Learning Strategies for Open-Domain Natural Language Question Answering

Eugene Grois, David C. Wilkins, Dan Roth
Beckman Institute
University of Illinois
405 N. Mathews Avenue
Urbana, IL 61801
e-grois@uiuc.edu, dcw@uiuc.edu, danr@uiuc.edu

Abstract

We present an approach to automatically learning strategies for natural language question answering from examples composed of textual sources, questions, and answers. Our approach formulates QA as a problem of first order inference over a suitably expressive, learned representation. This framework draws on prior work in learning action and problem-solving strategies, as well as relational learning methods. We describe the design of a system implementing this model in the framework of natural language question answering for story comprehension. Finally, we compare our approach to three prior systems, and present experimental results demonstrating the efficacy of our model.

1 Introduction

This paper presents an approach to automatically learning strategies for natural language question answering from examples composed of textual sources, questions, and answers. Our approach is focused on one specific type of text-based question answering known as *story comprehension*. Most TREC-style QA systems are designed to extract an answer from a document contained in a fairly large general collection [Voorhees, 2003]. They tend to follow a generic architecture, such as the one suggested by [Hirschman and Gaizauskas, 2001], that includes components for document pre-processing and analysis, candidate passage selection, answer extraction, and response generation. Story comprehension requires a similar approach, but involves answering questions from a single narrative document. An important challenge in text-based question answering in general is posed by the syntactic and semantic variability of question and answer forms, which makes it difficult to establish a match between the question and answer candidate. This problem is particularly acute in the case of story comprehension due to the rarity of information restatement in the single document.

Several recent systems have specifically addressed the task of story comprehension. The Deep Read reading comprehension system [Hirschman *et al.*, 1999] uses a statistical bag-of-words approach, matching the question with the lexically most similar sentence in the story. Quarc [Riloff and Thelen, 2000] utilizes manually generated rules that selects a sentence deemed to contain the answer based on a combination of syntactic similarity and semantic correspondence (i.e., semantic categories of nouns). The Brown University statistical language processing class project systems [Charniak, *et al.*, 2000] combine the use of manually generated rules with statistical techniques such as bag-of-words and bag-of-verb matching, as well as deeper semantic analysis of nouns. As a rule, these three systems are effective at identifying the sentence containing the correct answer as long as the answer is explicit and contained entirely in that sentence. They find it difficult, however, to deal with semantic alternations of even moderate complexity. They also do not address situations where answers are split across multiple sentences, or those requiring complex inference.

Our framework, called QABLe (Question-Answering Behavior Learner), draws on prior work in learning action and problem-solving strategies [Tadepalli and Natarajan, 1996; Khardon, 1999]. We represent textual sources as sets of features in a sparse domain, and treat the QA task as behavior in a stochastic, partially observable world. QA strategies are learned as sequences of transformation rules capable of deriving certain types of answers from particular text-question combinations. The transformation rules are generated by instantiating primitive domain operators in specific feature contexts. A process of reinforcement learning [Kaebling, *et al.*, 1996] is used to select and promote effective transformation rules. We rely on recent work in attribute-efficient relational learning [Khardon *et al.*, 1999; Cumby and Roth, 2000; Even-Zohar and Roth, 2000] to acquire natural representations of the underlying domain features. These representations are learned in the course of interacting with the domain, and encode the features at the levels of abstraction that are found to be conducive to successful behavior. This selection effect is achieved by a fusion of abstraction space generalization [Sacerdoti, 1974; Knoblock, 1992] and reinforcement learning elements.

The rest of this paper is organized as follows. Section 2 presents the details of the QABLe framework. In section 3 we describe preliminary experimental results which indicate promise for our approach. In section 4 we summarize and draw conclusions.

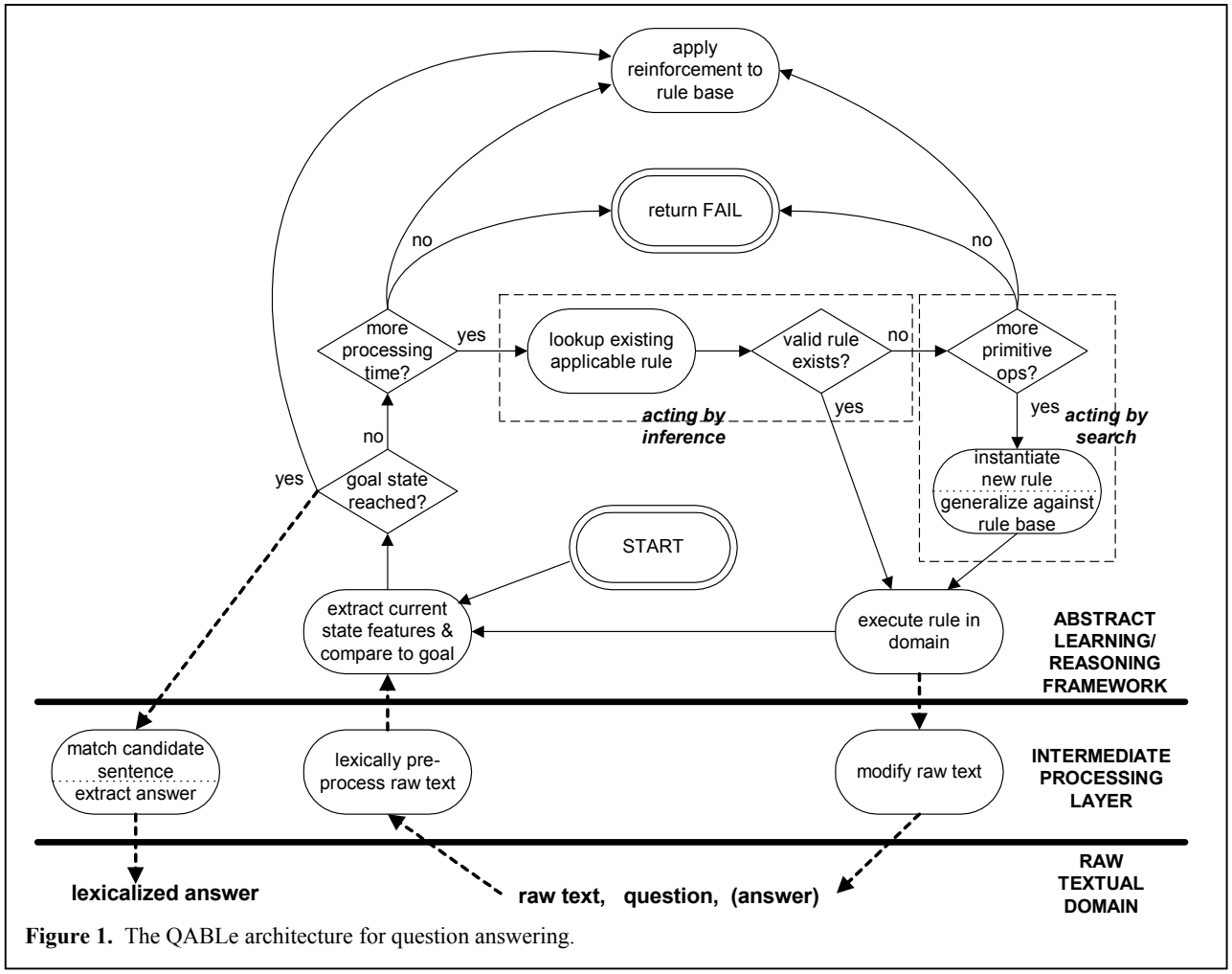

Figure 1. The QABLe architecture for question answering.

2 QABLe – Learning to Answer Questions

2.1 Overview

Figure 1 shows a diagram of the QABLe framework. The bottom-most layer is the natural language textual domain. It represents raw textual sources, questions, and answers. The intermediate layer consists of processing modules that translate between the raw textual domain and the top-most layer, an abstract representation used to reason and learn.

This framework is used both for learning to answer questions and for the actual QA task. While learning, the system is provided with a set of training instances, each consisting of a textual narrative, a question, and a corresponding answer. During the performance phase, only the narrative and question are given.

At the lexical level, an answer to a question is generated by applying a series of *transformation rules* to the text of the narrative. These transformation rules augment the original text with one or more additional sentences, such that one of these explicitly contains the answer, *and* matches the form of the question.

On the abstract level, this is essentially a process of searching for a path through problem space that transforms the world state, as described by the textual source and question, into a world state containing an appropriate answer. This process is made efficient by learning answer-generation strategies. These strategies store procedural knowledge regarding the way in which answers are derived from text, and suggest appropriate transformation rules at each step in the answer-generation process. Strategies (and the procedural knowledge stored therein) are acquired by explaining (or deducing) correct answers from training examples. The framework's ability to answer questions is tested only with respect to the kinds of documents it has seen during training, the kinds of questions it has practiced answering, and its interface to the world (domain sensors and operators).

In the next two sections we discuss lexical pre-processing, and the representation of features and relations over them in the QABLe framework. In section 2.4 we look at the structure of transformation rules and describe how they are instantiated. In section 2.5, we build on this information and describe details of how strategies are learned and utilized to generate answers. In section 2.6 we explain how candidate answers are matched to the question, and extracted.

Phrase Type	Comments
SUBJ	"Who" and nominal "What" questions
VERB	event "What" questions
DIR-OBJ	"Who" and nominal "What" questions
INDIR-OBJ	"Who" and nominal "What" questions
ELAB-SUBJ	descriptive "What" questions (eg. what kind)
ELAB-VERB-TIME	
ELAB-VERB-PLACE	
ELAB-VERB-MANNER	
ELAB-VERB-CAUSE	"Why" question
ELAB-VERB-INTENTION	"Why" as well as "What for" question
ELAB-VERB-OTHER	smooth handling of undefined verb phrase types
ELAB-DIR-OBJ	descriptive "What" questions (eg. what kind)
ELAB-INDIR-OBJ	descriptive "What" questions (eg. what kind)
VERB-COMPL	WHERE/WHEN/HOW questions concerning state or status

Table 1. Phrase types used by QABLe framework.

2.2 Lexical Pre-Processing

Several levels of syntactic and semantic processing are required in order to generate structures that facilitate higher order analysis. We currently use MontyTagger 1.2, an off-the-shelf POS tagger based on [Brill, 1995], for POS tagging. At the next tier, we utilize a Named Entity (NE) tagger for proper nouns a semantic category classifier for nouns and noun phrases, and a co-reference resolver (that is limited to pronominal anaphora). Our taxonomy of semantic categories is derived from the list of unique beginners for WordNet nouns [Fellbaum, 1998]. We also have a parallel stage that identifies phrase types. Table 1 gives a list of phrase types currently in use, together with the categories of questions each phrase type can answer. In the near future, we plan to utilize a link parser to boost phrase-type tagging accuracy. For questions, we have a classifier that identifies the semantic category of information requested by the question. Currently, this taxonomy is identical to that of semantic categories. However, in the future, it may be expanded to accommodate a wider range of queries. A separate module reformulates questions into statement form for later matching with answer-containing phrases.

2.3 Representing the Question-Answering Domain

In this section we explain how features are extracted from raw textual input and tags which are generated by pre-processing modules.

A sentence is represented as a sequence of words $\langle w_1, w_2, \ldots, w_n \rangle$, where $word(w_i, \text{word})$ binds a particular word to its position in the sentence. The k^{th} sentence in a passage is given a unique designation s_k. Several simple functions capture the syntax of the sentence. The sentence *Main* (e.g., main verb) is the controlling element of the sentence, and is recognized by $main(w_m, s_k)$. Parts of speech are recognized by the function *pos*, as in $pos(w_i, \text{NN})$ and $pos(w_i, \text{VBD})$. The relative syntactic ordering of words is captured by the predicate $before(w_i, w_j)$. It can be applied recursively, as $before(w_i, before(w_j, w_k))$ to generate the entire sentence starting with an arbitrary word, usually the sentence *Main*. The integrity of the sentence is checked by the function $inSentence(w_i, s_i) \Rightarrow main(w_m, s_k) \wedge (before(w_i, w_m) \vee before(w_m, w_i))$ for each word w_i in the sentence. A consecutive sequence of words is a *phrase entity* or simply *entity*. It is given the designation e_x and declared by a binding function, such as $entity(e_x, NE)$ for a named entity, and $entity(e_x, NP)$ for a syntactic group of type *noun phrase*. Each phrase entity is identified by its *head*, as $head(w_h, e_x)$, and we say that the phrase head controls the entity. A phrase entity is defined as $head(w_h, e_x) \wedge inPhrase(w_i, e_x) \wedge \ldots \wedge inPhrase(w_j, e_x)$.

We also wish to represent higher-order relations such as functional roles and semantic categories. Functional dependency between pairs of words is encoded as, for example, $subj(w_i, w_j)$ and $aux(w_j, w_k)$. Functional groups are represented just like phrase entities. Each is assigned a designation r_x, declared for example, as $func\_role(r_x, SUBJ)$, and defined in terms of its head and members (which may be individual words or composite entities). Semantic categories are similarly defined over the set of words and syntactic phrase entities – for example, $sem\_cat(c_x, PERSON) \wedge head(w_h, c_x) \wedge pos(w_i, \text{NNP}) \wedge word(w_h, \text{"John"})$.

Semantically, sentences are treated as events defined by their verbs. A multi-sentential passage is represented by tying the member sentences together with relations over their verbs. We declare two such relations – *seq* and *cause*. The *seq* relation between two sentences, $seq(s_i, s_j) \Rightarrow prior(main(s_i), main(s_j))$, is defined as the sequential ordering in time of the corresponding events. The *cause* relation $cause(s_i, s_j) \Rightarrow cdep(main(s_i), main(s_j))$ is defined such that the second event is causally dependent on the first.

Instantiate Rule
Given:
- set of primitive operators
- current state specification
- goal specification

1. select primitive operator to instantiate
2. bind active state variables & goal spec to existentially quantified condition variables
3. execute action in domain
4. update expected effect of new rule according to change in state variable values

Figure 2. Procedure for instantiating transformation rules using primitive operators.

2.4 Primitive Operators and Transformation Rules

The system, in general, starts out with no procedural knowledge of the domain (*i.e.*, no transformation rules). However, it is equipped with 9 primitive operators that define basic actions in the domain. Primitive operators are existentially quantified. They have no activation condition, but only an *existence condition* – the minimal binding condition for the operator to be applicable in a given state. A primitive operator has the form $C^E \rightarrow \hat{A}$, where C^E is the existence condition and $\hat{A}$ is an action implemented in the domain. An example primitive operator is

primitive-op-1 : $\exists\, w_x, w_y \rightarrow$ add-word-after(w_y, w_x)

Other primitive operators delete words or manipulate entire phrases. Note that primitive operators act directly on the syntax of the domain. In particular, they manipulate words and phrases. A primitive operator bound to a state in the domain constitutes a transformation rule. The procedure for instantiating transformation rules using primitive operators is given in Figure 2. The result of this procedure is a universally quantified rule having the form $C \wedge G^R \rightarrow A$. A may represent either the name of an action in the world or an internal predicate. C represents the necessary condition for rule activation in the form of a conjunction over the *relevant* attributes of the world state. G^R represents the expected effect of the action. For example, $x_1 \wedge \bar{x}_2 \wedge g_2 \rightarrow$ turn_on_x2 indicates that when x_1 is on and x_2 is off, this operator is expected to turn x_2 on.

An instantiated rule is assigned a *rank* composed of:

- priority rating
- level of experience with rule
- confidence in current parameter bindings

The first component, priority rating, is an inductively acquired measure of the rule's performance on previous instances. The second component modulates the priority rating with respects to a frequency of use measure. The third component captures any uncertainty inherent in the underlying features serving as parameters to the rule.

Each time a new rule is added to the rule base, an attempt is made to combine it with similar existing rules to produce more general rules having a wider relevance and applicability.

Given a rule $c_a \wedge c_b \wedge g_x^R \wedge g_y^R \rightarrow A_1$ covering a set of example instances E_1 and another rule $c_b \wedge c_c \wedge g_y^R \wedge g_z^R \rightarrow A_2$ covering a set of examples E_2, we add a more general rule $c_b \wedge g_y^R \rightarrow A_3$ to the strategy. The new rule A_3 is consistent with E_1 and E_2. In addition it will bind to any state where the literal c_b is active. Therefore the hypothesis represented by the triggering condition is likely an overgeneralization of the target concept. This means that rule A_3 may bind in some states erroneously. However, since all rules that can bind in a state compete to fire in that state, if there is a better rule, then A_3 will be preempted and will not fire.

2.5 Generating Answers

Returning to Figure 1, we note that at the abstract level the process of answer generation begins with the extraction of features active in the current state. These features represent low-level textual attributes and the relations over them described in section 2.3.

Immediately upon reading the current state, the system checks to see if this is a *goal state*. A goal state is a state who's corresponding textual domain representation contains an explicit answer in the right form to match the questions. In the abstract representation, we say that in this state all of the goal constraints are satisfied.

If the current state is indeed a goal state, no further inference is required. The inference process terminates and the actual answer is identified by the matching technique described in section 2.6 and extracted.

If the current state is not a goal state and more processing time is available, QABLe passes the state to the Inference Engine (IE). This module stores strategies in the form of decision lists of rules. For a given state, each strategy may recommend at most one rule to execute. For each strategy this is the first rule in its decision list to fire. The IE selects the rule among these with the highest relative rank, and recommends it as the next transformation rule to be applied to the current state.

If a valid rule exists it is executed in the domain. This modifies the concrete textual layer. At this point, the pre-processing and feature extraction stages are invoked, a new current state is produced, and the inference cycle begins anew.

If a valid rule cannot be recommend by the IE, QABLe passes the current state to the Search Engine (SE). The SE uses the current state and its set of primitive operators to instantiate a new rule, as described in section 2.4. This rule is then executed in the domain, and another iteration of the process begins.

If no more primitive operators remain to be applied to the current state, the SE cannot instantiate a new rule. At this point, search for the goal state cannot proceed, processing terminates, and QABLe returns failure.

When the system is in the training phase and the SE instantiates a new rule, that rule is generalized against the existing rule base. This procedure attempts to create more general rules that can be applied to unseen example instances.

Once the inference/search process terminates (successfully or not), a reinforcement learning algorithm is applied to the entire rule search-inference tree. Specifically, rules on the solution path receive positive reward, and rules that fired, but are not on the solution path receive negative reinforcement.

2.6 Candidate Answer Matching and Extraction

As discussed in the previous section, when a goal state is generated in the abstract representation, this corresponds to

System	who	what	when	where	why	Overall
Deep Read	48%	38%	37%	39%	21%	36%
Quarc	41%	28%	55%	47%	28%	40%
Brown	57%	32%	32%	50%	22%	41%
QABLe-N/L	48%	35%	52%	43%	28%	41%
QABLe-L	56%	41%	56%	45%	35%	47%
QABLe-L+	59%	43%	56%	46%	36%	48%

Table 2. Comparison of QA accuracy by question type.

System	# rules learned	# rules on solution path	average # rules per correct answer
QABLe-L	3,463	426	3.02
QABLe-L+	16,681	411	2.85

Table 3. Analysis of transformation rule learning and use.

a textual domain representation that contains an explicit answer in the right form to match the questions. Such a candidate answer may be present in the original text, or may be generated by the inference/search process. In either case, the answer-containing sentence must be found, and the actual answer extracted. This is accomplished by the Answer Matching and Extraction procedure.

The first step in this procedure is to reformulate the question into a statement form. This results in a sentence containing an empty slot for the information being queried. For example, "How far is the drive to Chicago?" becomes "The drive to Chicago is _____." Recall further that QABLe's pre-processing stage analyzes text with respect to various syntactic and semantic types. In addition to supporting abstract feature generation, these tags can be used to analyze text on a lexical level. Thus, the question above is marked up as [ELAB-VERB <quantity-distance> (WRB How) (RB far)] [VERB (VBZ is)] [SUBJ <action> (DT the) (NN drive)] [VERB-COMPL (TO to) (NNP <place> Chicago)]. Once reformulated into statement form, this becomes [ELAB-VERB <quantity-distance> _____] [VERB (VBZ is)] [SUBJ <action> (DT the) (NN drive)] [VERB-COMPL (TO to) (NNP <place> Chicago)]. The goal now is to find a sentence who's syntactic and semantic analysis matches that of the reformulated question's as closely as possible. Thus, for example the text may contain the sentence "The drive to Chicago is 2 hours" with the corresponding analysis [SUBJ <action> (DT the) (NN drive)] [VERB-COMPL (TO to) (NNP <place> Chicago)] [VERB (VBZ is)] [VERB-COMPL <quantity-time> (CD 2) (NNS hours)]. Notice that all of the elements of this candidate answer match the corresponding elements of the question, with the exception of the semantic category of the ELAB-VERB phrase. This is likely not the answer we are seeking. The text may contain a second sentence "The drive to Chicago is 130 miles", that analyses as [SUBJ <action> (DT the) (NN drive)] [VERB-COMPL (TO to) (NNP <place> Chicago)] [VERB (VBZ is)] [VERB-COMPL <quantity-distance> (CD 130) (NNS miles)]. In this case, all of the elements match their counterparts in the reformulated question. Thus, the second sentence can be matched as the correct answer with high confidence.

3 Experimental Evaluation

3.1 Experimental Setup

We evaluate our approach to open-domain natural language question answering on the Remedia corpus. This is a collection of 115 children's stories provided by Remedia Publications for reading comprehension. The comprehension of each story is tested by answering five *who*, *what*, *where*, and *why* questions.

The Remedia Corpus was initially used to evaluate the Deep Read reading comprehension system [Hirschman *et al.*, 1999], and later also other systems, including Quarc [Riloff and Thelen, 2000] and the Brown University statistical language processing class project [Charniak, *et al.*, 2000].

The corpus includes two answer keys. The first answer key contains annotations indicating the story sentence that is lexically closest to the answer found in the published answer key (AutSent). The second answer key contains sentences that a human judged to best answer each question (HumSent). Examination of the two keys shows the latter to be more reliable. We trained and tested using the HumSent answers. We also compare our results to the HumSent results of prior systems. In the Remedia corpus, approximately 10% of the questions lack an answer. Following prior work, only questions with annotated answers were considered.

We divided the Remedia corpus into a set of 55 tests used for development, and 60 tests used to evaluate our model, employing the same partition scheme as followed by the prior work mentioned above. With five questions being supplied with each test, this breakdown provided 275 example instances for training, and 300 example instances to test with. However, due to the heavy reliance of our model on learning, many more training examples were necessary. We widened the training set by adding story-question-answer sets obtained from several online sources. With the extended corpus, QABLe was trained on 262 stories with 3-5 questions each, corresponding to 1000 example instances.

3.2 Discussion of Results

Table 2 compares the performance of different versions of QABLe with those reported by the three systems described

above. We wish to discern the particular contribution of transformation rule learning in the QABLe model, as well as the value of expanding the training set. Thus, the QABLe-N/L results indicate the accuracy of answers returned by the QA matching and extraction algorithm described in section 2.6 only. This algorithm is similar to prior answer extraction techniques, and provides a baseline for our experiments. The QABLe-L results include answers returned by the full QABLe framework, including the utilization of learned transformation rules, but trained only on the limited training portion of the Remedia corpus. The QABLe-L+ results are for the version trained on the expanded training set.

As expected, the accuracy of QABLe-N/L is comparable to those of the earlier systems. The Remedia-only training set version, QABLe-L, shows an improvement over both the baseline QABLe, and most of the prior system results. This is due to its expanded ability to deal with semantic alternations in the narrative by finding and learning transformation rules that reformulate the alternations into a lexical form matching that of the question.

The results of QABLe-L+, trained on the expanded training set, are for the most part noticeably better than those of QABLe-L. This is because training on more example instances leads to wider domain coverage through the acquisition of more transformation rules. Table 3 gives a break-down of rule learning and use for the two learning versions of QABLe. The first column is the total number of rules learned by each system version. The second column is the number of rules that ended up being successfully used in generating an answer. The third column gives the average number of rules each system needed to answer an answer (where a correct answer was generated). Note that QABLe-L+ used fewer rules on average to generate more correct answers than QABLe-L. This is because QABLe-L+ had more opportunities to refine its policy controlling rule firing through reinforcement and generalization.

Note that the learning versions of QABLe do significantly better than the QABLe-N/L and all the prior systems on *why*-type questions. This is because many of these questions require an inference step, or the combination of information spanning multiple sentences. QABLe-L and QABLe-L+ are able to successfully learn transformation rules to deal with a subset of these cases.

4 Conclusion

In this paper we present an approach to automatically learn strategies for natural language questions answering from examples composed of textual sources, questions, and corresponding answers. The strategies thus acquired are composed of ranked lists transformation rules that when applied to an initial state consisting of an unseen text and question, can derive the required answer. We compare our approach to three prior systems, and present experimental results demonstrating the efficacy of our model. In particular, we show that our approach is effective given a sufficiently large training corpus, and reasonably accurate pre-processing of raw input data.

References

[Brill, 1995] E. Brill. Transformation-based error driven learning and natural language processing: A case study in part of speech tagging. In *Computational Linguistics*, 21(4):543-565, 1995.

[Charniak, *et al.*, 2000] Charniak, Y. Altun, R. de Salvo Braz, B. Garrett, M. Kosmala, T. Moscovich, L. Pang, C. Pyo, Y. Sun, W. Wy, Z. Yang, S. Zeller, and L. Zorn. Reading comprehension programs in a statistical-language-processing class. *ANLP/NAACL-00*, 2000.

[Cumby and Roth, 2000] C. Cumby and D. Roth. Relational representations that facilitate learning. *KR-00*, pp. 425-434, 2000.

[Even-Zohar and Roth, 2000] Y. Even-Zohar and D. Roth. A classification approach to word prediction. *NAACL-00*, pp. 124-131, 2000.

[Fellbaum, 1998] C. Fellbaum (ed.) *WordNet: An Electronic Lexical Database*. The MIT Press, 1998.

[Hirschman and Gaizauskas, 2001] L. Hirschman and R. Gaizauskas. Natural language question answering: The view from here. *Natural Language Engineering*, 7(4):275-300, 2001.

[Hirschman *et al.*, 1999] L. Hirschman, M. Light, and J. Burger. Deep Read: A reading comprehension system. *ACL-99*, 1999.

[Kaebling, *et al.*, 1996] L. P. Kaebling, M. L. Littman, and A. W. Moore. Reinforcement learning: A survey. *J. Artif. Intel. Research*, 4:237-285, 1996.

[Khardon *et al.*, 1999] R. Khardon, D. Roth, and L. G. Valiant. Relational learning for nlp using linear threshold elements, *IJCAI-99*, 1999.

[Khardon, 1999] R. Khardon. Learning to take action. *Machine Learning* 35(1), 1999.

[Knoblock, 1992] C. Knoblock. Automatic generation of abstraction for planning. *Artificial Intelligence*, 68(2):243-302, 1992.

[Riloff and Thelen, 2000] E. Riloff and M. Thelen. A rule-based question answering system for reading comprehension tests. *ANLP/NAACL-2000*, 2000.

[Sacerdoti, 1974] E. D. Sacerdoti. Planning in a hierarchy of abstraction spaces. *Artificial Intelligence*, 5(2):115-135, 1974.

[Tadepalli and Natarajan, 1996] P. Tadepalli and B. Natarajan. A formal framework for speedup learning from problems and solutions. *J. Artif. Intel. Research*, 4:445-475, 1996.

[Voorhees, 2003] E. M. Voorhees Overview of the TREC 2003 question answering track. *TREC-12*, 2003.