

Multi-Strategy Information Extraction for Question Answering

Laurie Hiyakumoto
Carnegie Mellon University
hyaku@cs.cmu.edu

Lucian Vlad Lita
Carnegie Mellon University
llita@cs.cmu.edu

Eric Nyberg
Carnegie Mellon University
ehn@cs.cmu.edu

Abstract

This paper presents a flexible approach using a utility-based planner to choose between different extraction methods in a question-answering (QA) system, enabling multiple run-time strategies. We model the QA process as a set of probabilistic actions, and use the planner to select an action sequence maximizing the expected value of information. The planner exploits redundancy and diversity by dynamically selecting among three statistical and pattern-based answer extractor actions in an end-to-end QA system. The expected value of information produced by each extractor is modeled as a function of the current question context. Experiments demonstrate that a planning-based approach can outperform a fixed strategy using the single-best extractor.

Introduction

Question answering architectures are becoming increasingly complex, incorporating a greater variety of natural language processing tools and knowledge resources. The performance characteristics of these tools and resources often vary widely: some provide broad question coverage with relatively low precision, others may be very precise but are only applicable in a very narrow domain (e.g., the Central Intelligence Agency World FactBook); some require significant computational resources (e.g., parsers), while others do not.

Although large-scale evaluations of question answering systems have been performed for several years, little has been done to understand the relative contributions of their individual components. Moreover, since the emphasis has been on fully automated systems that answer factual short-answer questions, most QA systems employ these resources in an ad hoc manner, implementing fixed pipelined architectures without consideration of resource constraints or user-interaction. Recent question answering systems have only begun to explore the value of automated feedback loops (Harabagiu *et al.* 2001), incremental relaxation of constraints (Moldovan *et al.* 2002), and the use of multiple system-level strategies (Chu-Carroll *et al.* 2003; Echihabi *et al.* 2003;

Burger & Henderson 2003) and component-level strategies (Azari *et al.* 2003; Nyberg *et al.* 2003).

As the number of available processing options continues to grow and question answering systems attempt to answer more complex questions, it is essential to have a principled method for dynamically generating different QA strategies, one that takes into account multiple information requests, enables the system to seek user feedback when appropriate, and considers both the *value* and *cost* (e.g. response time) of the information that can be provided to the user.

We address this challenge by combining a utility-based planner with an architecture consisting of modular components, each performing one of four major QA tasks: question analysis, document retrieval, answer candidate extraction, and answer selection. The planner controls the selection and execution of processing steps based on models of component performance and run-time observation.

The remainder of this paper focuses on our experience with multi-strategy QA using the planner to select among three different information extractors: a support vector machine-based binary classifier, finite state transducers, and an extractor that uses a simple proximity metric. We begin with a brief overview of each extraction technique and a description of the planner's models of the extractors, their estimated success likelihoods, and the confidence thresholds used to decide when additional extraction might be needed. We then present the results of an empirical evaluation that demonstrates the planner's potential to boost overall extraction performance. Although we have chosen to illustrate our approach by focusing solely on the problem of extractor selection, it can easily be applied to other stages in the question answering process such as document retrieval or answer generation.

Related Work

Statistical learning techniques for answer extraction have been explored with moderate success. A statistical QA system by (Ittycheriah *et al.* 2001) uses maximum entropy to model the distribution of correctness given answers and questions. (Ravichandran & Hovy 2002) show that simple surface patterns alone can be

used to answer some classes of questions. (Echihabi & Marcu 2003) argue for a noisy-channel approach to answer extraction that, although statistical in nature, accommodates the use of knowledge resources. In the same spirit, (Lita & Carbonell 2004) propose a fully statistical, instance-based approach to question answering that is easy to train and requires less human effort.

Multi-strategy approaches to QA have also emerged in recent years. (Chu-Carroll *et al.* 2003) combine two end-to-end QA systems, a knowledge-based system and a statistical system, sharing intermediate results at the analysis, retrieval, and generation stages in the process. The MITRE Corporation (Burger & Henderson 2003) has also performed system-level experiments in which the output of QA systems participating at TREC2002 were combined using distance-based metrics and n -grams of words and characters. Although no combination they tested was better than the top-performing system, their results still suggest that combining the outputs of several systems can be effective, given more training data and better features.

Multi-strategy approaches at the component-level have been introduced via feedback loops and threshold settings at intermediate points in the pipeline, enabling them to repeat stages of the QA process. (Harabagiu *et al.* 2001) made use of predefined thresholds to determine when query reformulation and additional retrieval are necessary. In more recent work, LCC (Moldovan *et al.* 2002) has developed a system that attempts to prove a question logically follows from an answer, using a reasoning mechanism based on axioms extracted from WordNet glosses. When the proof fails, the system incrementally relaxes conditions that prevent the completion of the proof. AskMSR-DT (Azari *et al.* 2003), a web-based QA system, explores the use of decision trees to provide answer quality predictions for dynamic generation of query rewriting strategies. Their approach is similar to the one described here, using the quality predictions as a heuristic to select a rewrite ordering and threshold on the number of rewrites to provide.

Approach

Our work has been conducted within the JAVELIN open-domain QA system, an architecture designed specifically to support research in multi-strategy question answering (Nyberg *et al.* 2003). The JAVELIN system consists of modular components, each performing one of four major QA processing stages distinguished by the system: question analysis, document retrieval, answer candidate extraction, and selection of a final answer. A centralized planner module, supplied with a model of the question answering task and performance estimates for individual components, controls the overall selection and execution of processing steps. It communicates with the rest of the system via an execution manager, which decouples the planner’s representation of the information state from the specific input and output formats used by the QA components. A

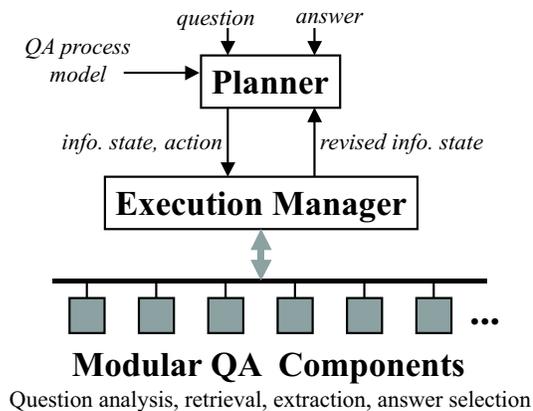


Figure 1: A high-level architectural view: The planner controls information flow among modules, selects among answering strategies, and enables user feedback depending on the relative utility and the cost of information. The execution manager is an implementation layer aware of QA module-specific interfaces.

conceptual view of the system’s architecture is depicted in Figure 1.

While any successful strategy will contain at least one pass through each of the four major processing stages, the modular planner-driven architecture also enables dynamic generation of strategies that invoke multiple modules within each stage, backtrack to previous stages if the actual outcomes fail to meet original projections, and can be tailored to the run-time features and resource constraints of the current question.

We describe our application of this approach to the problem of determining how to select among three information extractors. As such, the descriptions of the components in the following subsections are limited to the implementation of the extraction and planning components. Interested readers are referred to (Nyberg *et al.* 2003) for a description of the other modules comprising the JAVELIN system.

Answer Extraction Strategies

The role of the JAVELIN answer extraction components is to identify answer candidates within documents, based on the characteristics of the current question. The input to each extraction module is a linguistic analysis of the question and a small set of potentially relevant documents. The linguistic analysis includes syntactic processing, an expected *answer type* (the semantic category of the answer, e.g., *location*, *date*, *person*) and a *question type* (a relationship between the question terms and answer, e.g., *author-of*, *capital-of*). The output of each extractor consists of precise answer candidates in the form of short passages (e.g., dictionary definitions) or phrases (e.g., proper names), each with an associated confidence score and a 3-sentence contextual passage centered around the can-

didate. Many other QA systems blur the concepts of answer type and question type, merging them into a common ontology. However, we have found it useful to maintain the distinction, both for planning purposes and as features for the statistical extractors. When sufficient training data is available, we have also used them to develop type-specific extractor models.

Our development of multiple extractors is based on the premise that no single extraction method is likely to emerge as the best solution for all questions. We believe that optimal performance across different question and answer types requires a diverse set of extraction strategies, ranging from simple proximity measures, to statistical methods trained to discriminate correct answers from incorrect ones, to methods based on deeper linguistic and semantic processing. Thus far, three extractors have been implemented using proximity and statistical methods. Questions expressed in relatively simple syntax such as “*What is the capital of Kenya?*” and of a very common question type (e.g., *capital-of*) can frequently be answered using contextual patterns, WordNet (Miller 1995) structure, gazetteers, dictionaries, encyclopedias, and simple methods such as finite state transducers. However, these approaches often fail to reliably answer questions with a more complex structure and a less common question category (e.g., “*Which mountain range in North America stretches from Maine to Georgia?*”). Instead, statistical methods based on a large number of simple features shared across question types are likely to produce better answers. Moreover, there are questions that are ambiguous or difficult to parse (e.g., “*Where is it planned to berth the merchant ship, Lane Victory, which Merchant Marine veterans are converting into a floating museum?*”). For such questions, basic term-based proximity methods are a reasonable backoff strategy.

Extraction Using a Proximity Metric Our first answer extraction method selects candidates using a *non-linear weighted proximity metric*. Named entities (Bikel, Schwartz, & Weischedel 1999) matching the expected answer type (e.g., location, person) are identified and assigned scores based on their proximity to question terms and term expansions (conjugations, synonyms, etc). Terms found across sentence boundaries are assigned lower weights when computing the overall answer confidence.

SVM-Based Extraction Our second strategy views answer extraction as a classification problem (Figure 2), and employs a binary classifier to classify candidate answers as either *correct* or *incorrect*. Our extractor is based on a support vector machine (SVM), trained on a set of features generated from passages containing correct answers and passages lacking correct answers. The assumption is that given enough positive and negative training examples, the classifier can generalize and subsequently be used to classify new candidate answers accurately. Its output includes a score representing its confidence in the correctness of a candidate answer.

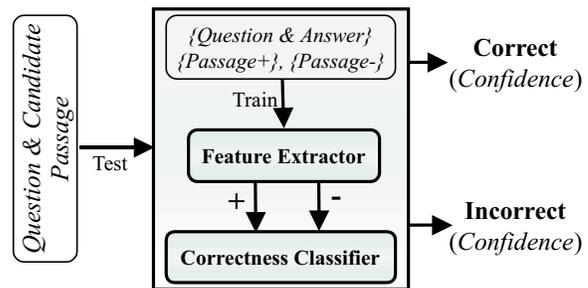


Figure 2: Binary classification approach - A classifier is trained on a set of questions with passages that contain the answer (*Passage+*), as well as passages that don’t contain the answer (*Passage-*). The classifier evaluates answer candidate correctness and assigns them confidence scores.

The feature set used by the SVM extractor includes surface form features such as *n*-grams and paraphrases, passage statistics such as sentence length, the percentage of keywords present in the passage, and metrics expressing the average distance between the candidate answer and the expanded question terms. In addition to a linear distance metric, the SVM uses exponential and quadratic distance metrics since experiments have shown that different answer types and different contexts require different distance functions.

FST-Based Extraction Our final strategy is based on finite state transducers (FSTs), a technique that has been successfully applied to other established information extraction tasks (Hobbs *et al.* 1993). Our FST-based answer extractor covers only the most frequent question types, and is appropriate for question types where most answers can be extracted using a handful of simple, learnable patterns.

Given a document containing named entities matching the expected answer type, the finite state transducer method estimates the likelihood that each named entity is actually an answer. A large number of question type-specific FSTs employing simple surface form patterns, capitalization, sentence and entity boundary flags, and WordNet features (e.g., overlap with synonyms and hypernyms) were generated from training data and then filtered for optimal performance on a validation set. Multiple transducers may match the same answer candidate, each generating an answer confidence. These confidence scores represent the transducer precision over the training data: the number of times an FST matches correct answers relative to the overall number of matches. Figure 3 shows how the FST strategy operates for a simple question type.

Planning

The role of the planner is to find and execute a sequence of actions (steps in the QA process) that transform the initial information state represented by the

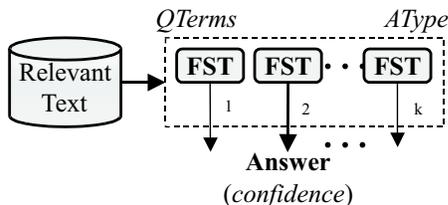


Figure 3: FST-based extraction: A set of FSTs are constructed for each question type. The confidences λ_i are learned using a training set of question-answer pairs with corresponding passages.

question into one containing an answer with maximal information utility. It requires a linguistic analysis of the question, a model of the QA process, and a utility function defining how the planner should assess a state’s information value and resource costs.

Action Models of the QA Process Each major processing stage is represented by one or more probabilistic actions. The actions used in our experiments include a retrieval action, an answer selection action, and three extraction actions, corresponding to the techniques previously described. We assume that question analysis is a preprocessing step that seeds the planning process, and do not include it as a planner action. Thus for this model, the main task of the planner is to decide which extractor to use first, and when to try alternate extraction techniques.

Each action is represented by a set of preconditions and probabilistic effects. Preconditions are conditions required for an action to be applied (e.g. an extractor action depends on the existence of a document set). Effects are discrete sets of changes to the information state that may result from executing the action. Both the probabilities of the effects and action cost estimates are dynamically determined at run-time using features of the current state.

Figure 4 depicts a simple probabilistic action with two possible outcomes, applicable in states where condition c_1 holds. Applying this action to an information state s_o leads to a new information state containing an outcome o^+ (e.g. a state in which we have an answer) with probability $p_{a_i}(s_o)$, or an outcome o^- with probability $1 - p_{a_i}(s_o)$. Each successor state has a utility $U(s)$, defined as a function of its information value and the costs (e.g. time) incurred to reach it.

Predicting Extractor Performance The planner uses two performance estimates for each extractor. The first is a set of prior probabilities for each extractor outcome, estimated from performance on a set of validation questions. The resulting values are stored in a lookup table by answer type, question type, and extractor type.

The second is a set of confidence thresholds used to classify the extraction outcomes. Thresholds are determined for each answer type and extractor, based on the

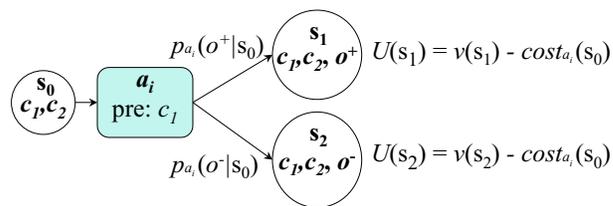


Figure 4: Sample action with two possible outcomes.

correlation between confidence score and accuracy on a validation set. Figure 5 presents a sample receiver operating characteristic (ROC) curve for each extractor on a set of location questions. This plot shows the absolute number of false positives and true positives produced by each extractor for different confidence thresholds between 0 and 1. The closer the curve to the top left corner of the plot, the better the confidence score as an indicator of extractor performance. Although none of the extractors produce very clean confidences, this plot suggests that the best performance would be a combination of the SVM and FST, with the proximity extractor used as a back-off strategy.

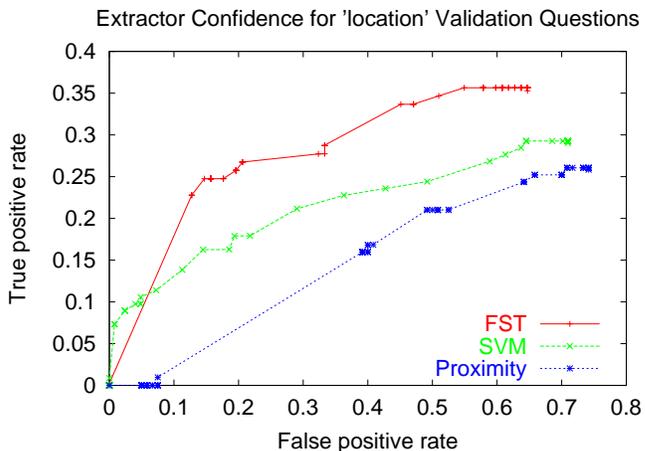


Figure 5: Extractor confidence threshold ROC curve.

Action Selection and Execution At each step, the planner evaluates all actions a_i applicable in current state s_0 (e.g., all extractor actions it can use once a document set is retrieved), selecting the one with highest expected information utility, computed as:

$$\sum_s p_{a_i}(s|s_0)U(s)$$

Upon choosing an action, the planner adds it to the partial plan, projects its internal state representation forward to reflect the action’s possible outcomes, and either continues planning or chooses to execute the next unexecuted action in the plan. When an action is executed (e.g., an extractor is called and returns a set of candidate answers), its results are used to update the

internal information state model on which future planning decisions are based. The process continues until the information goal is satisfied or available resources have been exhausted.

For an extractor, updating the information state model after execution consists of comparing the confidence score of the highest ranked answer candidate to the confidence threshold for that extractor. If it exceeds the threshold, the planner classifies the outcome as successful; if it falls below it, the outcome is considered unsuccessful and the planner must choose amongst the remaining extractors.

Utility Estimation Our extractor experiments make the simplifying assumption that each action has a constant cost, and assume the information value of a state is 1.0 if the top candidate exceeds the confidence threshold, 0.0 otherwise. While these assumptions are restrictive, they are a useful simplification for developing an initial model of extractor performance, one that considers only the extractors’ accuracy, question coverage, and predictive value of their confidence scores.

Experiments and Results

We tested our models for extractor selection on 416 questions from the TREC8-12 QA datasets (Voorhees 2003) that seek a *location* as an answer (e.g. “Where is the Holland Tunnel?”). Our goal was to evaluate the relative performance of each extractor and the performance improvement provided by using the planner to select among the three extractors.

Each question was preprocessed by a question analysis component, with the output corrected to eliminate classification errors, and 50 documents were retrieved for each. A total of 319 (76.7%) had at least one relevant document in the set of retrieved documents. To assess the impact of noise, we also constructed “ideal” sets containing only relevant documents.

The questions were split into training, validation, and test sets of roughly equal size (146, 136, and 134, questions respectively) such that each set contained a similar distribution of question subtypes. All extractor development was performed with the training set, and then tested on both the validation and test sets. All planner strategy optimization was based on the extractors performance on the validation set, and subsequently tested on the test set.

All relevant documents were preprocessed with BBN IndentiFinder (Bikel, Schwartz, & Weischedel 1999) in order to tag named entities. Part of speech tagging was also performed on the retrieved passages (Brill 1992). The support vector machine answer extraction is based on the SVM Light toolkit (Joachims 2002), which was trained using a linear kernel function. The positive to negative example ratio was tuned for best performance: a 1 : 3 ratio achieved the best training performance and was used in subsequent experiments.

The SVM was trained using unigrams, bigrams and trigrams of lexical items (“*is located in*”), part of speech

tags (“*NNS VBG NNP*”), and morphologically normalized words (“*he have be*”). In addition to *n*-grams based on surface forms, the SVM also uses *n*-grams that take into account digit and number classing: the number 143 can be described as three **digits** $\setminus d \setminus d \setminus d$ as well as a **number** $\setminus n$.

For the FST-based extractor, when multiple transducers matched in a passage, only the highest confidence answer was proposed. This selection strategy was based on training data, which suggested that if a high-confidence transducer matches, it is usually correct.

We assessed performance using the *TREC score* metric which measures the overall extractor accuracy: the fraction of questions for which the highest confidence answer is correct. This score represents the most conservative estimate of the extractor performance, as it does not consider any accuracy boost that may be provided by either clustering or candidate filtering during actual answer selection process that follows.

We also used the average *mean-reciprocal-rank* (*MRR*) metric, defined as r^{-1} , where *r* is the rank of the first correct answer occurring within the top 5 answers, 0 otherwise. This represents a more fair assessment of extractor performance, as the role of the extractor is to simply produce a good set of candidates for the answer selection stage and not necessarily to return a single correct answer.

Extraction	Top-50 Docs		Relevant Only	
	TREC	MRR	TREC	MRR
FST	0.307	0.345	0.360	0.400
Proximity	0.205	0.253	0.440	0.468
SVM	0.291	0.359	0.472	0.560
PL-random	0.281		0.463	
PL-FPSc	0.315		0.528	
PL-max	0.409		0.608	

Table 1: Extractor performance on location test questions. The maximum possible TREC score for the Top-50 documents is 0.767, bounded by IR performance.

Our results for the test set are summarized in Table 1. As expected, performance for all extractors is lower when the document set contains noise. However, the FST-based extraction method appears to be slightly less sensitive to the presence of irrelevant documents. The planner model developed from validation data sets the FST confidence threshold to 0.65 with successive back-off to the Proximity and SVM modules using thresholds of 0.0.

This model is somewhat counter-intuitive, considering that the SVM is the best overall single performer. However, it highlights the fact that selection order must account both for accuracy and coverage diversity (Figure 6). Applying the proximity-based extractor prior to the SVM-based extractor preserved additional answer diversity. Lower and upper bounds on planning performance using random choice (PL-random) and perfect selection (PL-max) are provided for comparison.

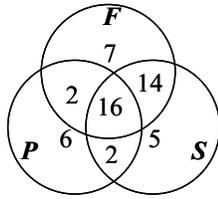


Figure 6: Extractor answer distribution and overlap for the *location* test questions using the Top-50 documents. Collectively, the three extractors provided correct answers to 52 out of 125 answerable questions.

Discussion

In this paper we have described three different answer extraction methods: extraction based on a non-linear proximity metric, binary classification using an SVM to estimate answer correctness, and finite state transducers. These methods differ in the way they extract answers, in the features employed, and also in the quality of their outputs (candidate answers and confidence scores). As our experiments with *location* questions demonstrate, although many questions can be answered by any of the extractors, there are some questions that can be answered exclusively by individual extractors. We have shown that a planner can take advantage of this fact using extractor models based on a few simple run-time information features (i.e., the question type, answer type, and confidence scores) to dynamically select which extractor(s) to call, boosting performance over a fixed strategy using the single best extractor.

We continue to refine our extractor models for all answer types and question types, and are actively exploring more sophisticated models of extractor performance that incorporate additional run-time features of the question, retrieved documents, and extractor output, along with realistic estimates of processing time. We are also investigating techniques for combining results from multiple extractors. Currently, differences in their confidence scores prevent a direct comparison of the candidates, forcing the planner to simply accept or reject the results of an individual extractor when deciding if additional extractors should be used. Merging techniques must also account for possible performance degradation due to an increase in incorrect candidates.

Although the system to date has focused primarily on statistical approaches to extraction, we are currently developing additional extractors that employ *deep* natural language processing techniques and provide interfaces to structured databases, gazetteers, and the Web. Because these new resources are likely to have very different characteristics (e.g., very high precision in a narrow area), we expect these will provide an even clearer demonstration of the planner's benefits. We also anticipate the addition of alternate retrieval and answer selection techniques that would make the planning task itself much more interesting.

References

- Azari, D.; Horvitz, E.; Dumais, S. T.; and Brill, E. 2003. Web-based question answering: A decision making perspective. In *UAI*.
- Bikel, D.; Schwartz, R.; and Weischedel, R. 1999. An algorithm that learns what's in a name. *Machine Learning* 34(1/2/3):211–231.
- Brill, E. 1992. A simple rule-based part of speech tagger. *ANLP*.
- Burger, J., and Henderson, J. 2003. Exploiting diversity for answering questions. In *HLT-NAACL*.
- Chu-Carroll, J.; Czuba, K.; Prager, J.; and Ittycheriah, A. 2003. In question answering, two heads are better than one. *HLT-NAACL*.
- Echihabi, A., and Marcu, D. 2003. A noisy-channel approach to question answering. *ACL*.
- Echihabi, A.; U.Hermjakob; Hovy, E.; Marcu, D.; Melz, E.; and Ravichandran, D. 2003. Multiple-engine question answering in TextMap. *TREC*.
- Harabagiu, S.; Moldovan, D.; Pasca, M.; Mihalcea, R.; Surdeanu, M.; Bunescu, R.; Girju, R.; Rus, V.; and Morarescu, P. 2001. Falcon: Boosting knowledge for answering engines. *TREC*.
- Hobbs, J.; Appelt, D.; Bear, J.; Israel, D.; Kameyama, M.; Stickel, M.; and Tyson, M. 1993. FASTUS: A cascaded finite-state transducer for extracting information from natural-language text. *IJCAI*.
- Ittycheriah, A.; Franz, M.; Zhu, W.; and Ratnaparkhi, A. 2001. Question answering using maximum-entropy components. *NAACL*.
- Joachims, T. 2002. Learning to classify text using support vector machines. *Dissertation, University of Dortmund, Germany*.
- Lita, L. V., and Carbonell, J. 2004. Instance-based question answering: A data-driven approach. *EMNLP*.
- Miller, G. 1995. Wordnet: A lexical database for english. *Communications of the ACM* 38(11):39–41.
- Moldovan, D.; Harabagiu, S.; Girju, R.; Morarescu, P.; Lacatusu, F.; Novischi, A.; Badulescu, A.; and Bolohan., O. 2002. LCC tools for question answering. *TREC*.
- Nyberg, E.; Mitamura, T.; Callan, J.; Carbonell, J.; Frederking, R.; Collins-Thompson, K.; Hiyakumoto, L.; Huang, Y.; Huttenhower, C.; Judy, S.; Ko, J.; Kupsc, A.; Lita, L. V.; Pedro, V.; Svoboda, D.; ; and Durme, B. V. 2003. The JAVELIN question-answering system at TREC 2003: A multi-strategy approach with dynamic planning. *TREC*.
- Ravichandran, D., and Hovy, E. 2002. Learning surface text patterns for a question answering system. *ACL*.
- Voorhees, E. 2003. Overview of the TREC 2003 question answering track. *TREC*.