

Question	Question Type	Answer Type
When did the Titanic sink?	event-completion	temporal
Who was Darth Vader's son?	concept-completion	person-name

Table 2: Top-level answer type categories recognized in JAVELIN at the time of the TREC evaluation.

object
proper-name
temporal
location
numeric-expression

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 current question. As a secondary function, the RS module also fulfills any other document repository requests made by individual system components, such as requests from the Information Extractor for specific documents or passages.

Internally, the Retrieval Strategist uses the Inquery retrieval system [3]. 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 [1], 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 data types. At indexing time, any 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 to Inquery 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 (upper and lower limits on the number of documents to retrieve and a time limit). 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 likely answer type is mapped to a set of named-entity types corresponding to those used by the BBN IdentiFinder named-entity tagger. For example, a 'temporal' answer type is mapped to either a 'Time' or 'Date' named-entity tag. These types are treated as special keywords included in the terms passed to the retrieval system.

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 retrieval upper limit, or no additional relaxation steps are possible.

The relaxation parameters are:

- The Inquery proximity/belief operator used to combine keywords. At each relaxation step, we either keep the same operator but expand the window size, or start with a new, more general operator. The operator applies to all keywords given in the query. For example, initially all keywords must be found within a proximity of three words. We then relax the operator to consider unordered 20-, 100-, and 250- word windows, followed by document-wide probabilistic AND, and so on.

- Phrase proximity, for any phrase keywords. This is usually kept at 3 words or less, until later in the relaxation regime, when it is expanded to the PHRASE operator.

- Proper name proximity, for any proper name keywords. This is usually kept at 3 words or less until very late in the relaxation regime, when it is expanded to the PHRASE operator.

- The inclusion or exclusion of the special named-entity keywords corresponding to the answer type. This alternates between on and off at every relaxation step.

The RS module output is a ranked list of document IDs. For some answer types, such as numerical types, offset information is provided with each document ID to simplify subsequent processing.

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.

The IX takes as input: a set of documents from which candidates will be extracted, the request object produced by the Question Analyzer, and processing constraints such as the minimum number of passages to be extracted or the time limit for the task. The IX output consists of passage-answer-score triplets, 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 $\{answer, passage\}$ pairs from the document set. Then the IX computes a set of features for each $\{passage, answer\}$ pair. These features are subsequently used by a 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 $\{passage, answer\}$ under scrutiny is relevant to the original question. For example, features may include patterns such as "QNOUN 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.

There are currently two versions of the IX module, each based on a different classifier: K-Nearest-Neighbor (KNN) implementing KD-trees, and a decision tree using the $c4.5$ algorithm. For the KNN version, a parameter optimization was performed in Matlab yielding the number of nearest neighbors per query $nn = 25$, positive neighbor emphasis $\alpha = 1.7$, and exponent for the nearest neighbors $\beta = 1.5$.

2.1.4 Answer Selection

The Answer Generator (AG) module's purpose is to produce a list of answer candidates sorted by confidence score. Its input is a list of potential answer candidates, their associated scores assigned by the IX, and the passages they were extracted from. The basic algorithm used by the AG is as follows: 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. For TREC, location, numeric, time and person-name answer candidates were canonicalized into type-specific formats, while all other types were normalized simply by removing punctuation and converting to lowercase. A numeric answer is canonicalized to a pair containing the unit, which may be empty, and the value. A location, such as a country, is converted to the standard short form as specified in the CIA World Factbook and converted to lowercase. Dates are cast in $mm/dd/yyyy$ format with unknown pieces marked. Names are split into first and last names, either possibly empty, and converted to lowercase.

The 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 $\frac{|A-B|}{B} < .05$, 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:

$$T_C = 1 - \Pi_{i=1}^{n}(1 - S_i) \tag{1}$$

2.1.5 Execution Coordination

For the TREC evaluation, the Execution Manager was used to coordinate batch processing of the test set, calling each of the four components in a fixed sequence to imitate a pipelined architecture.[1] At each step, the EM constructs the module-specific input XML, calls the next module in the sequence with that input, and stores the resulting module output in the system Repository for later use.

2.1.6 Data Storage

The Repository stores all the information the other modules produce, serving as the information backbone of the system. It stores not only the objects that represent information in each part of the process (question text, answers, answer type, etc ...) but also information that helps maintain system coherence and information used at the infrastructure level (e.g., for batch tests, 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 [14], and currently holds 45 tables and over 3 million records. It was designed for high loads and should be able to hold in excess of 100 million records.

2.2 Post-TREC Extensions

2.2.1 Planning

One of the major extensions to the JAVELIN system subsequent to the TREC submission was to integrate the Planner module, which now serves as the controller for the question answering process. 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.

The Planner operates as a service for the JAVELIN GUI, and communicates with the rest of the system via the Execution Manager. Upon receiving a new question, the Planner calls the Question Analyzer via the EM 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 [10]. 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.

[1] In the complete system, the EM relies on the Planner module to determine module sequencing, and simply acts as a broker between the Planner and the other modules in the system. It translates individual Planner requests into the input XML required by the requested module, saves results and process history information in the Repository, and provides user authentication.

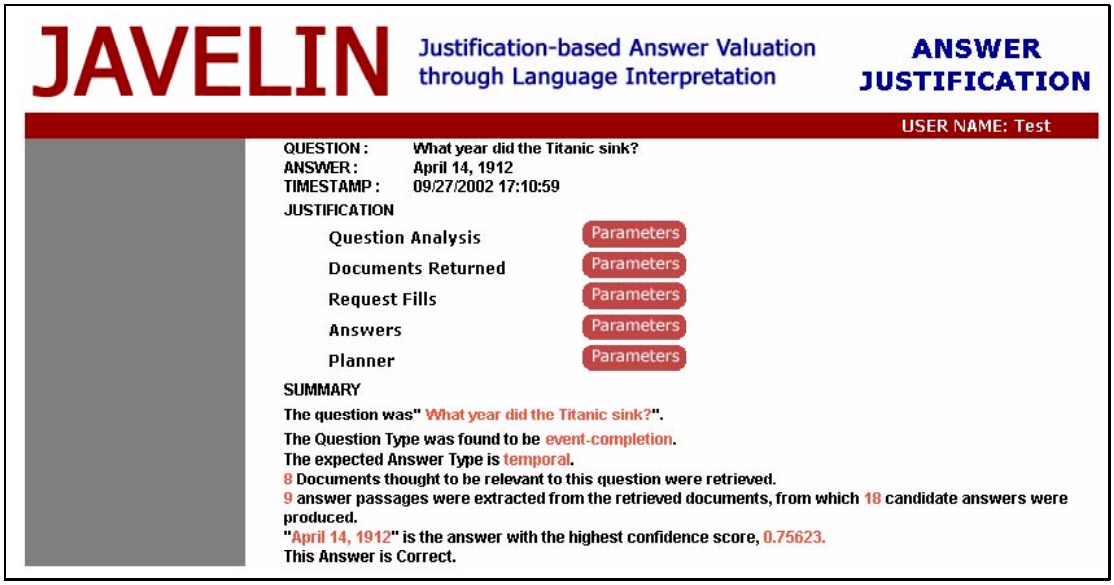

Figure 2: Screenshot of the output produced by the beta version of the Answer Justification module.

2.2.2 Answer Justification

The Answer Justification module produces an audit trail of the processing performed by JAVELIN during the course of answering a question. The purpose of this audit trail is twofold: first, it supplies evidence regarding an answer's correctness, and second, it documents the processing decisions made by the system. Our eventual goal is to use this module to enable a user to interactively provide feedback to the system, help guide processing choices, or correct system knowledge errors as they arise.

The current beta version of the Answer Justification module provides the user with a read-only display containing a brief summary of the information produced during processing, and 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 classification 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. A sample summary produced by the Answer Justification module is shown in Figure 2.

2.2.3 GUI

The JAVELIN GUI (Figure 3) is the front-end to the Planner-driven system. It is a Java application that resides on the user's desktop, forwarding user requests to the Planner module, and displaying the resulting answer(s). The GUI also provides a mechanism for the Planner to interactively request feedback from the user, by displaying Planner-initiated dialogs and returning user responses.

3 TREC Results

A preliminary version of JAVELIN without the Planner module and user-interface components was tested on the TREC QA track evaluation data in July 2002. This version used the Execution Manager to invoke each of the four major components in the following fixed sequence: the Question Analyzer, the Retrieval Strategist, the Information Extractor, and the Answer Generator. The goal of this initial test was to provide us with a baseline for subsequent evaluation of the complete system with the Planner and improved components.

Two TREC QA runs were submitted, one using the decision tree version of the Information Extractor, and the other using the KNN version to identify candidate answer passages. In each run, the system considered only the top 15 documents returned by the Retrieval Strategist module.

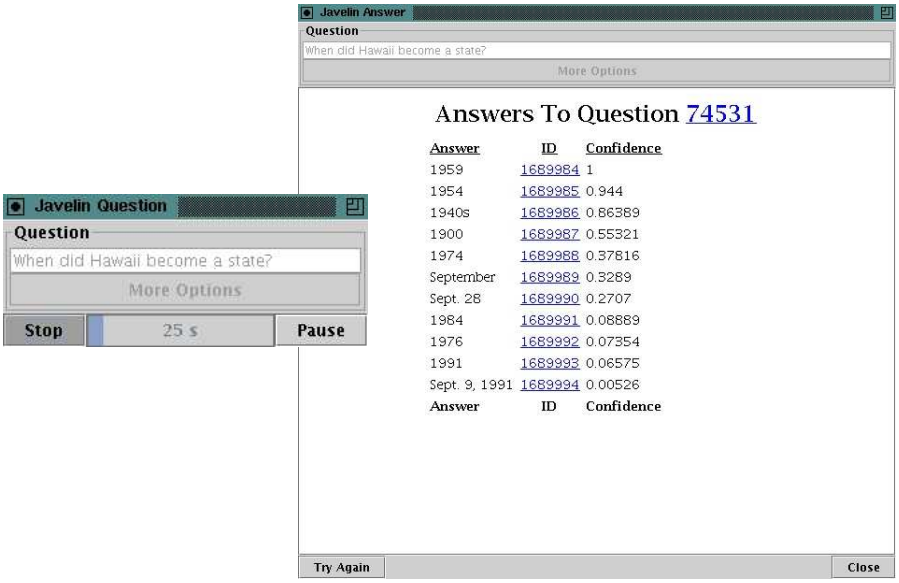

Figure 3: Screenshots of the JAVELIN GUI illustrating the main control window used to pose a question to the system, and the resulting answers displayed.

Table 3: JAVELIN TREC11 QA results using DT-based (CMUJAV000495) and KNN-based (CMUJAV000501) answer candidate identification, with retrieval of the top 15 documents.

Submitted Run	CMUJAV000495 (DT)	CMUJAV000501 (KNN)
Correct (R)	75	86
Inexact (X)	13	12
Unsupported (U)	10	8
Wrong (W)	402	394
Confidence-weighted score	0.251	0.209
No-answer precision	(12/79) 0.152	(10/61) 0.164
No-answer recall	(12/46) 0.261	(10/46) 0.217

Table 3 summarizes our official TREC results for these runs. The KNN run produced more correct answers than the DT run (86 vs. 75), but the DT run received the higher weighted score once the system's confidence estimates were taken into account (0.251 vs. 0.209). Both runs exhibited comparable precision in identifying questions without an answer (0.152 for the DT, 0.164 for the KNN run).

It is premature to draw conclusions about the relative performance of the two classifiers, given the limited size of the training and test sets. However, this preliminary evaluation did enable us to make several observations about the system's performance in general. The next section describes our post-TREC analysis.

4 Analysis

In the interim between performing the TREC QA evaluation and receiving our official scores, we conducted an internal performance analysis for a subset of the TREC 2002 question set. Project members manually identified correct answers for 193 questions, along with at least one document containing the answer. We then compared our manually generated answer key with the system's output to determine whether or not the system returned the correct answer, and if not, at what step the first failure occurred.

Table 4 summarizes the results of this analysis. For each module, we computed the number of questions where a module exception occurred, the number for which the module performed correctly, the number for which it might have performed correctly, and the number that resulted in erroneous output. The total

	Module	Total Questions	Exceptions	Correct	Possibly Correct	Incorrect
	EM	193	0	192 (99.5%; 99.5%)	NA	1 (0.5%; 0.5%)
	QA	192	0	154 (80.2%; 79.8%)	23 (12.0%; 11.9%)	15 (7.8%; 7.8%)
	RS	177	0	80 (45.2%; 41.5%)	70 (39.6%; 36.3%)	27 (15.3%; 14.0%)
KNN	IX unique	150	8 (5.3%; 4.2%)	37 (24.7%; 19.2%)	82 (54.7%; 42.5%)	23 (15.3%; 11.9%)
	tied			39 (26.0%; 20.2%)	80 (53.3%; 41.5%)	
	AG	119	0	47 (39.5%; 24.4%)	66 (55.5%; 34.2%)	6 (5.0%; 3.1%)
DT	IX unique	150	14 (9.3%; 7.3%)	36 (24.0%; 18.7%)	74 (49.3%; 38.3%)	26 (17.3%; 13.5%)
	tied			60 (40.0%; 31.1%)	50 (33.3%; 25.9%)	
	AG	110	0	41 (37.3%; 21.2%)	66 (60.0%; 34.2%)	3 (2.7%; 1.6%)

Table 4: Analysis of individual module performance for a subset of 193 questions from TREC 2002.

number of input questions (column 2) for each subsequent module excludes any for which an error or exception occurred earlier in the pipeline. The "correct" and "possibly correct" columns distinguish between completely correct analyses and partially correct analyses produced by the Question Analyzer, and cases where the output of the RS/IX/AG module included the correct information, but did not assign it the highest rank. Additionally, because it was possible for multiple answers to be assigned the same confidence by the IX module, we distinguished between cases where the correct answer was uniquely ranked first, and cases where the correct answer was one of several answers receiving the same high score. The fraction of the questions covered by each outcome is provided in parentheses as a relative percentage of the input questions for that module, and as an absolute percentage of the 193 questions we evaluated.

Errors occurring during question analysis were due primarily to insufficient coverage of classification patterns for question and answer type classification. This is an issue we were aware of going into the evaluation, given the limited number of answer type classes currently in use, and is being addressed as part of our ongoing development efforts.

Roughly equal percentages of the failures occurred during document retrieval and candidate extraction. For approximately 15% of the good input that the RS received, it failed to retrieve any documents (within the top 15) containing the correct answer. Likewise, in $15 - 17\%$ of the cases where at least one document containing the answer was passed to the Information Extractor, the IX failed to identify it as a candidate. While it is difficult to quantify exactly how much of an impact it had, it is almost certainly true that a significant number of these errors were due to the limited number of answer types recognized by the system. Our default strategy of assigning an "object" answer type provides very little additional information (beyond that provided by the keywords) for the Retrieval Strategist or Information Extractor to use in discriminating between documents and candidates respectively. The impact on the RS is simply a restriction on its ability to augment queries with named-entity tags. However, in the IX, where extraction and candidate scoring relies on answer-type dependent features, the impact is large.

In cases where the correct answer was included amongst the candidates passed to the Answer Generator, the clustering algorithm of the AG was sometimes able to compensate for IX ranking inaccuracies. In the KNN run, the Answer Generator successfully identified the correct answer in 9.5% of the cases where the Information Extractor did not rank the correct answer highest or gave it a tied ranking for the highest score. In the DT run, the Answer Generator was able to correctly compensate for IX ranking errors approximately 5% of the time. This smaller compensation is likely due to the fact that the decision-tree classifier produces less discriminatory (coarser-grained) confidence estimates. Cases where the Answer Generator failed to rank the correct answer highest included questions for which the input answer candidate set contained hundreds of unique answer candidates, suggesting the need for better filtering mechanisms prior to ranking, as well as more sophisticated methods of combining evidence.

To evaluate how much better our retrieval performance could potentially be using our existing query strategy and just increasing the number of documents retrieved, we computed the probability that the RS returns at least one correct document within the top N documents for several values of N. Table 5 presents the retrieval success rates for several values of N ranging from 15-120, using the previously analyzed subset of the TREC 2002 questions as the test set. Going from 15 documents to 60 documents increased the likelihood of retrieving a correct document by 11%, but doubling the number of documents again to 120 added very

Table 5: Retrieval Strategist success rates for various values of top N documents retrieved

N	Success Rate
15	0.74
30	0.80
60	0.85
120	0.86

little. Given the negative impact that a larger retrieval set has on processing time, and the potential for increased noise in the resulting answer candidate set, our use of 15 documents appears to be a reasonable trade-off between coverage and performance of the other modules in the system. We are now exploring alternate query construction strategies such as constrained query expansion to improve our retrieval success rate.

The failure rate of the Information Extractor for the TREC 2002 subset was consistent with its performance on previous TREC question sets. Table 6 summarizes the performance of the DT version of the IX module on past TREC data. The first column contains the number of questions in the test set, the second shows the number of times the correct answer was among the top five highest score answer produced by the IX, and the last column represents the number of times the correct answer existed in at least one document provided by the Retrieval Strategist. Setting aside the issue of answer type coverage, these results and an examination of individual failures suggest that more versatile extraction strategies are required in order to locate the right candidate answer more often. We are currently in the process of augmenting the system with an NLP-based version of the IX to supplement our current statistical approaches.

Table 6: Performance of the Information Extraction module on TREC question sets

Data	Corpus Size	$A\ in A^5$	$A \subset D$
TREC 8	200	71	189
TREC 9	693	218	424
TREC 10	500	119	313

5 Discussion

The JAVELIN system submitted to TREC is an integrated architecture for open-domain question answering. JAVELIN's modular approach addresses individual facets of the QA task with different modules. Question analysis addresses the taxonomy of question-answering types and type specific constraints by combining knowledge and pattern matching. Document retrieval includes query processing, document retrieval, and passage retrieval, and implements a strategy of incremental query relaxation. The information extraction module employs decision trees and KNN classifiers to identify answer passages in the relevant documents. Answer selection includes type normalization, conversion, clustering, and prototype selection. The behavior of the individual modules is coordinated by the Planner module, which controls the overall question answering process. The Execution Manager handles inter-module communication, persistent data storage and retrieval, and authentication while servicing requests made by the Planner. The Repository component supports the entire system with a large-scale, centralized data and state storage capability.

Using the decision tree classifier, the system obtained a confidence weighted score of 0.251, and using the KNN classifier the system obtained a score of 0.209. These results suggest several improvements for each module. Question analysis needs to perform deeper NLP processing. Document retrieval requires a more complex query expansion in order to improve the module's success rate. The TREC results also reflect the fact that the feature space used by the classifiers is limited; and a richer set of features and more training data are likely to improve performance. We also plan to improve the answer selection process, using external knowledge and additional local constraints, in order to combine multiple candidate answers from multiple sources.

References

[1] BBN Technologies. *IdentiFinder User Manual*, 2000.

[2] E. Brill. Unsupervised learning of disambiguation rules for part of speech tagging. In David Yarovsky and Kenneth Church, editors, *Proceedings of the Third Workshop on Very Large Corpora*, pages 1–13, Somerset, New Jersey, 1995. Association for Computational Linguistics.

[3] J. P. Callan, W. B. Croft, and S. M. Harding. The INQUERY retrieval system. In *Proceedings of the Third International Conference on Database and Expert Systems Applications*, pages 78–83, Valencia, Spain, 1992.

[4] C. Clarke, G. Cormack, D. Kisman, and R. Lynam. Question answering by passage selection. In *Proceedings of the 9th Text Retrieval Conference (TREC-9)*, 2000.

[5] C. Fellbaum. *WordNet - An Electronic Lexical Database.* Cambridge, Mass : MIT Press, 1998.

[6] O. Ferret, B. Grau, M. Hurault-Plantet, G. Illouz, C. Jacquemin, N. Masson, and P. Lecuyer. QALC - the question answering system of LIMSI-CNRS. In *Proceedings of the 9th Text Retrieval Conference (TREC-9)*, 2000.

[7] A. C. Graesser, N. Person, and J. Huber. *Mechanisms that Generate Questions*, chapter 9, pages 167–187. Lawrence Erlbaum Associates, 1992.

[8] S. Harabagiu, D. Moldovan, M. Pasca, R. Mihalcea, M. Surdeanu, R. Bunescu, R. Girju, V. Rus, and P. Morarescu. FALCON: Boosting knowledge for answer engines. In *Proceedings of The Ninth Text REtrieval Conference (TREC 9)*, 2000.

[9] L. Hiyakumoto. Planning and execution for open-domain question answering. Thesis proposal, November 2001.

[10] L. Hiyakumoto and M. Veloso. Towards planning and execution for information retrieval. In *Proceedings of the AIPS'02 Workshop on Exploring Real-World Planning*, Toulouse, France, 2002.

[11] E. Hovy, L. Gerber, U. Hermjakob, M. Junk, and C-Y. Lin. Question answering in webclopedia. In *Proceedings of the 9th Text Retrieval Conference (TREC-9)*, 2000.

[12] K. Kwok, L. Grunfeld, N. Dinstl, and M. Chan. TREC-9 cross language, web and question-answering track experiments using PIRCS. In *Proceedings of the 9th Text Retrieval Conference (TREC-9)*, 2000.

[13] W. G. Lehnert. *The Process of Question Answering.* Lawrence Erlbaum Associates, Hillsdale, NJ, 1978.

[14] Microsoft. *Microsoft SQL Server 2000.* `http://www.microsoft.com/sql/evaluation/overview/`.

[15] T. Mitamura and E. Nyberg. The KANTOO machine translation environment. In *Proceedings of AMTA-2000*, 2000.

[16] J. Prager, E. Brown, D.R. Radev, and K. Czuba. One search engine or two for question-answering. In *Proceedings of the 9th Text Retrieval Conference (TREC-9)*, 2000.

[17] R. Srihari and W. Li. Information extraction supported question answering. In *Proceedings of the 9th Text Retrieval Conference (TREC-9)*, 2000.

[18] T. Takaki. NTT DATA TREC-9 question answering track report. In *Proceedings of the 9th Text Retrieval Conference (TREC-9)*, 2000.