Fun with Reading Comprehension

Pranav Anand	Harvard
Eric Breck	MITRE
Brianne Brown	Bryn Mawr
Marc Light	MITRE
Gideon Mann	Johns Hopkins
Ellen Riloff	University of Utah
Mats Rooth	University of Stuttgart
Michael Thelen	University of Utah

October 10, 2000

1 Introduction

The reading comprehension project focused on building a system that can read text passages and answer corresponding questions. Consider the example below.

> **Mars Polar Lander - Where Are You?**
> (January 18, 2000) After more than a month of searching for a signal from NASA's Mars Polar Lander, mission controllers have lost hope of finding it. The Mars Polar Lander was on a mission to Mars to study its atmosphere and search for water, something that could help scientists determine whether life ever existed on Mars. Polar Lander was to have touched down December 3 for a 90-day mission. It was to land near Mars' south pole. The lander was last heard from minutes before beginning its descent. *The last effort to communicate with the three-legged lander ended with frustration at 8 a.m Monday.* "We didn't see anything," said Richard Cook, the spacecraft's project manager at NASA's Jet Propulsion Laboratory. The failed mission to the Red Planet cost the American government more than $200 million dollars. Now, space agency scientists and engineers will try to find out what could have gone wrong. They do not want to make the same mistakes in the next mission.
> (sources: CBC "For Kids" web page, Associated Press, CBC News Online, CBC Radio news, NASA)
>
> 1. When did the mission controllers lose hope of communicating with the lander?
>   Answer: *8AM, Monday Jan. 17, 2000*
> 2. Who is the Polar Lander's project manager?
> 3. Where on Mars was the spacecraft supposed to touch down?
> 4. What did the Mars Global Surveyor do?
> 5. What was the mission of the Mars Polar Lander?

The first question asks for a particular time that is stated in the italicized sentence of the passage. In this case, the answer is stated relatively explicitly in the passage. However, notice that there are some mismatches: *lose hope* in the question verses *last effort ... ended in frustration* in the passage. In addition, this example requires that the system resolve *Monday* to the 17th of January, 2000. Finally, there are a number of expressions denoting points in time in the passage that might distract the system: *January 18th, 2000; December 3rd; minutes before beginning its descent;* and *after more than a month.* This example passage and questions typify the exams we used at the workshop.

Natural language processing (NLP) techniques exist for many of the tasks that arise when taking reading comprehension exams. However, it is unclear

- how to assemble these techniques into a single working system,
- which techniques are necessary,
- which techniques need to perform better, and
- where novel techniques need to be developed.

During the workshop, we hypothesized answers to these questions and worked towards proving or disproving the hypotheses.

Before specifying our hypothesis and discussing our results, let us briefly address the following question: why build systems to take reading comprehension exams? To start with, there are both direct and indirect applications of such systems, e.g., foreign language tutoring systems, first language reading tutoring systems, question difficulty assessment for test construction, and question answering systems to name a few. More generally, they provide a context in which to explore NLP, human reading comprehension, and the combination of the two. Finally, such exams provide a good evaluation of NLP technology: automated evaluation exists, they are theory independent, they are tractable but not trivial, and the provide an impetus for useful/interesting research. Evidence in favor of using reading comprehension exams for evaluation of NLP can be found in (Hirschman et al. ,1999).

Returning to our hypotheses for the workshop, a central hypothesis was that one can fruitfully decompose the reading comprehension task into question analysis (**QAnalysis**) categorizing the question as one of 30 odd types, finding an answer region (**HotSpotting**), and finding the answer phrase in the answer region (**PinPointing**). (This "hypothesis" is really more of a design decision or framework.) Furthermore we hypothesized that we could attack these problems as follows:

QAnalysis: categorize the question based on a shallow parse of the question combined with lexically grounded regular expressions;

HotSpotting: find the answer region (i.e., sentence) using word overlap between question and region;

PinPointing (1): use independent tagger modules to mark phrases with types corresponding to the question types from **QAnalysis**;

PinPointing (2): rank the candidate answers using information from QAnalysis, HotSpotting, and PinPointing(1). (Candidate ranking is necessary since HotSpotting and PinPointing cannot be performed perfectly.)

Note that the primary techniques we used are part-of-speech tagging, shallow parsing, semantic entity tagging, and word overlap. To make things more concrete, let us return to our example:

> ***When* did the mission controllers lose hope of communicating with the lander?**
>
> ... The Polar Lander was to have touched down *December 3rd*$_{\mathbf{i}}$ for a 90-day mission. It was to land near Mars' south pole *at that time* $_{\mathbf{i}}$. The lander was last heard from *minutes before beginning its descent.* The last effort to communicate with the three-legged lander ended with frustration at *8 a.m Monday....*
>
> **Candidates:**
> {[Dec....],[at....]},{[minutes....]},{[8am....]}
> **Ranked Candidates:**
> {[8am....]} {[Dec....],[at....]}, {[minutes....]}

The processing would proceed as follows.

QAnalysis The question is categorized as a temporal question.

HotSpotting Based on the overlap of words such as *communicate* a number of probable answer regions are located.

PinPointing(1) Temporal taggers are used to find a list of candidate answer. In addition, a coreference module is used to chain together coreferential entities as denoted in the example with the subscript i.

PinPointing(2) Finally, a ranking system combines the information from HotSpotting and PinPointing(1) to decide which candidates are most likely.

In addition to the hypothesis concerning the decomposition of the problem, we made the hypothesis that the modules do not need to interact outside of simple input/output. Thus, all of our modules are relatively independent: they make their decisions with little regard for other modules. For example, the tagger that finds person-name phrases does not interact directly with the tagger that finds names of things. The only interaction is that a phrase can be either a name of a thing or a name of a person but not both and, thus, there is a first come first serve interaction. Assuming little interaction is a good starting point for system building. In addition, it makes it easy to experiment with different modules and it results in a simple pipeline architecture. One of the negative ramifications of this loose coupling of modules is that it is difficult to delay hard decisions in hopes that later modules may make the decision easier.

Finally, we hypothesized that a passing system could be constructed using information from a question analyzer, semantic entity taggers, and word overlap. Notice the absence of any information about the relations between entities (either syntactic or semantic), the events they take part in, or the predicates (other than the question types) which hold for them. In other words, we hypothesized that a system with perfect QA, perfect HotSpot, and perfect entity tagging but no information to distinguish candidates in the same hot-spot could pass our exams. This may seem like an obviously incorrect hypothesis but it is important to refute it before expending energy developing relational taggers and incorporating. In addition, an error analysis of such a system should provide clues to which relations need to be tagged. To summarize, the following hypotheses/design-decisions were fundamental:

- The task can be fruitfully decomposition into question analysis, location of answer region (HotSpot), location of answer phrase (PinPointing)
- The modules need only be loosely coupled
- Only shallow question analysis, semantic Entity tagging, and word overlap are necessary

1.1 Materials

Before discussing our results, let us look at the test materials. The text passages were taken from the Canadian Broadcasting Corporation web page for kids (http://cbc4kids.ca/). The CBC publishes five current-event stories a week and has been doing so for over two years. They seem to be aimed at elementary and middle school students (eight to thirteen year olds). The contained 450 words on average and had a Flesch Reading Ease [5] score of 80 (the higher the number, the more people who can read it). For comparison, 91.2 is the score for the Remedia 5W's exams (using in [7]) and 43.9 for AP Newswire.[1] The stories are often based on newswire stories. The bulk of the stories fell into one or more of the following domains: politics, health, education, science, human interest, disaster, sports, business, crime, war, entertainment, environment. The copyright on these stories allows for redistribution for non-profit and research reasons. We added between eight and twelve questions and an answer key to each CBC story. This work was performed by Lisa Ferro and Tim Bevins of The MITRE Corporation. Neither was directly involved in our reading comprehension system building efforts. Lisa Ferro has professional experience writing questions for reading comprehension exams and she led the question writing effort. The questions had varying degrees of difficulty. Below are some examples.

Easy: *Bill Clinton, President of the US, said today...*
Question: *who is the president of the US?*

Moderate: *this virus infects a primate species that is 98 percent related to humans.*
Question: *how closely related are chimpanzees to humans?*

Difficult: *the start of the trip was delayed... because of stomach problems after Clarke had his first taste of the Bedouin delicacy of goat brains*
Question: *why did Clarke become ill before the journey?*

The answer key was created to facilitate the automated scoring of these exams. Alternative answers were indicated where necessary:

- Same answer but different ways of saying it:
  - levels of granularity
    Toronto, Ontario | Toronto | Ontario
  - amounts of information given
    he died | he died in his sleep of natural causes
  - wordings/paraphrases
    Human Immunodeficiency Virus | HIV
- Entirely different answers
  - Where did the boys learn how to survive a winter storm?
    winter camping tips from a friend | their backyard

[1] Lisa Ferro is responsible for these calculations.

Questions were added to 250 stories. We split the data into a training set of 75 stories, a test set of 75 stories, and saved 100 for later use. We intend to release the data to the wider community in the Spring. For comparison reasons, we also looked at a set of short-answer reading exams from Remedia Incorporated which where described in [7]. We also prepared 250 questions of Instructional Fair Incorporated data. Finally, we considered developing reading comprehension versions of the TREC data and short-answer versions of TOEFL exams from ETS.

1.2 Preliminary Results

We continue to work on Spot, the system we started at the workshop. For this reason, the word *preliminary* is included in this section's title. Once Spot has stabilized, there are numerous experiments which we will performed as will be described in the future work section.

1.2.1 Baselines

The table below presents a number of baseline performance numbers that we calculated. The first is simply how often a simple bag-of-words approach could answer CBC questions correctly. Simple bag-of-words meant finding the passage sentence with the greatest number of question words in it using stemming and stopping. How well such a system was able to find a sentence containing the answer is the second number. This is a simple baseline for the HotSpotting task. The final number gives a rough estimate of how well a baseline system based on a QAnalysis, HotSpot, PinPoint design would work. We calculated this number by multiplying

- 87% the performance of Spot's question analysis system,
- 45% bag-of-words HotSpotting baseline
- 56% Confusability (difficulty of the candidate ranking task)

Calling this number a baseline is somewhat of a misnomer since question analysis is quite good and the confusability assumes perfect tagging. The Confusability number is explained in detail below.

Bag-of-Words (answer)	under 10%
Bag-of-Words (HotSpot)	45%
QAnalysis * HotSpot * Confusability	22%

1.2.2 An Upper Bound

In order to prove or disprove our hypothesis that **only shallow question analysis, semantic Entity tagging, and word overlap are necessary**, we did the following. By hand, we

- categorized the questions (QAnalysis),
- marked the sentences that contained an answer (HotSpotting),
- marked the correct and incorrect candidates that matched the QAnalysis type in the HotSpot (part of PinPointing).

Then, for each question, we calculated the expected score for a system that cannot distinguish between candidates in a hotspot, i.e., the number of correct candidates divided by the number of incorrect candidates. Finally, we averaged this number across questions. This average is the expected score for this question for a system that cannot distinguish between candidates in a hotspot but has an oracle for the question type, the hotspots, and the candidates of the matching type in the hotspots. In other words, it is a rough upper bound on a system that uses only **only shallow question analysis, Semantic Entity tagging, and word overlap are necessary** and adheres to our decomposition of the problem into QAnalysis, HotSpotting, and PinPointing. 56% is the number we get. This number can be broken down by question type as is down in the table below. The columns are

1. question type,
2. number of questions of this type,
3. expected SpotCheats score for questions of this type,
4. total number of correct candidates of this type in the hotspots,
5. total number of candidates of this type in the hotspots.

age	2	1.000	2	/	2
agent	23	0.399	37	/	109
ambigbig	4	0.875	5	/	6
ambighow	11	0.332	12	/	51
comparison	1	1.000	1	/	1
defaultnp	28	0.250	34	/	181
defaultvp	15	0.416	19	/	51
definition	7	0.342	7	/	30
duration	4	0.667	4	/	8
event	5	0.523	7	/	17
explanation	41	0.433	61	/	175
length	2	0.750	2	/	3
location	29	0.675	52	/	82
mass	1	1.000	1	/	1
measure	1	1.000	1	/	1
money	8	0.875	8	/	10
organization	3	0.722	4	/	6
person	3	0.667	6	/	9
personname	13	0.833	15	/	21
personnoun	5	0.530	7	/	15
province	2	1.000	2	/	2
quantity	14	0.774	14	/	21
statement	10	0.626	12	/	29
temporal	26	0.750	29	/	44
thingname	1	0.500	1	/	2
title	1	0.500	1	/	2
TOTAL	260	0.559	344	/	879

1.2.3 Current Spot Performance

Spot 5.0, our current system, is scoring 28% on the CBC test set. The table below summarizes the performance on a number of systems on the Remedia test set, the CBC training set, and the CBC Test set. Spot 0.0 was a rudimentary system we put together before the workshop began. Spot 4.0 is the system we had at the end of the workshop and Spot 5.0 is our current system. Spot Cheats is the "system" based on the hand tagging described in the previous section.

	Remedia	**CBC Training**	**CBC Test**
Bag-of-Words	4%	5%	8%
Spot 0.0	24%	21%	21%
Spot 4.0	23%	26%	25%
Spot 5.0			28%
Spot Cheats		56%	

1.2.4 What can we say about about our hypotheses?

With respect to the decomposition of the problem into QAnalysis, HotSpotting, and PinPointing, we can say that we found this framework useful for both conceptual and engineering issues. We did not feel restricted. On the other hand, the hypothesis concerning the loose coupling of modules did seem problematic in that accumulation of error down the pipeline was a problem and it was difficult to make use of some modules' ability to place confidences and/or probabilities on their output. Finally, the hypothesis that shallow question analysis, entity tagging, and word overlap should suffice for passing the CBC exams was refuted: the Spot Cheats failing score of 56% refuted this claim. It is clear that some sort of relational tagging is needed in order to distinguish between entities of the same type.

1.3 Outline of the Rest of the Report

In the next section, we discuss the automated evaluation metrics used to compute many of the scores presented above. Then we will look more closely at the answer type hierarchy and the question analyzer. Next we will describe a number of the taggers that help perform the PinPointing task. Then we discuss the ranking problem. The next two sections describe exploratory work that has not yet been integrated into the current Spot system. First we describe preliminary work on intra-sentential features for discriminating between candidates in the same hotspot. This work makes use of deeper syntactic information than we use elsewhere. Next we look at latent semantic analysis as a way of improving upon our HotSpotting ability. We then look at an alternative architecture which couples modules much more tightly as part of a generative stochastic process. We then conclude. Finally, we offer a short aside contrasting Question Answering a la TREC and Reading Comprehension.

2 Automated Evaluation Metrics

The score a child gets on a reading comprehension test is usually the number of questions the child got right. We would like to use the same criterion for our system, but what does "right" mean? We need a notion of correctness which is intuitively reasonable, consistently annotatable, and automatically calculable.

In previous Reading Comprehension work ([7],[11],[4],[12],[10]), the response returned by a system was an entire sentence, so metrics based on marking up all possible sentence answers were feasible. Now that the system has the potential of constructing new strings (e.g. to resolve a relative date), we need a new evaluation.

Two Notions for Evaluation

During the course of the workshop we decided that the following two notions of correct responses fulfill our desiderata: **Short Correct Response** and **Inferable from Response**. An example should elucidate the two notions. We look at answers to the question, *Who is the Polar Lander's project manager?*, based on the story shown earlier.

For **Short Correct Response**, a desirable response would be *Richard Cook*, which is all and only the text which answers the question. The goal in this notion of correctness is that the system be precise in its response. This approximates a test of the system's "understanding" of the question.

For **Inferable from Response**, a good response would be *"We didn't see anything," said Richard Cook, the spacecraft's project manager at NASA's Jet Propulsion Laboratory.* From just this text, a human could infer the correct answer (Richard Cook). This approximates a test of whether the system's response would answer a human user's question.

Automatable Metrics

We then designed a number of evaluation metrics which are able to predict when a response is a **Short Correct Response** or a **Inferable from Response**. For **Inferable from Response** we use Recall-thresholded (**RecT**): i.e., does the text have enough right words? For **Short Correct Response**, we use Recall-thresholded & Spurious-Thresholded (**RTST**). RTST asks if the answer have enough right words, then goes a step further and asks does it have few enough wrong words? This latter question is answered by **Spur** which is the percentage of response words which are spurious. Without **Spur** a system could get perfect* RecT by returning the document. These metrics are defined formally as follows:

$$SR = \text{set of words in the stemmed, content-word system response}$$

"We didn't see anything," said Richard becomes $\{see, anything, Richard\}$

$$AK = \text{set of words in the stemmed, content-word answer key}$$

Richard Cook becomes $\{Richard, Cook\}$

$$Recall = \frac{|SR \cap AK|}{|AK|}$$

$$Spur = 1 - \frac{|SR \cap AK|}{|SR|}$$

The overlap between the response and answer key is $\{see, anything, Richard\} \cap \{Richard, Cook\} = \{Richard\}$, one word. The recall is one word out of two words in the answer key, or 50%. The spuriousness is two content words out of three in the response, or 66%.

Then the actual metrics we use, **RTST** and **RecT** are defined as follows.

$$RecT = \begin{cases} 1 & Recall \geq \text{threshold} \\ 0 & \text{otherwise} \end{cases}$$

$$RTST = \begin{cases} 1 & Recall \geq \text{threshold} \text{and} Spuriousness \leq \text{threshold} \\ 0 & \text{otherwise} \end{cases}$$

Evaluation of the Evaluation Metrics

Having defined these metrics, we need to evaluate how well they correlate with a human's evaluation according to **Short Correct Response** or **Inferable from Response**.

Gideon and Marc evaluated one of the final test results by hand, using the abstract definitions given above as guidelines. These definitions were vague, so this initial experiment resulted in an interannotator agreement of only 87%. However, their disagreements were systematic, suggesting that a more detailed set of guidelines could greatly enhance the agreement.

The **RecT** metric correlated with the human annotation of **Short Correct** between 73 and 91% of the time, depending on choice of threshold and annotator. **RTST** had an even higher variance of correlation with **Inferable**, varying from 55 to 92% agreement. While in previous tasks the choice of threshold has not been very important, it appears here that the threshold choice is critical to maximize agreement with human judgment. ??Do we agree with this next sentence?? As such, we propose that for future work, human annotators score a number of sentences according to a desired set of guidelines, and then the thresholds for more automatic grading be set so as to best agree with that human annotation.

The **RecT** metric was also evaluated on data from the 1999 Text REtrieval Conference Question Answering track. Using the 37,927 system responses and their human judgments as the test, the metric agreed with humans 93-95% of the time, depending on threshold. It should be noted that in this evaluation spurious word measurement was not as important because of the length limit imposed on system responses.

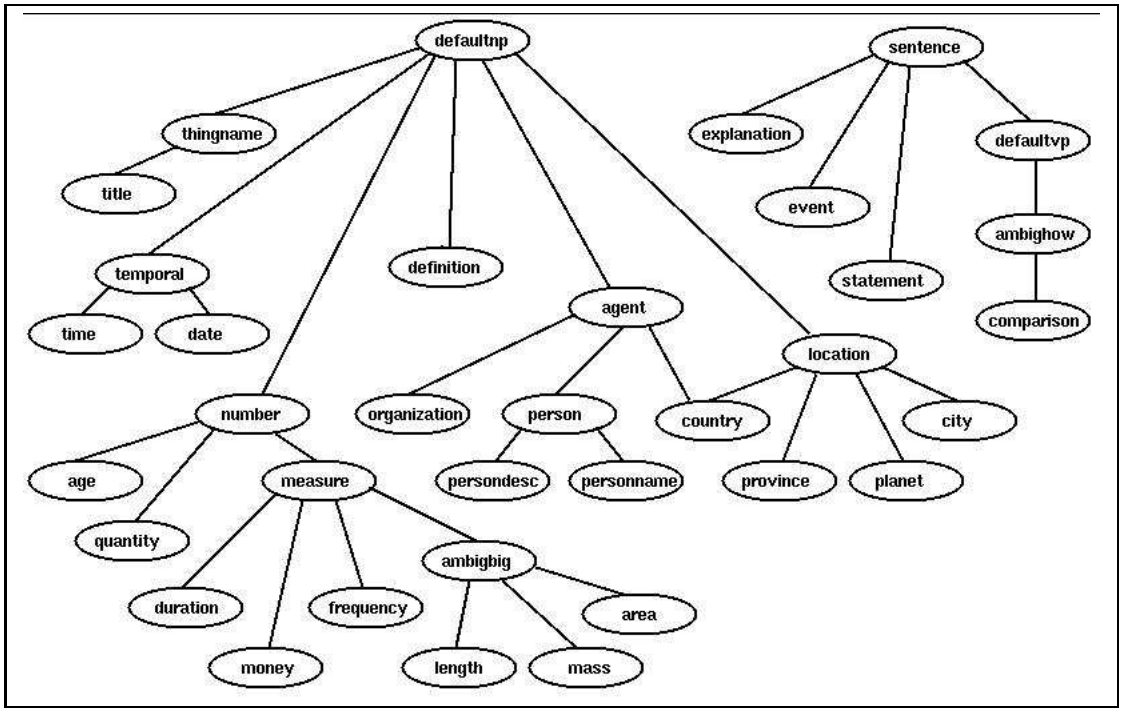


Figure 1: Question Hierarchy

3 Question Analysis

An important step in retrieving the correct answer to a question is to determine what type of answer the question expects. Once this is determined, then Spot can respond with an answer of the same type that the question is asking for.

In order to address this task, one of the first modules in Spot's pipeline is the question analysis module. Its job is to assign each question a type based on the type of answer the question expects. For example, the question "When did the mission controllers lose hope of communicating with the lander?" expects a temporal answer, and "Who is the the Polar Lander's project manager?" expects the name of a person.

The Question Hierarchy

We developed a hierarchy of 34 question types by looking at the questions in the training set. The hierarchy is shown in Figure 1. These question types were designed with three goals in mind. First, the question types were designed to have a high coverage of the data. Second, each question type should be coarse enough to be extracted from a question with high accuracy. Third, the question types should be coarse enough that answers of corresponding types would be able to be extracted from the text. For example, there is no "physicist with wild hair and a moustache" type because it could not readily be extracted from the question, and answers of that type could not readily be extracted from the text.

Question Type	CBC Training(447 questions)	CBC Test(600 questions)
person	8.7% (39)	7.3% (44)
agent	9.6% (43)	9.2% (55)
organization	0.9% (4)	3.0% (18)
location	12.1% (54)	13.5% (81)
temporal	10.3% (46)	11.2% (67)
number/measure	12.5% (56)	10.0% (60)
explanation	15.2% (68)	13.0% (78)
defaultnp	11.9% (53)	11.2% (67)
defaultvp	4.5% (20)	3.7% (22)
other types	14.3% (64)	16.9% (101)

Table 1: Question-type distribution

The question types achieve high coverage on both the training and test data. On the training set, for which the type hierarchy was designed, 100% coverage is attained. Coverage of 99.8% is reached on the test questions. However, these numbers may be somewhat misleading since some question types are more meaningful than others. For example, the "defaultnp" and "defaultvp" types convey little semantic information; rather, they only indicate that the question is looking for a noun phrase or a verb phrase, respectively. If we modify the type hierarchy by removing these two types, then coverage of 83.7% on the training set and 85.1% on the test set is achieved.

Table 1 shows the distribution of question types over the questions in the training and test sets. The frequencies are grouped together by relative location in the type hierarchy; for example, the row labeled "person" contains the total frequencies for the type "person" and the types it subsumes, "personname" and "personnoun". However, the row labeled "agent" only contains frequency counts for the type "agent", since all of its children in the hierarchy ("person" and "organization") have rows of their own.

The Question Analysis Module

The question analysis module uses hand-coded rules to determine the type of each question. Most of these rules look for specific words or phrases such as "who", "where", "when", and "how long". For example, a question that begins with the words "how long" will be assigned the type "duration" without further processing.

Some rules use semantic knowledge as well as lexical matching. For example, if the main question word is "which" or "what", followed by a noun phrase, then the question is probably asking for an answer of the same type as the noun phrase. The question analyzer has a small semantic lexicon of words belonging to the types in the hierarchy, such as "city", "town", and "capital", all of which map to the "city" type. There are also special rules that use semantic information, such as the "how many" rule, which assigns type "quantity" unless "how many" is followed by a time expression such as "how many hours", in which case the question is assigned type "duration".

Another important rule is based on the observation that when a question contains a person's name, it is usually asking for a noun phrase referring to that person, and vice versa. For instance, the

	TRAINING: 91.9%		TEST: 80.5%	
Question Type	**Recall**	**Precision**	**Recall**	**Precision**
person	64.1%	86.2%	40.9%	85.7%
agent	100.0%	100.0%	100.0%	67.9%
organization	97.7%	87.5%	27.8%	100.0%
location	100.0%	96.4%	95.1%	89.5%
temporal	100.0%	100.0%	95.5%	95.5%
number/measure	96.4%	98.1%	85.0%	83.6%
explanation	92.6%	100.0%	89.7%	85.4%
defaultnp	92.5%	86.0%	83.6%	62.9%
defaultvp	100.0%	95.2%	90.9%	83.3%
other types	84.4%	79.4%	66.3%	79.7%

Table 2: Question Analyzer Performance

question "Who is Abraham Lincoln?" is likely to be asking for a phrase containing a person-word, such as "the 16th President of the United States". It is unlikely that a noun phrase with the correct answer will contain the name "Abraham Lincoln". In the same way, the question, "Who was the 16th President of the United States?" contains the person-word "president", and so it is probably asking for the name of a specific person, namely Abraham Lincoln. The question analyzer has access to a lexicon of 5267 person-words, 5271 of which were automatically retrieved from WordNet. A WordNet entry was chosen for inclusion in the person-word lexicon if 60% of its word senses were human word senses.

Maybe include a full list of rules?

Performance of the Question Analysis Module

The question analysis module performs quite well given only a small set of hand-coded rules and a few (mostly small) lexicons. Table 2 shows the module's performance as measured by recall and precision. For each data set (training or test), the overall recall and precision are the same, because the question analysis module returns one and only one type per question. The question analyzer performs at 91.9% on the training set (for which it was designed), and its performance drops to 80.5% on the test set. The most notable decreases in performance are in recall for the person and organization types. The question analyzer relies heavily on lexicons to assign both of these types, which indicates that perhaps the lexicons were tuned too closely to the training set.

4 Semantic Taggers

The purpose of the semantic taggers is to identify text fragments that correspond to possible answer types. Spot's semantic taggers fall into two categories: *entity taggers* and *clausal taggers*. Spot's entity taggers recognize semantic classes that are relatively well-defined and correspond to short answers, such as simple noun phrases and numbers. The entity taggers identify items such as named entites, temporal expressions, and measures. Spot's clausal taggers recognize semantic classes that are characterized by longer answers, such as verb phrases, clauses, or even entire sentences. The clausal taggers recognize items such as explanations, actions, and statements. We will describe the two types of taggers separately since they use different methods.

4.1 Entity Taggers

Spot's entity taggers are responsible for recognizing named entities, temporal expressions, measures, numerical expressions, and some general syntactic constituents. The complete set of entity taggers is shown in the table below.

Entity Tagger	**Tag Set**
location tagger (are there also country/city taggers?)	*CITY, COUNTRY, PLANET, PROVINCE, LOCATION (default)*
number tagger	*AGE, MEASURE, QUANTITY, NUMBER (default)*
organization tagger	*ORGANIZATION*
person tagger	*PERSON_DESCRIPTION, PERSON_NAME*
temporal tagger	*TIME*
title tagger (does this really exist?)	*TITLE*
proper name tagger	*PROPER NAME*
noun phrase tagger	*NOUN PHRASE*
verb phrase tagger	*VERB PHRASE*

Most of these entity taggers were previously developed by MITRE as part of their Qanda question answering system (is this true? if so, cite them). However, the location, organization, and person taggers were created for the Spot system during the CLSP summer workshop. These taggers were built on top of the Sundance natural language processing system, which is a shallow parser built at the University of Utah. Sundance includes a pattern recognizer that allows words to be chunked and tagged based on lexical, syntactic, and semantic properties. The patterns are defined as a simple rule base, and they can use syntactic properties assigned by the parser. Many of the rules also check for membership in a semantic class, which is done via dictionary lookup in Sundance's semantic lexicon. Sundance's dictionaries were created by hand over time, and new entries were added for the reading comprehension task during the workshop. The list of semantic classes and the number of dictionary entries for each class are shown in the table below.

Semantic Class	Dictionary Entries
CITY	495
COUNTRY	467
LOCATION	62
ORGANIZATION	239
PERSON_DESC	827
PERSON_NAME	2041
PLANET	10
PROVINCE	71

The pattern-matching rules used by the location, organization, and person taggers are listed in the following tables. All of the rules label a base noun phrase (e.g., "the man"), or a base noun phrase followed by a prepositional phrase ("the man of La Mancha"). NP_HEAD refers to the head noun of the noun phrase.

Location Rule	Tag	Example
NP_HEAD belongs to semantic class *City*	*CITY*	"Calgary"
NP_HEAD belongs to semantic class *Country*	*COUNTRY*	"Canada"
NP_HEAD belongs to semantic class *Planet*	*PLANET*	"Jupiter"
NP_HEAD belongs to semantic class *Province*	*PROVINCE*	"Alberta"
NP_HEAD belongs to semantic class *Location*	*LOCATION*	"the Atlantic ocean"
NP_HEAD = "Hill" or "Hills"	*LOCATION*	"Bloomington Hills"

Organization Rule	Tag	Example
NP_HEAD = "Party" or "UN" or "government"	*ORGANIZATION*	"the Nova Scotia Liberal Party"
NP_HEAD = "House" or "government" and NP is followed by PP with preposition "of"	*ORGANIZATION*	"the House of Commons"
NP_HEAD belongs to semantic class *Organization* and NP_HEAD is capitalized	*ORGANIZATION*	"the Halifax Regional School Board"
NP_HEAD belongs to semantic class *Organization* and NP_HEAD is capitalized and NP is followed by PP with preposition "of" or "for"	*ORGANIZATION*	"the Assembly of First Nations"

Person Rule	**Tag**	**Example**
NP_HEAD belongs to semantic class *Person Name*	*PERSON_NAME*	"John"
NP_HEAD belongs to semantic class *Nationality*	*PERSON_DESC*	"a Canadian"
NP contains a word in semantic class *Person Desc*	*PERSON_DESC*	"the witty herpetologist"
NP contains a word in semantic class *Person Name* and all words are capitalized	*PERSON_NAME*	"John Fjorski"
NP contains a word in semantic class *Title* and all words are capitalized	*PERSON_NAME*	"Mr. Fjorski"
NP contains a word in semantic class *Person Name* and a word in semantic class *Person Desc*	*PERSON_NAME*	"the witty herpetologist John Fjorski"

Simple Bootstrapping for Person Tagging

Tagging people names is especially difficult for two reasons: (1) the number of person names is virtually unlimited, so out-of-vocabulary words are a major problem, and (2) distinguishing person titles from person names can be tricky because titles are often appended to names (e.g., "Buzz President Art Engleton"). We added a simple bootstrapping mechanism to the person tagger to improve its coverage. The bootstrapping procedure involves three steps:

1. *Run the person tagger* over the text and compile a list of person noun phrases (NPs) identified by the tagger. For example, "Chairman Mao Tse Tung" would be tagged as a person NP because the word "Chairman" is in Sundance's dictionary as a person description word.

2. *Infer person names and descriptions* by partitioning each identified person NP into a string of person description words and a string of person name words. Either (but not both) of these strings can be empty. For example, "Chairman Mao Tse Tung" would be partitioned into a person description string "Chairman" and a person name string "Mao Tse Tung". If the original person NP contained only "Chairman", then "Chairman" would be the person description string and the person name string would be empty. The partitioning heuristics separate descriptions from names by looking for a capitalization change (e.g., "leader Mao Tse Tung") or a known person description word, such as "Chairman".

3. *Perform a second bootstrapping pass* over the text to tag unmarked occurrences of the people names or descriptions inferred during the previous step. For example, a reference to "Mao Tse Tung" would have been left unmarked by the person tagger because none of its words are in Sundance's dictionary. But if the phrase "Chairman Mao Tse Tung" also appeared in the text, then "Mao Tse Tung" could be inferred to be a person name because "Chairman" is a known person descriptor. The bootstrapping pass would then scan over the text again and tag all occurrences of "Mao Tse Tung" as a person. Similarly, a person description can be inferred from a known person name. For example, "poet laureate John Smith" would be

tagged as a person because "John" is in the dictionary. "Poet laureate" could then be inferred to be a person descriptor and the bootstrapping process would scan the text again to tag all occurrences of "poet laureate" as a person.

4.2 Clausal Taggers

Certain types of questions, such as WHY and HOW questions, require answers that are more extensive than those generated by the entity taggers. Many questions require answers that are characterized by verb phrases, clauses, or even entire sentences. We developed a second set of taggers called *clausal taggers* that are designed to extract longer expressions corresponding to explanations, actions, and statements. The table below lists the three clausal taggers used in the Spot system. The explanation tagger identifies possible answers to WHY questions (e.g., "Q: Why is the school opening the club? A: to provide a needed service in Liverpool"). The action tagger identifies possible answers to HOW DO questions (e.g., "Q: How does YNN make a profit? A: by selling advertising space"). The statement tagger identifies possible answers for opinion questions (e.g., "Q: How should politicians respond to racism, according to Bushie? A: encourage racial tolerance".

Clausal Taggers	**QType**	**Examples**
Explanation	Why	The National Arts Centre raised ticket prices because of a lack of funds.
Action	How do	Vettree saved his house by soaking it with water.
Statement	Opinion	He suggested more aboriginal education in schools.

Finding answers to WHY and HOW questions is difficult because the set of possible answers seems relatively unconstrained. (In contrast to WHO questions, for example, which are almost always answered by a noun phrase.) We noticed, however, that many answers to WHY and HOW questions exhibit syntactic similarities. So we explored the idea of using lexico-syntactic patterns to identify possible WHY and HOW answers.

First, it is important to understand how the clausal taggers are used by Spot. The goal of each tagger is to identify text fragments that could be possible answers for a question type, but the taggers themselves do not have access to or care about any specific question. This approach is based on the design of the Spot architecture. For example, the person tagger identifies *all* people mentioned in a text; it is the responsibility of later modules to determine *which* person is the answer to a specific WHO question. Similarly, the explanation tagger identifies every possible explanation that it can find, and it is the responsibility of later modules to determine *which* explanation is the answer to a specific WHY question.

There is one major difference between the entity taggers and the clausal taggers. The entity taggers are designed to be complete. That is, each entity tagger is expected to find *all* instances of the

appropriate type (e.g., all people in a text). In contrast, the clausal taggers are definitely not complete because our understanding of WHY and HOW questions is still lacking. So the clausal taggers identify possible WHY and HOW answers, but later modules treat these taggers differently because they are known to be incomplete.

Most of our work focused on the explanation tagger, which tries to identify possible answers to WHY questions. We discovered 12 lexico-syntactic rules that often characterize explanations. These rules are listed in the table below. The brackets indicate the portion of the sentence that is extracted as the explanation.

Explanation Pattern	**Example**
[**because ***]	Many English-speaking Quebecers did not like Bill 101 because they felt it took away their freedom to put up English signs.
[**because of NP**]	He applied to be an astronaut in the 1960s, but was rejected because of a medical problem.
saying [**NP ***]	Dr. Laurin made no apologies for the law's toughness, saying the law simply reflected Quebec's demographic reality.
[**so ***] if "so" is not in an NP, VP, or followed by "if"	The group broke up last summer so its members could pursue independent careers and interests.
[**NP**] **meant** if NP != "it" or "that"	In those days, marriage meant an end to a young women's career.
means that [*]	That simply means that Canadians will be able to use their health care cards if they move to another part of Canada except Quebec.
means [**the ***]	This means the hair or skin may still contain the DNA needed for the cloning experiment.
[**S**] **(This|That) (could|should|would) mean ...**	The health care agreement is expected to send as much $2.5 billion to the provinces. This could mean that hospitals will have more money to improve care.
[**for GERUND ***]	He was sentenced last week to four months of house arrest and 18 months probation for growing marijuana in his backyard.
[**S**] **That is why ...**	When Moshe Akavak became a jail guard 16 years ago, he was appalled so many Inuit had lost their traditional skills. He believes that's why some became criminals.
[**S**] **So there ...** if "so" begins the sentence	Two out of three aboriginals live in a home with at least one smoker. So there is almost always second hand smoke in the air.
[**InfinitiveVP ***] if "to" is preceded by a base verb or no verb.	She says the school took it on itself to provide a service needed in Liverpool.

The explanation rules generally fall into three categories.

1. **Lexical rules** look for specific words associated with explanations, such as "because", "so", and "means". For example, most clauses beginning with "because" are explanations.

2. **Syntactic rules** look for syntactic structures associated with explanations, such as infinitive VP structures. For example, the question "Why did Mary buy the book?" could be answered with the infinitive phrase "to read on the bus". Not all infinitive verb phrases are explanations. Infinitives used by subcategorization frames are generally *not* explanations. For example, the infinitive in "John wants to buy a car" is functioning as a subcategorization frame for "want", not as an explanation. The InfinitiveVP rule is a heuristic for trying to identify infinitive verb phrases that are not part of a subcategorization frame. Ultimately, however, real subcategorization data should be used to identifying these cases more precisely.

3. **Discourse rules** look for phrases suggesting that the previous sentence was explanatory in nature. For example, the phrase "That is why" indicates that the previous sentence contains an explanation. For these rules, the entire sentence preceding the key phrase is tagged as an explanation.

The tables below list the patterns used to identify actions and statements. Only a single rule is used to identify actions but this rule is quite common. Questions that ask how something is done are often answered by a prepositional phrase that have the preposition "by" and a GERUND as the object of the preposition. The statement patterns include one syntactic pattern ("as" prepositional phrases with a GERUND as the object of the preposition) and several lexical patterns.

Action Pattern	**Example**
[by GERUND *]	YNN makes a profit by selling advertising space.

Statement Pattern	**Example**
[as GERUND *]	Bill 101 is credited by Quebec nationalists as helping preserve the French language in a sea of English in North America.
called on NP to [*]	Bushie called on politicians to encourage racial tolerance.
condemned NP as [*]	But the president of the Alliance Quebec English-rights lobby condemned Laurin's masterwork as an attempt to "wipe out bilingualism" in Canada.
described NP as [*]	Ryan described Rankin as a dedicated family man and master carpenter who was like a younger brother to him.
suggested [*] suggested that [*]	The unsigned flyers suggested that First Nations people love welfare and hate white people because "they have brand new cars nice homes... and new clothes."

The clausal taggers were developed to explore whether it is possible to identify answers to WHY and HOW questions using only lexico-syntactic rules. We believe that the clausal taggers have demonstrated both the feasibility and limitations of this approach. We identified a significant set of lexico-syntactic rules that do seem to reliably capture explanations, actions, and statements. The rules are not perfect or complete, but the fact that we found some good rules shows that

this approach has potential. This result is especially encouraging because the lexico-syntactic rules are domain-independent, which means that they can be used to answer questions about any topic. On the other hand, it is doubtful that all WHY and HOW questions will be amenable to lexico-syntactic rules. Some questions will undoubtedly require more sophisticated discourse analysis and a richer text representation, which we hope to explore in future research.

5 Candidate Ranking

Recall that after each syntactic/semantic tagger annotates the document, the set of tags is partitioned into equivalence classes by co-reference. Each co-reference classes is a candidate answer, and it is the ranker's purpose to select the best candidate. Note that ranking is an unavoidable consequence of Spot's loosely-coupled tagging architecture.

Ranking may be considered as two subtasks: the extraction of a set of discriminating features from the candidates, and the scoring of the candidates according to these features. We explored the following questions:

1. Are overlap between question and candidate and the type of the question sufficient to define a discriminating feature set?
2. Can one train a ranking system?

The first question is discussed in Section 2, which describes our feature set. Section 3 examines the ranking task and outlines the two rankers constructed. Section 4 discusses some finer points of implementation, and provides comparative results of our rankers.

5.1 Examining Features

5.1.1 A Typology

Though the scope of possible features can be daunting, we constructed a typology that is both useful and reasonably robust. Under this typology, all features fall into one of three categories: non-sentential, inter-sentential, and intra-sentential.

Non-sentential features are those that describe constraints on the answer, irrespective of its context (i.e. sentence). Obvious examples are the question types, which encode a preference list over the tag types (or, semantic classes).

Inter-sentential features gauge the *contexts* for likelihood to contain the answer. Examples of these include overlap between a candidate's context and the question, as well as metrics of semantic relatedness (such as latent semantic analysis).

Finally, intra-sentential features are used to select the best candidate within a given sentence. Examples include whether candidate and trace of the question are in similar syntactic configurations and the proximity of the candidate to structurally prominent words.

Under this typology, question (1) may be recast more precisely: How successful is a program that does not consider intra-sentential features? That is, how necessary is local syntactic information in answering reading comprehension questions?

5.1.2 Our Features

Our final feature set is given below, according to the typology.

Non-Sentential Features Our non-sentential (or type) features ask several questions about the relation between the question type (QT) and candidate type (CT), per the Spot type hierarchy outlined in Part II of this report. We decided eventually on five questions, listed below. These features are calculate by comparing the question type (QT) and candidate type (CT).

- typeSameDomain: Are CT and QT under the same depth 1 node?
- typeMatchBinary: Does CT equal QT?
- typeNG: Is the candidate an NG, or default noun group?
- typeSub: Is CT under QT in the hierarchy?
- typePref: Is CT a preferred descendent of the QT (i.e. proper names are preferred to descriptive names when looking for a person)

The features for type-match and type-subsumption should be clear. TypeSameDomain encodes the intuition that there are properly two types of answers, the sentential (explanations, statements, actions, etc.) and the non-sentential (named-entities, mass nouns, etc.). TypePref encodes the *a priori* favoritism of a parent category; for example, proper names are a preferred to descriptive names when looking for a person. Finally, typeNG encodes a universal dispreference for default noun groups (tokens identified by the part-of-speech tagger as nouns that were not tagged by anything else), which ideally ought not be answers to many of our questions, especially those selecting specific semantic classes.

Inter-sentential Features Our inter-sentential features are entirely measures of overlap between the answer context (sentence) with the question sentence. Though none are surprising, some attention should be paid to overlapBigram_noNNP. We found that bigram counts, although indicative of sentences containing the terms of the question, could be artificially inflated by proper noun phrases. For example, if the question were to cite "John Morris Rankin," bigram overlap would be higher for a sentence containing the entire name than it would for one containing only "Rankin."

- overlap: Stemmed word overlap between CN and QS.
- overlapN: Stemmed noun overlap between CN and QS.
- overlapV: Stemmed verb overlap between CN and QS.
- overlapBigram: Bigram overlap between CN and QS.
- overlapBigram_noNNP: Bigram overlap between CN and QS, after removing proper nouns.
- overlapNNP: Proper noun overlap between CN and QS.

In addition to the above low-level features, we also computed if the answer context contained certain syntactically important words from the question. These are given below.

- overlapQV: Is the matrix verb of the question in the CN?
- overlapQH: Is the "head" of the question in the CN? The head is basically the most prominent NP.
- overlapWHEAD: Is the head of a "What" or "Which" wh-phrase in the CN?
- overlapQO: Is X in the question's "of X" PP in the CN?
- overlapQT: Is the question's THEME in the CN?

Intra-sentential Features As of press, we have only one intra-sentential feature, overlapM. This feature measures the overlap between the answer itself and the question. OverlapM encodes the intuition that for most comprehension questions the answer is not contained in the question. For example, if the system is queried with, "Which instruments did John Rankin play in addition to the violin?", an answer extract of "violin" from the sentence "Rankin played the piano, flute, and violin" would be most undesirable. To reiterate:

- overlapM: Stemmed word overlap between the answer and QS.

5.1.3 The Difficulty of the Ranking Task

Recall that the consideration of question (1), under reformulation, is the discriminatory power of non- and inter-sentential features. To address this, we computed the average number of competitors of the proper type in the best sentence in the document. That is, suppose the system could only narrow the field to the best sentence. Then if it had to pick an answer (by chance), how good would it do?

We undertook this first using Spot 4.0 to identify both the best sentence and the tags within that sentence (the competitors). The experiment was repeated, once with oracular choice of best sentence and Spot 4.0 sematic tagging, and once more with oracular choice of sentence and oracular tagging.[2] The results are in the following table.

The third row is most illuminating. It argues that even with perfect extraction and scoring of information up to the sentential level, there are still almost two candidates to choose from. One cannot shirk from considering intra-sentential information.

[2]In addition, the first two experiments relied on the Spot question analyzer, while the final experiment used an oracle.

Table 3: Average Number of Competitors in Best Sentence

Modules	No. competitors
Spot 4.0	9.1
Oracular sentence locator	4.3
Oracular sentence locator, taggers, and question analyzer	1.92

5.2 Examining Scoring

5.2.1 The Rankers

One goal of the Spot Team was to attempt to train a ranker on the features listed in the previous section. We considered two ranking strategies: a lexicographic scoring system, and a log-linear model. The first, which specifies a fixed order (that may vary by question type), has as its chief advantages its simplicity, transparency, and lack of training. However, in the case of Spot, where modules' output is not necessarily trustworthy, the algorithm's draconian ranking places unrealistic burdens of trust on certain features.

Hence, we also decided to construct a log-linear model, based on code developed by Warren Greiff, Eric Breck, Jason Rennie, and Misha Voloshin at the MITRE Corporation. We constructed the training data by passing our Test set through Spot, and training the model on the resulting feature vectors for each *tag* (not candidate), with the tag's RecT score serving as dependent variable. Due to time limits, we were unable to train the system on RTST, our eventual metric for the reading comprehension task.

5.2.2 Lessons

Though our log-linear model did not train on the RTST, there were several points learned. In fact, the unsuitability of the RecT metric for reading comprehension was empirically demonstrated by the very training with respect to it – the system learned to prefer sentences to any other answer type.

Further, we also learned that the importance of features varies drastically across types, and not accounting for this can change behavior in very unexpected ways. For example, the weight for the typeSameDomain feature was negative and non-trivial, in part due to training on RecT[3], and in part because several conceptually non-sentential question types (default NPs, for example) may indeed be answered by a sentential answer, and vice versa (consider nominalizations for explanations).

The upshot of this is that while it is important to create universal features, many features one might suspect to be universally good vary in performance across type of question. One possibility we are pursuing is to train on data "split" across question types.

[3]Under RecT, sentences are extremely good answers to all question types, even non-sentential ones.

5.3 Gory Details

This section details how the lexicographic and LLM rankers operate.

5.3.1 Lexicographic Ranking

Our eventual ranker of this type employs heuristics to account for variation of feature order for sentential candidates. The default ordering of features is the following: typeMatchBinary, typeSameDomain, typePref, typeSub, overlapM, overlapBigram_noNNP, overlapQV, overlapQH, overlapNNP, overlapQO, overlapQT, overlapWHEAD, overlapBigram, overlap, overlapV, overlapN.

For sentential answers, we use the additional "hotspot" feature (this was not discussed above because it pertains as of yet only to the heuristic scorer), which indicates if a sentence is, or is adjacent to, the document sentence having highest overlap with the question. If a candidate is not in the hotspot, it is removed from consideration. Thus, the system first examines the sentential answers of the correct type. If none lie in the hotspot, it simply returns the best sentence (which happens to be the hotspot).

5.3.2 Log-Linear Ranking

The sole consideration of the log-linear model not addressed above is the encoding of the dependent variable. After experimenting with data from TREC 1999 (using human judgments as dependent variable), we decided to construct a binary event vector, where each element is ANDed with the dependent variable. This is hardly an obvious decision, as it may be possible that certain features work best with OR, or perhaps XOR. However, conjunction produced the best performance.

To convert from integral features (such as overlap) to binary features, we expanded a feature into a histogram, determined by whether the feature's value was greater than or equal to a given bin size. For example, suppose we divided overlap up into 3 bins: 2, 5, and 8. If the feature value was 2 or greater, the first feature would be 1; if it were 5 or greater, the same. Hence a value of 7 would set the first two features on.

5.4 Conclusion

We close by returning to the questions in the Introduction:

1. Are overlap between question and candidate and the type of the question sufficient to define a discriminating feature set?
2. Can we train a ranking system?

The answer to question (1) we have shown to be no. To obtain a "passing grade" on a reading comprehension exam, more intra-sentential information must be added.

Although we did not complete our log-linear model, its future looks bright. Already it has provided two interesting results: that universal low-level feature sets are not extremely practical, and that RecT is not the correct metric for the reading comprehension task.

Finally, we have developed strong motivation for two areas of future research:

1. the incorporation of intra-sentential features
2. the construction of ranking systems that take into account question type.

6 Syntax

It has been suggested that the reading comprehension task may lend itself particularly to deep syntactic parsing. While the strategy undertaken by the Spot Team did not include the use of deep parsing, the workshop seemed appropriate for an initial look at its utility.

One motivation for syntactic mark-up is the observation that the governing verb of the question trace often appears in the document, modulo synonymy issues, as the governor of the answer. This is not surprising; we are simply using structure as a proxy for local semantic relations. Figure 2 provides a prototypical example.

Q What has been built along the river to hold back high water levels?

DOC A federal-provincial Fraser [River] flood-control [program] has rebuilt about 250 kilometres of [dikes] to withhold water [levels] as high as those in the 1894 [flood].

Brackets represent correct type for question (default NP).

Figure 2: Example of governor similarity between question trace and answer in document.

In the question shown in (Fig. 2), [what] is the object of [built]. In the document, [dikes] is the object of [rebuilt], the concept of which entails *built*. The lexico-syntactic similarity of the structures around [dikes] and the trace renders [dikes] a better candidate.

Encoding this intuitive observation required the following steps:

1. Parse question to determine position of trace.
2. Calculate governor vector of trace.
3. Parse document.
4. Calculate governor vector of each terminal in document.

It might seem that steps (2) and (4) are trivial read-outs from the parse. However, a complete parse is a relatively macroscopic view of a sentence, and it is not true that the best macroscopic view necessarily entails the best local view. Hence, instead of considering the most probable parse tree, the algorithm examines the entire forest to arrive at a vector of governors (with probabilities) for a given word. This procedure will be covered in more detail in the following section.

Construction of the document and question grammars will be taken up in Section 6.2. We note for the present that the document grammar was taken from an extant English PCFG, and the question grammar was adapted from this by the common introduction of slash categories.

6.1 Governors Algorithm

The Governors Algorithm annotates each word with a vector of governors and estimated frequencies. As an example, consider again the document sentence from (Fig. 1): A federal-provincial Fraser River flood-control program has rebuilt about 250 kilometres of dikes to withhold water levels as high as those in the 1894 flood.

In the correct structure of the sentence, [dikes] is the object of [rebuilt]. In another tree structure, it is the object of a verb [program]. Hence, the governor vector for [dikes] is (ordered by estimated frequency):

- rebuild obj 0.6
- program obj 0.1
- of pobj 0.1
- ...

The governor vector is a representation based upon all possible parses in the forest, since, as mentioned above, we are inquiring about local behavior, which is not necessarily determined by global behavior. To search through the parses, we considered the forest as a headed product-sum circuit, where each non-terminal was either a sum gate, representing the union of its child inputs, or a product gate, representing the product of inputs. This is illustrated in (Fig. 2).

Starting from the root node, the algorithm is applied to the forest resulting from the inside-outside algorithm, and each gate is labelled with a governors vector (recall that this circuit is *headed*).

The frequency calculation of the gates is as follows. Let D_V be the parse forest. For $v \in D_V$ let $in(v)$ be the tree under v and let $h(u)$ be the expectation that u is the head of a tree.

Then for each v in D_V, consider all possible types of non-terminal child nodes: sum gates, heads of product gates, and non-heads of product gates. If v is a product gate, and u a child of v the head of $in(v)$, then $h(u) = h(u) + h(v)$, because u is the only head of $in(v)$.

If v is a sum gate, there is no head of $in(v)$. Thus, for u a child of v, expectation contribution $h(v)$ makes to $h(u)$ is proportional to the importance of u in v. That is, $h(u) = h(u) + \frac{i(u)}{i(v)}h(v)$.

The final case to consider is when v is a product gate and u under v is a non-head. Then we let $h(u)$ represent the governors vector for u. This is summarized in (Fig. 3).

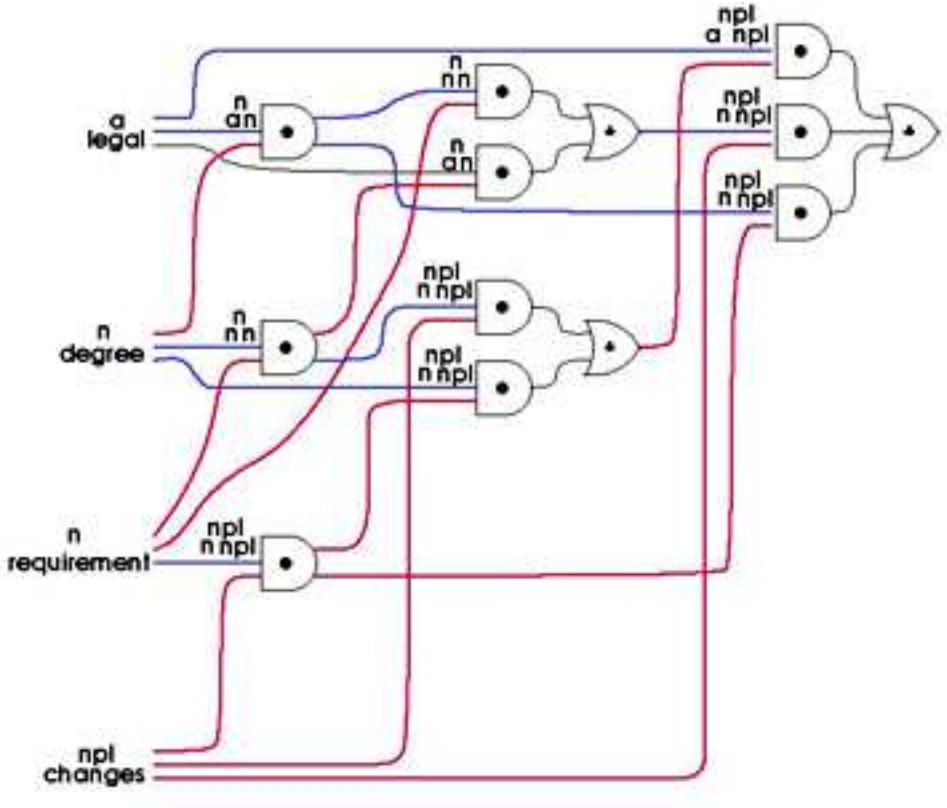


Figure 3: The Headed Product-Sum Circuit

for each v in D_V, from root down: for $u \in in(v)$,

$$h(u) = \begin{cases} h(u) + \frac{i(u)}{i(v)} h(v) & v \text{ a sum gate} \\ h(u) + h(v) & v \text{ a product gate and head}(u, v) \\ h(u) + (w(v), c(v), c(u) \mapsto f(v)) & \text{otherwise.} \end{cases}$$

Figure 4: Formalization of governors algorithm

6.2 The Grammars

We took as our base grammar an existing lexicalized PCFG of English, with 793 hand-built linguistic rules and 928 phrase bigram robustness rules. The language model was estimated on roughly 60 million words by the inside-outside algorithm.

The advantage of this large grammar is its very latitude. However, because training had been on declarative structures, it could not effectively parse questions. Due to constraints of both time and data, it would be infeasible to construct a separate question grammar of approximate robustness to the original grammar.

This problem was remedied by constructing a GPCFG (Generalized Phrase Context Free Grammar) of the original grammar by introduction of slash categories, which indicate that a child element is missing (because of *wh*-movement in this case).

The introduction of slash categories allows us to bridge the grammar peculiar to questions and the robust grammar developed on declaratives. In any question tree, categories from the old grammar dominate only categories from the old grammar. This allows the grammar to be re-estimated on the question corpus, and merged with old PCFG parameters. In addition, 183 context-free question rules were added to the rule set. An example rule is given in (Fig. 4), and (Fig. 5) presents an example question parse with the slash categories.

Qvslash whNP VDF NC1 VBASE/w' X

Figure 5: Example Question Rule.

Nonetheless, we found the training corpus (651 questions) insufficient. In particular, there were insufficient parameter tying features in formalism to allow lexicalized question rules to share parameters with old grammar.

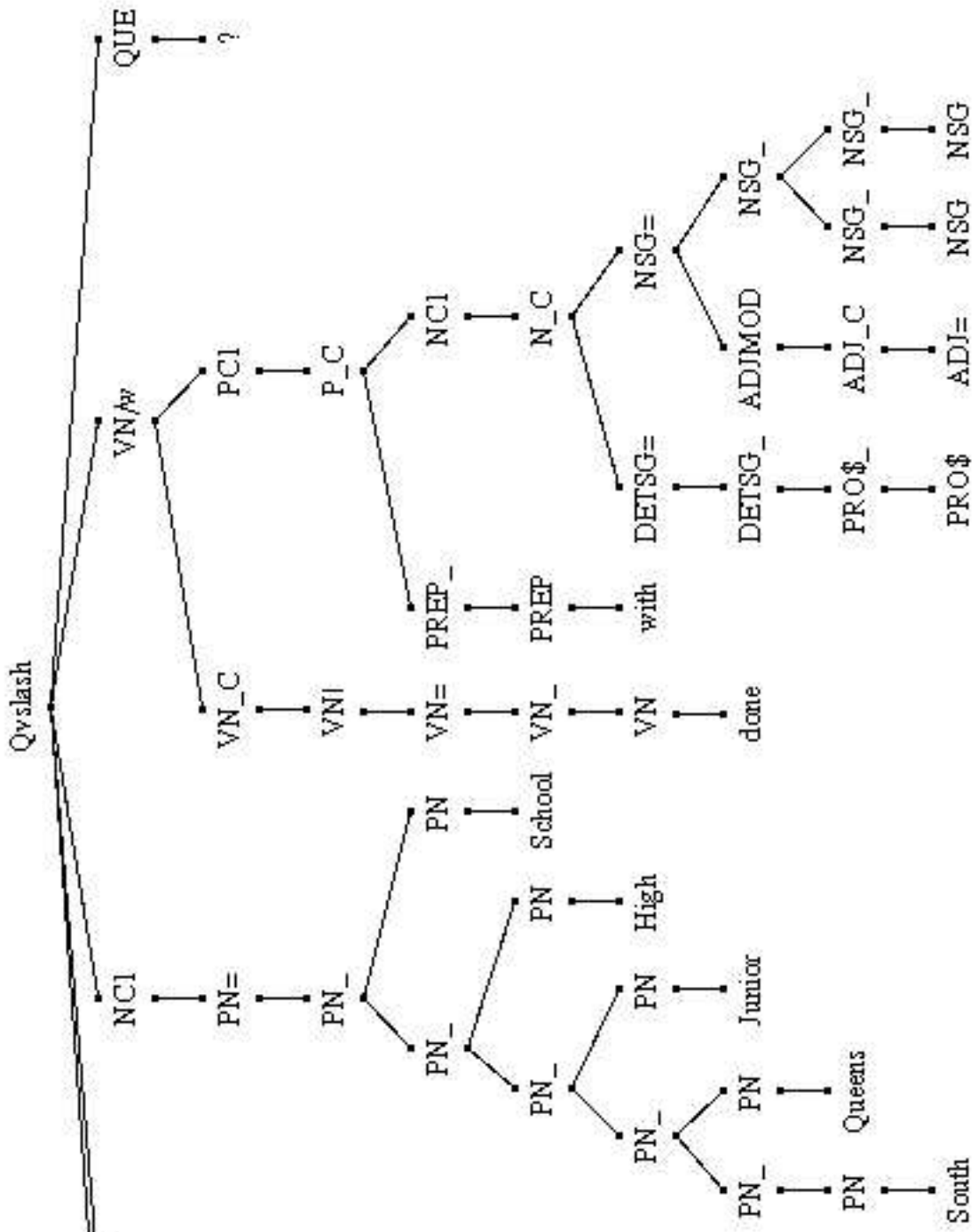


Figure 6: Example question parse.

7 Latent Semantic Analysis for Better HotSpotting

7.1 Introduction

In the reading comprehension task, one way to decompose the problem is to look first for the sentence likely to contain the answer (the answer sentence). We examined whether LSA (Latent Semantic Analysis) would help in identifying the answer sentence. We tested the performance of this method using a pre-tagged set of questions and answer sentences.

LSA (Latent Semantic Analysis) is a method pioneered by Tom Landauer for determining semantic similarity between texts. To use LSA, one generates a term by document co-occurrence matrix and then uses SVD (Singular Value Decomposition) to reduce the dimension of that matrix down to a small number (100-300). Similarity between two terms in this matrix is determined by the cosine of their vectors of values in the semantic spaces. Similarity between two texts is defined to be the cosine between the text centroids. The centroids are computed by word. Exact matches will have a cosine of 1, and unrelated texts will have cosines of around 0.01.

7.2 Implementation

To construct the SVD for a **corpus**, we go through a number of steps :

- Prepare the documents: chop the corpus into documents and put a sentence on each line
- Create the co-occurence matrix: sample the chopped files, generate word usage statistics, and create a sparse matrix for matlab input
- Reweight the co-occurrence matrix: create batch files to with Matlab
- Create SVD: use Matlab to do svd de-composition of co-occurrence matrix

7.3 Word Similarity

After building the machinary to construct the LSA matricies, we tested the machinary initially by looking at simple word to word similarity results.

We started with two days of LA Times news wire (ap/ap89?220). In these two files there were 584 documents, 20078 term types, and 218,750 tokens. We created a co-occurrence matrix and reweighted it by taking log (C(t,d) / Sum C(t,d)) / H(d|t). C(t,d) = count of term and document pair, H(d|t) = entropy of document given term. This reweighting is standard in LSA research. Then we took the SVD decomposition to get the top 200 singular values which produced three matricies [U S V]. Next we created a term-weighting as T = U*S. We called this matrix "tiny". Next we constructed a corpus from 1.9M words, having 60k terms and 5.5k documents, and called this matrix "twomillion". We further constructed a matrix from 3.5M words with 66k terms and 10k documents, and called it "fourmillion". This matrix was stemmed.

Since we were not looking for an ultimate test of whether these matricies were correct, just a confirmation that we had not computed meaningless matricies, , we looked at term-similarity for a subset of words that we arbitrarily and without design picked. The results are listed below.

		tiny	**twomillion**	**fourmillion**
father	mother	0.30	0.36	0.37
husband	wife	0.27	0.32	0.34
husband	married	0.15	0.37	0.16
doctor	nurse	0.21	0.41	0.35
doctor	hospital	0.20	0.26	0.43
marry	divorce	0.14	0.00	0.30
doctor	quickly	0.00	0.03	0.09
doctor	hotelier	0.01	0.04	0.11
happy	sad	0.00	0.13	0.01
marry	bowls	0.00	0.00	0.11
bracelet	brains	0.01	0.00	0.01
push	pull	0.00	0.02	0.07
push	fat	0.00	0.15	0.27

These results are largely satisfying, except for "happy-sad" and "push-pull", which are quite low in tiny, and "push-pull", "push-fat" and "marry-divorce" for the two million word matrix. It does appear that the "fourmillion" matrix has additional noise ("marry" "bowls" for example). The results weren't completely heartening, but they seemed reasonable enough that we chose to go on and try a test on the ultimate task.

7.4 Sentence Identification

7.4.1 Setup

For this experiment, we examined the performance of word overlap as compared to performance by using LSA on picking the sentence most likely to contain the answer to the question. Word overlap chooses the sentence with the most word contained in both the question and the answer. LSA chooses the sentence whose centroid has the highest cosine to the question's centroid.

We used a twomillion matrix which created from 1.9M term token with 60k term types 5.5k documents. We also used a verblarge matrix, which was a matrix stripped of everything but likely verbs (as determined by a POS tagger), from 2.9M term tokens, 7.8k term types, and 46.8k documents. Finally, we used Landauer's **General Adult** matrix as well.

We tested against the gold standard of marked-up sentences which Michael Littman tagged cbc.annot.txt. This file consists of the sentences which are used in answering the question. For some questions, multiple sentences were needed to construct the answer, and those were marked by Michael. In those cases, we counted our system as correct if we retreived any of the marked sentences.

We report results in terms of matches between the computer generated answer and the answer marked up by Michael. In cases where there was more than one sentence marked, we report success

if any of the sentences marked was retrieved. We initially dealt with out-of-vocabulary words by simple ignoring them. This did not seem to be approriate because of terrible performance, so we tried an arbitrary combination method. In this hack, we used word overlap between out of vocabulary words in each of the question and sentence, divided that number by the total number of overlaped words, and used as a mixture coefficient the percentage of words in the question which were out-of-vocabulary to mix this into the LSA similarity. An additional modification we tried was to remove words from the centroid computation and treat them as out-of-vocabulary words if they appeared less than X times in the corpus. We examined three levels (10,100,200) out of which 100 seemed to be the best. Additionally, we examined building a matrix from text which had been converted to lower case which also appeared to improve performance.

7.5 Results

Method	**Notes**	**Matrix**	**Percent Correct**	**Not In Overlap**
Overlap			40% (35/87)	
LSA		verblarge	29%(25/87)	3
LSA		twomillion	24%(21/87)	2
LSA	count cutoff-10	twomillion	25%(22/87)	3
LSA	count cutoff-100	twomillion	31%(27/87)	4
LSA	count cutoff-100, no caps	twomillion	35%(30/87)	5
LSA	count cutoff-200, no caps	twomillion	32%(28/87)	4
LSA		Landaur's General Adult	31%(27/87)	

example:

Question:Where *was* Jimmy Rankin *when he heard the news* of his brothers death
Overlap :Family friend Russell De Carle lead singer for Prairie Oyster said *was* stunned *when he heard the news* at the benefit.
LSA : Rankins brother Jimmy Rankin left a Farm Aid concert in Toronto after being notified of the death.

In the above example words which overlap are in italics.

7.6 Orthogonality

We measured the number of times when LSA was correct and word overlap was incorrect, and it turned out that for the matrix "twomillion" there were 8 times when "twomillion" was correct and word overlap incorrect – this turns out to be around 10% of the time (since the test set is 86 questions). Landauer's matrix was only different from overlap in 3/21 cases ($\sim$ 14% or 3% overall). This result suggests that LSA is capturing similarity lost in plain word overlap, and that using its results might be useful.

7.7 Conclusion

LSA didn't appear to be able to improve performance when used in this way. It is hard to say exactly why this happened. Out of vocabulary words clearly played a mjor role. Many questions ask about a particular person or location, both proper names and likely to be out of the vocabulary of the LSA matrix. Our method of handling these word was not optimal. Another way of understanding these results is to conjecture that perhaps LSA is not well suited to making the kind of fine-grained analysis required to distinguish text on the sentence level. Perhaps at that level, syntax is vital. In any event, our results did not justify spending more time figuring out how to make LSA work.

8 An Alternative Architecture

Reading comprehension tests consist of a short essay and some questions on a specific topic. It is assumed that the answers to the questions lie somewhere in the text and require minimal outside knowledge to locate. During the CLSP 2000 workshop, a computer system, Spot, was developed to take these tests. Spot piped the text through industry-standard taggers and each in succession embedded its mark-up into the text. After all of the taggers had finished, a separate module used the embedded tags to identify and rank possible answers for each question.

Spot was simple to implement and easy to augment with additional taggers, but this loosely coupled, pipelined architecture had some inherent disadvantages. It was difficult to integrate probabilistic outputs from taggers. Probabilities had to be thresholded, which resulted in a loss of information. In general, decisions needed to made before needed. These observations fit common intuitions about these types of systems; as [9] succinctly notes: "pipeline architectures suffer from a serious disadvantage: errors accumulate as they propagate through the pipeline."

This proposal outlines an integrated probabilistic alternative to Spot. This model is inspired by statistical machine translation research (SMT) and work in using SMT techniques for answer finding [2]. In an integrated probabilistic model the uncertainty of the taggers is retained until the final calculations. As a results, decisions are not made until needed and errors made early can be corrected by overwhelming evidence at later stages.

8.1 A Probabilistic Model

The task of reading comprehension is to find the best **term answer** $\hat{TA}$ to a question Q found in a story $\mathcal{S}$. This problem can be posed as a statistical estimation problem as follows:

$$\hat{TA} = \underset{TA}{\operatorname{argmax}}\ P(TA|Q,\mathcal{S})$$

For simplicity, we assume the term answer lies within a contiguous region in the text. A version of the text with brackets around a possible term answer we call the **annotated sentence** $\mathcal{AS}$, and the bracketed area $b(\mathcal{AS})$. We can now rewrite the above probability as follows:

$$\underset{TA}{\operatorname{argmax}}\ P(TA|Q,\mathcal{S}) = \underset{\mathcal{AS}}{\operatorname{argmax}}\ \delta(TA, b(\mathcal{AS}))P(\mathcal{AS}|Q,\mathcal{S})$$

δ is the Kronecker delta (1 if the terms are similar, 0 otherwise). We can continue and rewrite the second term as:

$$\begin{aligned}
\underset{\mathcal{AS}}{\operatorname{argmax}}\ P(\mathcal{AS}|Q,\mathcal{S}) &= \underset{\mathcal{AS}}{\operatorname{argmax}}\ \frac{P(Q,\mathcal{S}|\mathcal{AS})P(\mathcal{AS})}{P(Q,\mathcal{S})} \\
&= \underset{\mathcal{AS}}{\operatorname{argmax}}\ P(Q|\mathcal{AS})P(\mathcal{S}|\mathcal{AS}P(\mathcal{AS}) \\
&= \underset{\mathcal{AS}}{\operatorname{argmax}}\ P(Q|\mathcal{AS})P(\mathcal{AS}|\mathcal{S})P(\mathcal{S}) \\
&= \underset{\mathcal{AS}}{\operatorname{argmax}}\ P(Q|AS)P(\mathcal{AS}|\mathcal{S})
\end{aligned}$$

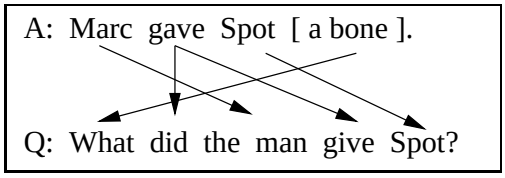
AS is defined to be a sentence annotated with a term answer (an **annotated sentence**). In these manipulations we assumed that Q only depends on AS not $\mathcal{AS}$ (the sentence containing the answer, not the entire story) and the Q and $\mathcal{S}$ are independent given $\mathcal{AS}$. Given these assumptions we have now reduced the problem to estimating two distributions:

1. $P(Q|AS)$ - the **translation model**
2. $P(\mathcal{AS}|\mathcal{S})$ - the **answer model**

In the remainder of the proposal we sketch strategies for parameterizing these two models. We illustrate some possible extensions to the model. Finally we show how to collect the necessary training data.

8.1.1 The Translation Model

The term $P(Q|AS)$ models the way in which an annotated sentence is related to a question. One way to quantify this relatedness is to hypothesize a process which transforms the annotated sentence into the question.



The above figure gives an example where syntactic re-ordering and lexical variation transform the annotated sentence into the question. This process resembles a translation process, and statistical machine translation (SMT) models appear to provide a jumping off point for estimating this term. Applying the IBM model [1] [8] to our application would result in the following formulas:

$$P(Q|AS) = \sum_{\alpha} P(Q, \alpha|AS)$$

$$P(Q, \alpha|AS) = \prod n(\phi|s_i) \prod t(q_j|s_i) \prod d(i|\alpha_j, l, m)$$

In the above parameterization Q is the question and AS the annotated sentence as before. α is the alignment of words in one sentence to words in the other (what the arrows in the above figure represent). The n term is the **fertility model** and represents the likelihood of one word s_i translating into ϕ distinct words. The t term is the **lexical model**, the probability of one word s_i translating into another word q_j. The d term is the **distortion model** and represents the likelihood of the word in position j in the annotated sentence going to position i in the question given the length of the sentence l and that of the question m.

This model allows for much of the phenomena involved in the transformation from questions to answers: lexicon change (WH words in questions), lexical change ("gave" to "give"), and syntactic

change (WH movement). Of course, some of the particulars of the problem are different from translation. Translation is less one-to-one, since much from the source can be dropped. The WH word and term answer are privileged elements in the alignment. A complete generation model is not needed, instead an alignment model will suffice, One additional caveat is that this translation model seems to be ill-suited for finding long answers. While it is conceivable that the model could be extended to handle long answers, initially work will be strictly on short, phrasal answers. Despite these differences, there seem to be many insights to glean from work in SMT, and these models will be the starting point for estimating the translation term.

One of the main topics of further research in this model is a lexical movement model to replace the distortion model. The distortion model is not ideal since the syntactic change is highly structured and well understood. A more structured model (e.g. [13]) might work more effectively.

8.1.2 The Answer Model

The term $P(\mathcal{AS}|\mathbb{S})$ describes the range of possible answers the system can return. One simple distribution would allocate uniform probability to all noun phrases. Alternatively, it might be useful to parameterize the type of the question QT as follows:

$$\begin{aligned} P(\mathcal{AS}, QT|Q, \mathbb{S}) &= P(Q|AS, QT)P(\mathbb{S}|\mathcal{AS}, QT)P(\mathcal{AS}, QT) \\ &= P(Q|AS, QT)P(\mathcal{AS}, QT|\mathbb{S}) \\ &= P(Q|AS, QT)P(\mathcal{AS}|QT, \mathbb{S})P(QT) \end{aligned}$$

Adding this latent variable QT might allow better probability estimates for the possible annotations. For example, noun phrases containing names of people might be given higher likelihood of begin answers in cases where the question is a "who" question. It might also make sense to privilege some question types, so that $P(QT)$ is not uniform.

The exact formulation is not completely defined, and finding the most efficient and effective parameterization will be a key direction of research.

8.1.3 Coreference Resolution

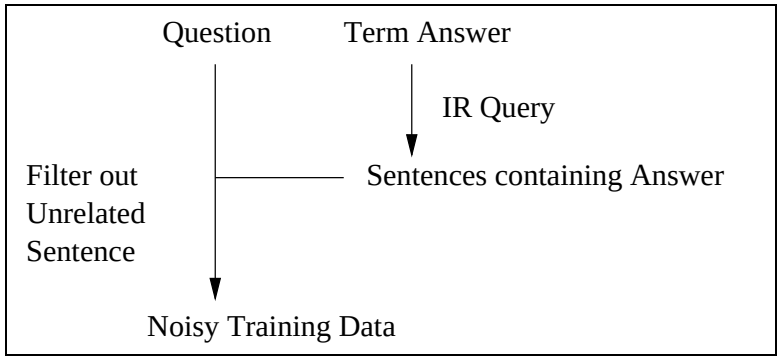
Coreference resolution has been shown to be useful in taking reading comprehension tests [7], but the above model does not incorporate this information. One could integrate a probabilistic coreference resolver (e.g. [6]) naturally in the following way:

$$\begin{aligned} P(cr(\mathcal{AS})|Q, \mathbb{S}) &= P(Q, \mathbb{S}|cr(\mathcal{AS})) \\ &= P(Q|cr(AS)P(cr(\mathcal{AS})|cr(\mathbb{S}))P(cr(\mathbb{S})|\mathbb{S}) \end{aligned}$$

In this equation cr is the function which resolves coreference. In the above equation the first two terms are the translation and answer model described earlier. The third term, $P(cr(\mathbb{S})|\mathbb{S})$, can be generated from the probabilistic coreference model and plugged in. A looming research question is how to search this space efficiently. Possible a modified beam search will be necessary to examine all reasonable possibilities. Other extra-sentential information could be integrated into the model in a similar way.

8.2 Training Data Collection/Generation

This model relies on a large training set in order to be able to estimate these probabilities. The amount of data needed might be less than for SMT since the translation process is more constrained. However, generating a data set automatically will be necessary. Here's one possible outline of how to do that.



Given a set of questions and their term answers, perform an IR query to retrieve sentences containing those answers. Then look for sentences which have words in common with the questions, generating noisy training data. As the model improves, use it to generate cleaner data and train more sophisticated models.

8.3 Future Work

The following steps comprise a plan for pursuing this work.

1. Generate Training Data
2. Build a system using off-the-shelf components
3. Work on a lexical movement model to replace the distortion parameter
4. Augment the model with coreference resolution

This work would give an idea of the feasibility of this research program and might hint at the kinds of problems that are likely to occur. The training data would be useful for anyone who wants to pursue a statistical approach to this problem.

9 Conclusion

Below we attempt to explicate some of the contributions we feel the project made this summer:

- Corpus creation: Two new corpora available: CBC (2250 questions), IFI (250 questions)
- Modeling the Problem of Reading Comprehension
  - Decomposition into separately measurable sub-tasks: question analysis, location of answer region (HotSpotting), location of answer phrase (PinPointing)
  - Creation of metrics and automated evaluation techniques corresponding to these sub-tasks, including new measures of answer correctness based on answer conciseness as well as "coverage" of answer key concepts,
  - New probabilistic generative model outlined, where text passages generate "source" questions, analogous to statistical MT approach
- Provided a number of useful baseline system performance measurements,
- Found an upper bound for systems that use only QA, entity taggers, and word overlap
  - This is the score for a system with perfect QA, perfect tagging, and perfect HotSpot but no information to distinguish candidates in the same hot-spot.
  - Thus it is necessary to extracting matching relations, either syntactic or semantic, from the question and hot-spot

9.1 Future Work

We still have a great deal of work to do within the current framework. We need to better understand Spot is not at the upper bound of systems of its design. Thus, we need to stabilize Spot, do a thorough error analysis, and set of ablation experiments.

In addition, we need to improve the ranking systems and incorporate the syntactic features discussed.

10 Acknowledgements

We received a great deal of support from people at the MITRE corporation:

- Lynette Hirschman helped with many aspects of the basic conception of the project
- Lisa Ferro, Tim Bevins, Jessica Teague, and Beverly Nunan created and/or prepared the CBC and IFI corpora. In addition, much of the hand tagging required for the upper bound calculations was performed by Lisa Ferro and Beverly Nunan

- John Burger, Inderjeet Mani, Jason Rennie helped with numerous modules of the system

We would also like to thank Michael Littman, Helmut Schmid, Karen Kukich for their technical efforts on behalf of the project.

We would like to thank Bill Byrne, Pascale Fung, Lynette Hirschman, Warren Greiff, and David Yarowsky for extensive help with the section describing an alternative architecture.

We would also like to thank the system administration at the workshop for supporting our computing needs: Jacob Laderman, Lynny Liu, Meng Wang, and Damon Brown.

Amy Berdann, Wilhelmena Braswell, Sarah James, and Aaron Moak took care of many other needs such as cookies, cakes, coffee, paper, pencils, overhead projectors, plane tickets, per diem checks, etc.

Thanks finally to Frederick Jelinek, William Byrne, and Sanjeev Khudanpur for providing general guidance and support.

References

[1] A. L. Berger, P. F. Brown, S. A. Della Pietra, V. J. Della Pietra, J. R. Gillett, J. D. Lafferty, R. L. Mercer, H. Printz, U. Lubos. 1994. "The Candide System for machine Translation". Proceedings of the 1994 AHPA Workshop on Human Language Technologies.

[2] A. L. Berger, R. Caruana, D. Cohen, D. Freitag, V. Mittal. 2000. "Bridging the lexical chasm: Statistical approaches to answer-finding". Proceedings of the 23rd Annual Conference on Research and Development in Information Retrieval (ACM SIGIR). Athens, Greece.

[3] E. J. Breck, J. D. Burger, L. Ferro, L. Hirschman, D. House, M. Light, I. Mani 2000. "How to Evaluate your Question Answering System Every Day and Still Get Real Work Done". Proceedings of the Second conference on Language Resources and Evaluation (LREC-2000).

[4] E. 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, L. Zorn 2000. "Reading Comprehension Programs in a Statistical-Language-Processing Class". Proceedings of Workshop on Reading Comprehension, NAACL/ANLP-2000.

[5] R. Flesch 1943. "Marks of Readable Writing". PhD dissertation.

[6] N. Ge, J. Hale, E. Charniak. 1998. "A Statistical Approach to Anaphora Resolution". Proceedings of the Sixth Workshop on Very Large Corpora.

[7] L. Hirschman, M. Light, E. Breck, J. D. Burger. 1999. "Deep Read: A reading Comprehension System". Proceedings of ACL 1999.

[8] K. Knight. 1999. "A Statistical MT Workbook." *http://www.isi.edu/natural-language/people/knight.html*

[9] S. Miller, H. Fox, L. Ramshaw, R. Weischedel. 2000. "A Novel Use of Statistical Parsing to Extract Information from text". Proceedings of the 1st meeting of NAACL

[10] H. T. Ng, L. H. Teo, J.L.P. Kwan 2000. "A Machine Learning Approach to Answering Questions for Reading Comprehension Tests". Proceedings of EMNLP/VLC-2000 at ACL-2000

[11] E. Riloff, M. Thelen 2000. "A Rule-based Question-Answering System for Reading Comprehension Tests". Proceedings of Workshop on Reading Comprehension, NAACL/ANLP-2000.

[12] W. Wang, J. Auer, R. Parasuraman, I. Zubarev, D. Brandyberry, M.P. Harper 2000. "A Question Answering System Developed as a Project in a Natural Language Processing Course". Proceedings of Workshop on Reading Comprehension, NAACL/ANLP-2000.

[13] D. Wu. 1995. "An Algorithm for Simultaneously Bracketing Parallel Texts by Aligning Words". Proceedings of ACL 1995.

Appendix: Reading Comprehension as a Question Answering Task

We are often asked how the reading comprehension task is different from a general question answering task, such as the TREC Q/A task. From an applications perspective, the main difference is that the reading comprehension task is "document-specific". Each question is asked with respect to a specific document and the answer must be located from within that document. In contrast, a general-purpose question answering task poses questions without reference to any specific document. The goal is to find the answer to a question by searching all available resources. For example, the TREC Q/A task involved finding the answers to questions by searching a large text collection.

Document-specific question answering poses different challenges than general question answering because an answer generally appears only once in a document.[4] Therefore a document-specific Q/A system usually only has one shot to find the answer. In a general Q/A task, many documents often contain the answer to a question, so there are multiple opportunities to find the answer. To illustrate this point, we manually reviewed 51 randomly chosen TREC-8 questions and identified all answers to these questions in the text collection. An answer was defined as a string of text which answered the question or coreferred with text that did. The TREC-8 histogram is shown below, where the bottom row indicates the number of questions that had the number of answer occurrences shown in the top row. For example, 13 questions had only 1 answer occurrence in the text collection, 10 questions had exactly 2 answer occurrences in the text collection, etc. We also generated a similar histogram for CBC texts used in our reading comprehension experiments.

Histogram of TREC-8 answer occurrences

1	2	3	4	5	6	7	8	9	12	18	27	28	61	67
13	10	6	4	2	3	5	1	1	1	1	1	1	1	1

Histogram of CBC answer occurrences

1	2	3	4	5	6
176	36	4	2	0	1

The two histograms confirm that there is a big difference in the number of answer occurrences for the TREC Q/A task and our reading comprehension task. On average, there are 7.2 occurrences of each answer in the TREC collection, while there are only 1.25 occurrences of each answer in a CBC text. The histograms show that the number of answer occurrences varies widely, but it is clear that there are often multiple opportunities to find the correct answer to a TREC question. 75% of the TREC questions have 2 or more answer occurrences, and 55% of the TREC questions have 3 or more answer occurrences. In contrast, only 20% of the CBC questions have 2 or more answer occurrences, and only 3% of the CBC questions have 3 or more answer occurrences.

[4] When we say that an answer "appears" in a document, we mean that it occurs in a context sufficient to answer the question. For example, consider the question "Who is president of the United States?". A document may contain many occurrences of Bill Clinton, but we only count the occurrences that mention that he is president.

One reason to focus on the reading comprehension task is to ensure that a Q/A system can find the answer to a question when there is only one opportunity. Another benefit of document-specific Q/A is that multiple opportunities to find an answer can hide the strengths and weaknesses of a Q/A system. A system that finds an answer 10 times, in both easy and difficult contexts, will get the same credit as a system that finds the answer only once in an easy context. There is less incentive for a Q/A system to try to recognize answers in difficult contexts. And ironically, the larger the text collection, the worse this problem becomes.

Using reading comprehension exams as a testbed for document-specific Q/A also has several advantages, including:

- Reading comprehension exams are a readily available source of questions and answers. Since these exams were created to judge the reading ability of children, they are an objective way to judge the question answering ability of our computer models.
- The exams are available at increasing levels of difficulty based on grade level, allowing us to quantitatively assess a Q/A system's level of performance. The availability of different grade levels also provides us with increasingly difficult Q/A tasks to pursue.
- Many reading comprehension exams have difficulty assessments and question categories that allow us to qualitatively evaluate the ability of the Q/A system.
- We can compare the performance of our Q/A system with benchmarks of human performance on these exams.
- Insights learned about reading comprehension can serve as a springboard for educational applications related to reading and foreign language learning.