Question-Answering via Enhanced Understanding of Questions

Dan Roth
Chad Cumby, Xin Li, Paul Morie, Ramya Nagarajan,
Nick Rizzolo, Kevin Small, Wen-tau Yih
Department of Computer Science
University of Illinois at Urbana-Champaign

Abstract

We describe a machine learning centered approach to developing an open domain question answering system. The system was developed in the summer of 2002, building upon several existing machine learning based NLP modules developed within a unified framework.

Both queries and data were pre-processed and augmented with pos tagging, shallow parsing information, and some level of semantic categorization (beyond named entities) using a SNoW based machine learning approach. Given these as input, the system proceeds as an incremental constraint satisfaction process. A machine learning based question analysis module extracts structural and semantic constraints on the answer, including a fine classification of the desired answer type. The system continues in several steps to identify candidate passages and then extracts an answer that best satisfies the constraints.

With the available machine learning technologies, the system was developed in six weeks with the goal of identifying some of the key research issues of QA and challenges to it.

1 Introduction

The question answering track in TREC 2002 requires participants to construct a system that can answer open-domain natural language questions automatically given a large collection of news articles with the possible help of external knowledge sources. The questions are all factual and fairly restricted in the syntactic structure and the given answer should be a single phrase, without any additional information. Contrary to previous years, definition questions do not occur in this year's test set.

The open domain question answering (QA) system described in this paper has been implemented as a platform for studying and experimenting with a unified approach to learning, knowledge representation and inference that, we believe, is required to perform knowledge intensive natural language based inferences.

The fundamental assumption that underlies the system described here is that deep analysis of questions plays an important role in the question answering task. An information retrieval(IR) module, an answer selection and verification(AS) module and other supporting components of a QA system all rely on the information extracted from questions which we view here as *constraints* on possible answers. Our approach views the selection of the answer and its justification as an incremental constraint satisfaction process – information extracted from the question constrains the syntactical and semantical structures that can appear in corresponding answers. This process applies restrictions extracted by analyzing the questions using machine learning based classifiers and operates on data annotated also by machine learning based classifiers.

Specifically, the system analyzes the question to extract structural and semantic constraints on the answer and then proceeds in several steps, including a passage retrieval and an answer selection stage, to narrow down the list of candidate answers and rank them based on how well they satisfy the known constraints on the desired answer.

The system is centered around a unified machine learning and inference approach. Classifiers learned according to the SNoW learning architecture [2; 14] are used along with a SNoW based CSCL approach [12] to augment the questions and text documents with additional information including pos tagging information, shallow parsing and some level of semantic categorization (beyond named entities) - information that all the modules of our system exploit, including passage retrieval and answer selection. The same learning architecture is also used to train a question analysis module that provides an accurate and fine-grained semantic classification of the desired answer as well as deeper analysis, including identifying the required relation (if relevant) and some other syntactic and semantic constraints on the answer.

Consider the question [5]:

`What is the fastest car in the world?`

The candidate answers are:

1. `Jaguar`. With the justification: `...the Jaguar XJ220 is the dearest (415,000 pounds), fastest (217mph) and most sought after car in the world.`

2. `Volkswagen.` With the justification: `...will stretch Volkswagen's lead in the world's fastest growing vehicle market. Demand for cars is expected to soar.`

If we only consider a set of search terms — probably `fastest`, `car` and `world` — and assume that one knows that the search is for a proper name, then both sentences equally qualify as justifications. However, a slightly deeper analysis reveals that "fastest" needs to modify "car", while in one of the candidate sentences it modifies "market". In many cases, especially given the current definition of the TREC QA task, ad hoc proximity constraints can do the job. That is, key terms that are close together are more likely than not to be indicative of the correct answer. However, we believe that in order to make big progress in this task, this level of deeper analysis is still necessary.

The constraints may consist of various syntactic and semantic conditions that an answer has to, or is very likely to satisfy. For example, the answer may be known to be a noun phrase, a phrase that describes a location, a date or a city. If the answer is a person's name, an additional constraint may specify an action that this person is taking; It may specify a plurality or a gender constraint, etc.

Information extracted from the question analysis not only constrains possible answers, but also guides the downstream processing in choosing the most suitable techniques to deal with different types of questions. For example, given the question "*Who was the first woman killed in the Vietnam War?*", we want to avoid testing every noun phrase as an answer candidate. At the very least, we would like to be able to tell that the target of this question is a *person*, thereby reducing the space of possible answers significantly by only searching person names.

To achieve this goal, the QA system described here was constructed with an enhanced question analysis module. In the stage of question analysis, a fine-grained and fairly accurate question classification is performed to identify some of 50 predefined classes as the semantic classes of the desired answer. This is done using a machine learning technique that builds on an augmented representation of the question that includes pos tagging, shallow parsing information, and some semantic categorization information. A set of typical semantic relations is extracted from the questions, along with one of their arguments, and we also identify whether the desired answer is the first or the second argument of a binary relation. Eventually, the goal of question analysis is to generate an abstract representation of the question, based on syntactic and semantic analysis.

A passage retrieval module attempts first to apply the information extracted by the question analysis module to determining an ordered set of key terms (along with their properties, e.g., named entities) and construct a search policy to locate a passage that contains the answer. The passage retrieval module is based on the concept of structured text queries which are extended by searching over non-textual concepts such as named entities and part of speech types. Targeted strategies to generate queries for specific question types are formed according to the question analysis results with the help of additional information hinging, of course, on massive document preprocessing and indexing. Several interesting techniques have been developed for that purpose which allow, for example, this module to index different writings of a named entity with a single key.

Once a set of candidate passages are chosen, their location information is passed to the answer selection module to determine the most appropriate answer. The answer selection stage becomes more challenging due to the new requirements enforced in this year's TREC competition: the answers provided by the QA system should be the exact phrase of the answer; and, only a single answer can be output.

Our answer selection is thus an optimization procedure that is constructed, at this point, in an ad hoc way, with an attempt to increase precision. For this purpose, we focused only on answers that can be supported within a *single* sentence and otherwise we just returned "no answer" (NIL). The performance of our system on questions that have no answer is 48% and on questions whose answer is believed to be supported by the system, 24%. This is achieved by several methods. First, the IR module enforces key terms to be within a small window in the returned passages. The AS module itself implements strict constraint-satisfaction criteria in locating answers. In addition, by utilizing the relation information extracted by the question analyzer, it is possible to capture the right answers for specific types of questions and even rely on a knowledge base in some cases. Finally, a procedure for ranking answers is adopted to pick the most suitable answer from the candidates.

This project, with the exception of the IR module, started as a summer project in early June, 2002 and was completed before the end of July. The reliance on mature learning methods allowed us to put together a system for this task in such a short time. Needless to say, there are several important components that are still missing and many of our design goals have not been fully achieved.

Another working assumption is that a robust and accurate question answering system will depend on a large number of predictors. These will be used at many levels of the process and will support a variety of functions, from knowledge acquisition to decision making and integration of information sources. Along with these, there needs to be knowledge representation support that allows, for example, to keep track of predictions as input to higher level decisions and to maintain a coherent representation of a question or a story; and, there also needs to be an ability to make inferences by combining the outcomes of lower level predictions along with some constraints, e.g., those that are implied by the questions. Some of our modules already make use of this view by integrating different levels of classifications and inferences

[12; 15]. However, at the system level, the system developed here only makes some preliminary steps in these directions by putting forward a suggestion for a few of the learning components and a few of the higher level inferences required within a question answering system.

The rest of the paper is organized as follows: Sec. 2 describes the overall system architecture and highlights some of the methods applied. Sec. 3 summarizes the preprocessing of the queries and text. Sec. 4 describes the question analysis module. Sec. 5 and 6 present our information retrieval technique and answer selection and verification module. In Sec. 7, we provide preliminary evaluation on some of the system modules.

2 System Description

The three major modules of our QA system are the question analysis, information retrieval and answer selection modules. Each of them is a combination of multiple submodules and tools. The question analysis module extracts constraints from a question and stores them in a question analysis record. The IR module uses this information to extract relevant passages from the corresponding documents that are indexed at the word, sometimes the phrase and the named entity level. The answer selection module analyzes the candidate passages or searches a small knowledge base to extract the exact answer for the question. We highlight below a few important processing steps in our system.

1. An improved question analysis module provides an accurate question classification (answer semantic classes) and a detailed analysis of the question. The IR and the AS module utilize the analyzed questions and the answer semantic class to select query terms and locate a candidate answer, respectively.

2. A new indexing mechanism based on named entities and pos tags is applied to the document set and a passage retrieval module is employed, that makes use of the concept of structured text queries; it is also capable of searching over non-textual concepts such as named entities and part of speech types.

3. Question-specific strategies are applied both in the question analysis and in answer selection. In the question analysis they enable, for example, recognizing some binary semantic relations along with a missing argument, which is the target of the question. In answer selection, the semantic classes of the answer, as predicted by the question classifier, guide the application of class–specific answer selection rules.

4. A small knowledge base has been acquired and incorporated into the system. It contains a collection of binary relations along with their arguments and can be accessed using the name of the relation and one of the arguments, to extract the second argument. For example, the relation that A is the capital of B will be stored in our knowledge base for all countries and U.S. states. When the answer analyzer identifies that a question is concerned with this relation, the knowledge base will be searched, but after that, the normal AS process will still search for a justification for this answer.

Figure 1 presents the basic structure of our QA system. The following sections will give a detailed introduction of the internal modules.

3 Preprocessing of Questions and Text

All the questions and documents were preprocessed using a part–of–speech tagger, a named entity tagger and a shallow parser which perform a basic–level syntactic and semantic analysis.

The pos tagger is a SNoW based one [4] that makes use of a sequential model of classification to restrict the number of competing classes (pos tags) while maintaining, with high probability, the presence of the true outcome in the candidate set. The same method is used to give pos tags to both known and unknown words. Overall, as shown in [4], it achieves state–of–the–art results on this task and is significantly more efficient than other part-of-speech taggers. Note that we use the pos tagger both for the questions and the documents. Although pos taggers are typically evaluated on declarative sentences, we have evaluated our pos tagger also on questions and found its performance to be satisfactory.

Shallow parsing (text chunking) is the task of identifying phrases, possibly of several types, in natural language sentences. The shallow parser employed here is the SNoW based CSCL parser described in [13; 7]. Primitive classifiers are trained to identify the beginning and the end of each (type of) phrase. The final decision is made using a constraint satisfaction based inference, which takes into account constraints such as "Phrases do not overlap".

In question analysis, we apply this tool to identifying three types of phrases: noun-phrases, verb-phrases and prepositional-phrases. The definitions of these phrases follow those in the text chunking shared task in CoNLL-2000 [6]. When analyzing the documents, in order to save processing time, only noun-phrases and verb-phrases are identified. Below is an example to the type of information provided by this module.

Question: *Who was the first woman killed in the Vietnam War ?*
Chunking: *[NP Who] [VP was] [NP the first woman] [VP killed] [PP in] [NP the Vietnam War] ?*

Both the pos tagger and the shallow parser are available at `http://L2R.cs.uiuc.edu/~cogcomp`.

Our named entity recognizer categorizes noun phrases into one of 34 different categories of varying specificity. The scope of these categories is broader than usual for an average named entity recognizer. With additional categories such as **title**, **profession**, **event**, **holiday**, **festival**, **animal**, **plant**, **sport**, and **medical**, we redefine our task in the direction of semantic categorization. For the above example, the named entity tagger will get:

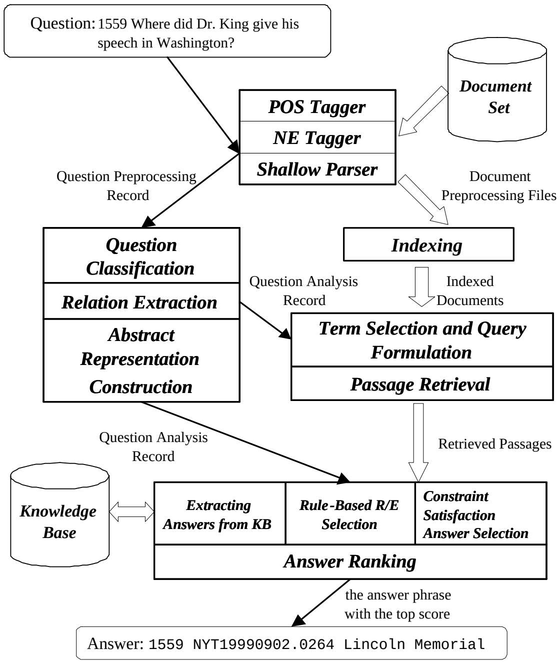
Question: 1559 Where did Dr. King give his speech in Washington?
Document Set
POS Tagger
NE Tagger
Shallow Parser
Question Preprocessing Record
Document Preprocessing Files
Question Classification
Relation Extraction
Abstract Representation Construction
Indexing
Question Analysis Record
Indexed Documents
Term Selection and Query Formulation
Passage Retrieval
Question Analysis Record
Retrieved Passages
Knowledge Base
Extracting Answers from KB
Rule-Based R/E Selection
Constraint Satisfaction Answer Selection
Answer Ranking
the answer phrase with the top score
Answer: 1559 NYT19990902.0264 Lincoln Memorial

Figure 1: System Architecture

NE: *Who was the [Num first] woman killed in the [Event Vietnam War] ?*

Like the shallow parser, the named entity recognition process centers around the SNoW based CSCL [13], with the addition of some predefined lists for some of the semantic categories. One major setback in developing this tool was a lack of sufficient training data. Since our decision process crucially depends on this categorization process, both in question classification and answer selection, we are planning to work on improving the accuracy of this tool.

4 Question Analysis

The goal of the question analysis is two-folded. First, we attempt to recognize the semantic type of the sought after answer, so that we can apply more specific and more accurate strategies when locating the answers. Second, we try to extract informative syntactic and semantic constraints over the possible answers.

To achieve these, question analysis consists of three subtasks. First, a machine learning approach is applied to performing a fine-grained question classification and identifying the semantic classes of the answer [8].

Second, given that the questions in TREC actually concentrate on a limited range of topics, it is possible to define a fairly small number of semantic relations that are the target of a large number of questions. Specifically, for a binary relation, the question identifies one of it's arguments, and looks for the other as the answer. In addition to the answer types, we therefore attempt to extract the target semantic relation in the question. For example, in the question "*When was Davy Crockett born ?*", the goal is to infer that the question is looking for the first argument in the relation *Birthdate_of (?, Davy Crockett)*. The information that the first argument is likely to be a date is also utilized.

Third, we fully parse the question and construct an abstract representation based on this parse result. The representation consists of some general syntactic and semantic relations such as subject–verb, verb–object, modifier–entity pairs, etc.

4.1 Fine Grained Question Classification

The purpose of question classification [8] is to identify the semantic classes of the desired answer. People working in QA seem to agree that this task is an essential and crucial step in the QA process. For example, [10] claims that 36.4% of the total errors of the QA system can be attributed to mistakes at this early stage. Moreover, it seems that the more specific the classification is, the greater the benefit to downstream processes. For example, in the next two questions, knowing that the targets are a "city" or a "country" will be more useful than just knowing that they are locations.

Q: What Canadian city has the largest population?

Q: Which country gave New York the Statue of Liberty?

Therefore, our taxonomy of question classification include 6 coarse classes(ABBREVIATION, ENTITY, DESCRIPTION, HUMAN, LOCATION and NUMERIC VALUE) and 50 fine classes (animal, color, event, food, language, plant, city, mountain, code, individual, title, speed, money, equivalent term, etc.). By using learning with multiple syntactic and semantic features, we can improve the classification accuracy to a level that can be relied upon in downstream processes. The question classifier developed is a two-layered hierarchical learned classifier based on the SNoW architecture. See [8] for details.

One difficulty in the question classification task is that there is no clear boundary between classes. Here are some examples of the internal ambiguity of this task.

Consider

1. What do bats eat?
2. What is the fear of lightning called ?
3. Who defeated the Spanish armada?

The answer of Question 1 could be food, plant or animal; The answer of Question 2 could be an equivalent term or a disease. And Question 3 could ask for a person or a country. Due to this ambiguity, it is hard to categorize these questions into one single class; it is likely that mistakes will propagate into any downstream processes. To avoid this problem, we allow the classifier to assign multiple class labels for a single question. This method is better than only allowing one label because we can apply all the classes in the downstream processing steps without loss of accuracy. In those steps, inaccurate answer candidates will be filtered out by applying further constraints over the answers. To implement this model, we choose to output k ($k \leq 5$) classes for a question. k here is decided by a decision module over the activation of each class. For example, for Question 2, all of *food, animal* and *plant* are returned as the possible answer types.

An important change in this year's TREC competition is that all definition questions are removed from the test set. This change increases the pressure on our question classifier because definition questions are relatively simpler to classify.

4.2 Relation Extraction

According to our statistics, about 30% of TREC 10 questions contained a specific and simple semantic relation which can be described as a binary relation. In TREC 2002, there are also a lot of relational questions. By extracting the semantic relations, the questions can be easily converted into a logical form. For example,

1. *When was Davy Crockett born ?*
   ⟶ Birthdate_of(?, Davy Crockett).
2. *What is the capital city of Algeria ?*
   ⟶ Capital_of(?, Algeria).
3. *Who invented the fishing reel ?*
   ⟶ Inventor_of(?, fishing reel).

The same relation should also be satisfied by the answer. Specifically, in these cases, the question specifies a binary relation along with one of its arguments, and the answer is the other argument. Several systems [1; 16] have previously shown that sometimes, even simple pattern matching methods can achieve high precision in answering those relational questions. While our goal is to develop a more general relation identifier [15], at this point, we define a set of over 30 specific binary semantic relations (such as **Birthdate_of, President_of, Stateflower_of, Inventor_of, Speed_of, Capital_of**, etc.) and apply heuristic rules to determining whether the questions satisfy typical patterns for those relations. For example, typical patterns to extract relations are like:

1. *Who invented/developed A ?* ⟶ Inventor_of(?, A)
2. *What's the capital [city] of A ?* ⟶ Capital_of(?,A)
3. *What's the [flying/...] speed of A ?* ⟶ Speed_of(?, A)
4. *How fast is A ? / does A fly ?* ⟶ Speed_of(?, A)

4.3 Abstract Representation

We represent questions using a simple abstract representation that reflects the basic dependencies between constituents in the question. The representation is extracted from a full parse tree of a question and contains the fields: *Action, Action Subject, Direct Object, Indirect Object, Target Description, Target Modifier, Action Modifier, Location, Time, Extreme Case, and Unit.*

5 Passage Retrieval using Structured Information Queries

In contrast to the more general method of passage extraction from a set of retrieved documents, our system directly retrieves candidate passages from the corpus for analysis by the answer selection module. We search for candidate passages along two primary dimensions: text structure and text classification. Along the structure dimension, our approach uses many of the concepts described within the frameworks of *overlapped lists* [3] and *proximal nodes* [11].

Constraining our search with structure, we are able to specify concept orderings and restrict the size of the text space that must contain the specified concepts. By searching over multiple text classifications, the simplest being the text itself, we are able to make more expressive restrictions over the corpus being searched. For example, in the question **How tall is the John Hancock Building?** searching for a document containing the terms **{john, hancock, building}** is not precisely searching for the desired information, but instead the most available information. When looking for the answer to this question, we only want occurrences of passages that explicitly or implicitly describe the John Hancock Building. A more appropriate description may be "a location near or containing the word **hancock**". This is due to the facts that the John Hancock Building is officially the John Hancock Center, people often simply refer to it as "the Hancock" in many contexts, and the words **{john, building}** are relatively common words, thereby providing limited information. However, we still want to eliminate references to John Hancock the person or John Hancock the company. While there are specific strategies to eliminate each of these problems in general information retrieval approaches, our system captures these ideas naturally and succinctly.

In this section, first we will briefly describe the indexing representation and searching mechanisms. Second, the query language will be provided including some examples of more common usages. Finally, we will describe the heuristics used to dynamically determine the retrieval query and determine when to return the list of feasible candidate passages.

5.1 Indexing and Searching Mechanisms

Indexing is performed via a set of document information files and a set of index term files. Each of these sets is comprised of dictionary files that are fixed size record files for hash lookup purposes and reverse index files which are of variable length per entry. Examples of files are stated below:

- *file.dict* contains a record for each document. Namely the fields of the entry are the document number, the document title, the starting position pointer of the document in the *file.struct* file, and the ending position pointer in the *file.struct* file.
- *file.struct* contains a list of sentence lengths for each document along with the length of the list (the number of sentences in the document). This can naturally be abstracted to other structural properties, but was not done in this case.
- *text.dict, NE.dict, ...* contains a record for each index term comprised of the index term, the starting position pointer in the *text.index* file and the corresponding ending position pointer.
- *text.index, NE.index, ...* contains a list of documents containing the index term that points to a list of *sentence*, *word* tuples representing all of the locations of that specific term.

Searches are performed in document number space and the results are translated into a <document, sentence range> pair for processing by the answer selection module. The fundamental search strategy is as follows. First, all documents containing the required search terms are extracted, much like in a Boolean retrieval model. The allowable range surrounding the first term in this set is calculated for each instance using the information contained in the *file.struct* file. Then each instance of the other terms is checked against the range calculated within the same document to see if it satisfies this constraint. In those cases that it does, the minimum window that contains all constraints is returned as a positive instance.

5.2 Query Language

When forming queries, the basic idea is that the first term serves as the initial set node and we search for a passage containing the subsequent terms within the proximity constraints in each direction. If only one proximity constraint is given, it is assumed to be symmetric. The *union* operator provides a method to induce a disjunction of structural and conceptual constraints. Table 1 shows this language in Backus-Naur form. One additional note is that the absense of a *concept* identifier when describing a term implies *text*, which is only syntactic sugar. This can be seen in the following examples:

```
query → term
      | ( operator query query {query} )
term → concept . keyword
concept → text | NE | POS | ···
operator → [ proximity ] | [ proximity proximity ]
      | [union]
proximity → ( integer , structure )
structure → document | sentence | word | ···
keyword → baseball | holyfield | nobel | ···
integer → ··· | − 1|0|1|2| ···
```

Table 1: Query Language BNF

1. Find the words *george* and *bush* in the same document. (standard boolean retrieval)
   ⟶ ([(0,document)] george bush)
2. Find the word *hancock* as part of a location.
   ⟶ ([(0,word)(1,word)] hancock NE.location)
   Note that this example also accounts for hancock being tagged incorrectly and building (or an equivalent word) being tagged correctly by the named entity recognizer.
3. Find the name *Snoop Dogg*.
   ⟶ ([(1,word)(2,word)] snoop dogg)
   Note that this will match *Snoop Dogg*, *Snoop Doggy Dogg*, and *Dogg, Snoop*.
4. Find passages describing the murder of John F. Kennedy.
   ⟶ ([(0,sentence)] ([(2,word)(3,word)] john kennedy) ([union] murder assasinate kill murdered assasinated killed))

While manual query generation seems somewhat cumbersone, queries are formulated using the information provided during question analysis which is the only interface to the human user. Therefore, this language captures the elements necessary for our constraint satisfaction approach in a succinct and easily usable form.

5.3 Term Selection and Query Formulation

While the query language and retrieval engine provide an expressive and efficient retrieval mechanism, the effectiveness is strictly limited by the queries generated. Our basic query formulation strategy is to iteratively select keywords and refine the query until a threshold quantity of passages is reached or refinement by additional terms would not return any documents that satisfies the required constraints. Since the answer selection mechanism was limited to a single sentence, the proximity constraint was always restricted to the value of *(0, sentence)*. Depending of the expected utility of the next keyword selected, additional operations such as term expansion or union operations are also performed. This process can be viewed as a greedy search over the space of document passages.

The first stage of the passage retrieval algorithm is to determine the strategy to follow based on results from question analysis. For each question analysis classification, an instance of the query formulation agent is instantiated. Each of these instances run independently and their results are accumulated and returned to the answer selector. Simple examples of basing the retrieval strategy on the question analysis module are if a proposed solution is found in the knowledge base or if a quotation is present in the question. In these cases, passages containing these concepts are first retrieved and additional keywords are used primarily to find support for the answer in the corpus.

In the general case, the aforementioned information is not available and we follow the basic search strategy by first extracting the two highest ranking keywords as scored according to Table 2. If two valid keywords cannot be extracted, we return to searching on a single keyword, but this is extremely rare and did not occur on any questions from the TREC contest. These keywords generate our initial search of the form:

([(0,sentence)] text.*keyword1* text.*keyword2*)

Description	Beginning Chunk Score	Default Score
Proper Nouns that are Named Entities	0	1
Nouns that are Named Entities	2	3
Remaining Named Entities except Adjectives or Adverbs	4	5
Remaining Named Entities	6	7
Extreme Adjectives	8	9
Nouns	10	11
Verbs	12	13
Any Remaining Keyword	14	14

Table 2: Ranking Keywords

Once the initial search is constructed, this set of passages is iteratively constrained by selecting the next highest ranking keyword until the termination condition is satisfied or until there are no usable keywords. The termination condition was tuned according to the equation $\frac{(passages_{max}-passages_{min})}{num\_score\_values} last\_score + passages_{min}$, where $last\_score$ is the score of the last keyword selected according to Table 2. In our case, $passages_{max} = 150$ and $passages_{min} = 25$ as determined experimentally. Once the number of candidate passages is less than this value, the results are passed to the answer selection module. The basic idea behind the termination condition is to continue refining the set of candidate passages if higher quality information is still available, but to stop this process once further refinement seems arbitrary. An example of this process follows.

When did Mike Tyson bite Holyfield's ear?
([(0,sentence)] tyson holyfield)
Result: 637 passages
([(0,sentence)] tyson holyfield ear)
Result: 99 passages
([(0,sentence)] tyson holyfield ear ([union] bite bit bites biting bitten burn burns burned prick pricked sting stings stung))
Result: 81 passages

Note that we did not search on the term *Mike* as it is a first name, which generally provides limited utility, and *81* passages seems a reasonable amount. Even though the approach of refining by desired named entity type was abandoned for the contest since it was not fully developed, the earliest experiment was to exploit the fact that we were searching for a date as an answer. Consider the case of the query,

([(0,sentence)] *previous result* NE.B-Date)
Result: 49 passages

which could be viewed as a evidence of a possibly promising approach. However, since the major effort was put into improving the general case, these approaches were not fully developed. Yet, similar strategies were developed to specifically look for words corresponding to abbreviations and abbreviations corresponding to words, but they are not discussed here since they were not used in the actual contest and are a subject of future study.

6 Answer Selection and Verification

Given question analysis records, the answer selection module locate and select the correct answer from extracted passages, taking the following three steps: Sentences in these passages are first analyzed syntactically and semantically by the pos tagger, the shallow parser, and the name entity recognizer. Candidate answers are then located in preprocessed passages. Because only the exact answer is judged as correct this year, we only consider specific types of named entities that the question asks for, or some basic noun phrases, as candidates. Finally, each candidate answer is evaluated and ranked. The top one is output as the final answer along with the document ID.

In the second step, different strategies are adopted to handle two different types of questions. If the question asks for an argument of a semantic relation identified, the answer selector first checks if our knowledge base has the answer. If exists, any occurrence of the answer string in the passages is simply picked. Otherwise, the module searches and identifies the relation in the passages and locate the sentences containing it. For the questions without a relation, the named entities or base noun phrases that satisfy other constraints identified by question analysis are chosen as candidates. Finally, these answers will be scored according to some heuristic rules.

6.1 Deriving Answers from the Knowledge Base

A part of the questions in TREC are asking for some simple facts, which can easily be answered with the help of a dictionary or an almanac. In addition, this type of information is usually available on the web. Examples of this type of question are *What is the capital city of Algeria?* or *Where is Lake Louise?* For the first one, our knowledge base stores the name of the capital city of each country. Once the *capital_of* relation is extracted from the question, we know the correct answer instantly. For the second one, our knowledge base stores the locations of many famous places in advance. Therefore, finding an answer in the document is just a simple string matching process.

Since in the TREC contest, we are required to not only find the answer but also return the document ID as the justification, randomly picking a document that has the answer string is not enough. If more than one document have the answer string, we (Section 6.4) will attempt to pick the document that has the best support.

6.2 Rule-based Relation/Entity Selection

In the rule-based portion of the answer selection module, we seek to apply pattern-based matching criteria to possible answer passages in order to identify relation pairs that correspond to the semantic relations observed in the question analysis. To match a potential candidate answer with a relation obtained from question analysis, we first determine which argument of the relation must be filled in by the candidate. That is, if we have identified a relation *Location_of* in the question and know Y corresponds to "the Statue of Liberty", which is located in X, then X is the argument which our candidate answer ought to match.

Therefore the next step in the matching process is to determine whether the Y argument obtained from question analysis is present in the sentence containing the candidate answer (candidate sentence). For each noun–phrase in the candidate sentence, we test whether it matches the known Y argument exactly or can be resolved to the same concept(meaning). If so we will proceed as outlined below. Otherwise we will test whether any noun-phrase in the candidate sentence matches the first noun-phrase of the Y argument. This partial matching is useful in cases where the argument identified tends to be very long, such as in the question *What is the name of the canopy at a Jewish wedding?* where the known argument of the identified *Name_Of* relation is *canopy at a Jewish wedding.*

Once the presence of Y is known in the candidate sentence, pattern matching rules based on relation types will be applied. We examine the words surrounding the positions of Y and the candidate answer. If they fit any pattern of the specified relation type, we add a predefined score to the total score of that candidate answer. Examples of the patterns for some common relations are listed below:

Location_Of:

1. Y `is (in,at)` ...X
2. X `(has,contains,includes)` ...Y

Capital_Of:

1. X `(is,became,was) ...the capital of` Y
2. `the capital of` Y `...(',',is,contains,includes)` X

For patterns partially matching the candidate sentence, the score awarded to the corresponding candidate answer is reduced compared to the score of full matching.

6.3 Constraint Satisfaction Answer Selection

Since the question analyzer provides detailed analysis of a question, the answer selection module only treats the named entities or base noun phrases that satisfy the constraints as candidate answers.

For example, if a question asks for a person name, then only the phrases that are tagged as *PERSON* will be considered. Note that there can be multiple answer semantic types for a question. Therefore, our refined constraint matchings are actually a mixture of decisions with the help of WordNet [9]. For instance, suppose a question asks for the name of the highest mountain in the US. Those phrases annotated as *LOCATION* are first picked by the named entity tagger as candidates. Then, we use WordNet to filter out those phrases that are not *mountains.*

WordNet can also help reduce the number of candidate answers when additional properties of the answer are given in the question. For questions like *What is a female rabbit called?*, WordNet can check if a string *is-a* rabbit.

Another example of constraint checking is the unit of a numeric answer. For instance, questions asking for *How long is Mississippi river?* cannot have an answer like "100 gallons".

6.4 Ranking Answers

To rank all the candidate answers, we evaluate the confidence with them based on heuristic rules. These rules generally test how closely the abstract representations of the candidate answers match the representation of the question. For instance, a candidate answer will get higher confidence if many of the nearby phrases contain or overlap *Target Modifier*, *Extreme Case*, or other semantic fields in the abstract representation identified from the question.

Note that, ideally, all answer candidates picked in the previous submodules should already be correct answers and ranked high by this submodule. However, since our relation extraction module is not 100 percent accurate and not every constraint derived from question analysis is fully checked, we just hope in this ranking process, the correct answer achieves a higher score than incorrect answers.

7 Evaluation and Error Analysis

Via an enhanced understanding of questions and other new techniques we incorporated with the QA system like the IR engine and new answer selection mechanisms, the total number of correct answers of our system has increased from last year's 54 out of 500(in rank 1) to 109 even with the stricter requirement over the answers(only the exact answer is counted as correct), which is an indication of the effectiveness of the recent work. What's more, the confidence-weighted score of the 500 answers reaches 0.299. Limited by the difficulty of creating reasonable and feasible evaluation standards for some modules of the question answering system, we could only perform evaluation results on part of our QA system. For the question classification and the information retrieval, we obtain evaluation results as follows:

7.1 Question Classification

The learning classifier for questions is built using a training set of 5,500 questions. Because of our specific decision model of allowing multiple labels for a question, in this paper, we count the number of correctly classified questions according to two different precision standards P_1 and $\text{P}_{\leq 5}$. Suppose k_i labels are output for the ith question after the decision model and are ordered decreasingly according to their density values computed by the learning algorithm in the classifier.

We define

$$I_{ij} = \{ \begin{array}{l} 1,\ \textit{if the correct label of the ith} \\ \quad \textit{question is output in rank } j; \\ 0,\ \textit{otherwise.} \end{array} \qquad (1)$$

Then, $P_1 = \sum_{i=1}^{m} I_{i1}/m$ and $P_{\leq 5} = \sum_{i=1}^{m} \sum_{j=1}^{k_i} I_{ij}/m$ where m is the total number of test examples. P_1 is the usual definition of precision which allows only one label for each question, while in $P_{\leq 5}$ we allow multiple labels.

We have a thorough evaluation of the question classifier on the 500 questions of TREC 10.

No.	Train	Test	P_1	$P_{<=5}$
1	1000	500	71.00	83.80
2	2000	500	77.80	88.20
3	3000	500	79.80	90.60
4	4000	500	80.00	91.20
5	5500	500	84.20	95.00

Table 3: Classification precision for fine classes on different training and test sets. Results are evaluated in P_1 and $P_{\leq 5}$.

For TREC 2002 question, our question classification accuracies reach 81% and 88.6% for the 50 fine classes in P_1 and $P_{\leq 5}$ separately. The average number of fine classes output for each question is only 2.15, which shows the decision model is accurate as well as efficient.

7.2 Information Retrieval

Tables 4 and 5 summarizes an evaluation of the passage retrieval module for the TREC 2002 contest questions, disregarding questions which are believed to have no answer contained in the corpus. Let P_j represent the percentage of questions that the passage retrieval module returns at least one passage containing justification for the correct answer. Let N_c be the number of passages returned for questions which generated at least one justified passage and N_i represent the number of passages returned for questions that did not generate any justified passages. These calculations were performed both on the singular sentences returned and on passages comprised of the singular sentence and a "window" of one sentence on each side. Table 4 shows the results when $passages_{max} = 150$ as done with our submission and table 5 shows the results when $passages_{max} = 300$ to show that constraining the number of allowable documents to a smaller threshold increases recall, but also increases the average number of passages returned.

	single sentence	one sentence window
P_j	59.7%	66.3%
N_c	54.2	51.9
N_i	26.0	25.1

Table 4: Passage Retrieval Evaluation: $passages_{max} = 150$

	single sentence	one sentence window
P_j	61.0%	68.7%
N_c	68.4	65.3
N_i	28.2	26.9

Table 5: Passage Retrieval Evaluation: $passsages_{max} = 300$

It should be noted that there were 7 questions that generated more than 500 passages and 8 questions that generated no passages, which were not considered in the above calculations. We also have evidence to support that recall percentages would further improve if the query strategy spanned multiple sentences, but as this approach was not used during TREC, we do not report these statistics here. Finally, it should be noted that the fact that the number of passages returned for questions that generate a justified passage is more than twice that of those that generate no such passage may point to a necessity for a more sophisticated search strategy beyond the greedy algorithm employed thus far.

8 Conclusion

TREC-like question answering requires generating some abstract representation of the question, extracting (for efficiency reasons) a small portion of relevant text and analyzing it to a level that allows matching it with the constraint imposed by the question. This process necessitates, we believe, learning a large number of classifiers, at several levels, that need to interact in various ways and be used as part of a reasoning process to yield the desired answer. Along with these, there needs to be a knowledge representation support that allows, for example, to keep track of predictions as input to higher level decisions and maintain a coherent representation of a question or a story; and, there needs to be an ability to use the outcomes of lower level predictions to make inferences that use several of these predictors along with some constraints, e.g., those that are implied by the questions.

This paper summarizes some preliminary steps we took in this direction by putting forward a suggestion for a few of the learning components and a few of the higher level inferences required within a question answering system. Some of our components work pretty well independently; however, at the system level, the system developed here hasn't achieved very satisfactory results,

This project, with the exception of the IR module, started as a summer project in early June, 2002 and was completed before the end of July. The reliance on mature learning methods allowed us to put together a system for this task in such a short time. Needless to say, there are several important components that are still missing and many of our design goals have not been fully developed. Some of our future research directions include developing our unified approach further in several directions. We plan to incorporate a level of semantic categorization that, hopefully, can impact all modules in the system; we are working on learning better representations for questions, incorporating inference across sentences in the answer selection process and a principled approach to answer selection.

References

[1] E. Brill, J. Lin, M. Banko, S. Dumais, and A. Ng. Data-intensive question answering. In *Proceedings of Text REtrieval Conference (TREC-10)*, pages 393–400, 2001.

[2] A. Carlson, C. Cumby, J. Rosen, and D. Roth. The SNoW learning architecture. Technical Report UIUCDCS-R-99-2101, UIUC Computer Science Department, May 1999.

[3] C. L. Clarke, G. V. Cormack, , and F. J. Burkowski. An algebra for structured text search and a framework for its implementation. *The Computer Journal*, 38(1):43–56, 1995.

[4] Y. Even-Zohar and D. Roth. A sequential model for multi class classification. In *EMNLP-2001, the SIGDAT Conference on Empirical Methods in Natural Language Processing*, pages 10–19, 2001.

[5] Sanda Harabagiu and Dan Moldovan. Open-domain textual question answering. In *Tuturial of the Second Meeting of the North American Chapter of the Association for Computational Linguistics*, 2001.

[6] E. F. T. Kim-Sang and S. Buchholz. Introduction to the CoNLL-2000 shared task: Chunking. In *Proceedings of CoNLL-2000 and LLL-2000*, pages 127–132, 2000.

[7] X. Li and D. Roth. Exploring evidence for shallow parsing. In *Proc. of the Annual Conference on Computational Natural Language Learning*, 2001.

[8] X. Li and D. Roth. Learning question classifiers. In *COLING 2002, The 19th International Conference on Computational Linguistics*, pages 556–562, 2002.

[9] G. Miller, R. Beckwith, C. Fellbaum, D. Gross, and K.J. Miller. Wordnet: An on-line lexical database. *International Journal of Lexicography*, 3(4):235–312, 1990.

[10] D. Moldovan, M. Pasca, S. Harabagiu, and M. Surdeanu. Performance issues and error analysis in an open-domain question answering system. In *Proceedings of the 40th Annual Meeting of ACL*, pages 33–40, 2002.

[11] G. Navarro and R. Baeza-Yates. Proximal nodes: A model to query document databases by content and structure. *ACM Transactions on Information Systems*, 15(4):400–435, 1997.

[12] V. Punyakanok and D. Roth. Shallow parsing by inferencing with classifiers. In *CoNLL*, pages 107–110, Lisbon, Protugal, 2000.

[13] V. Punyakanok and D. Roth. The use of classifiers in sequential inference. In *NIPS-13; The 2000 Conference on Advances in Neural Information Processing Systems*, pages 995–1001. MIT Press, 2001.

[14] D. Roth. Learning to resolve natural language ambiguities: A unified approach. In *Proc. of the American Association of Artificial Intelligence*, pages 806–813, 1998.

[15] D. Roth and W. Yih. Probabilistic reasoning for entity & relation recognition. In *COLING 2002, The 19th International Conference on Computational Linguistics*, pages 835–841, 2002.

[16] M.M. Soubbotin. Patterns of potential answer expressions as clues to the right. In *Proceedings of Text REtrieval Conference (TREC-10)*, pages 293–302, 2001.