CS 598DNR Project 3 (Team 4): Final Story Comprehension System

William Lee wwlee1@uiuc.edu
Xu Ling xuling@uiuc.edu
Yoonkyong Lee ylee11@uiuc.edu
Geoffrey Levine levine@uiuc.edu
Benjamin Liebald liebald@uiuc.edu

12/15/2004

1 Introduction

This paper describes a question-answering system using a combination of ideas from [1], [3], and [8], in addition to our own refinements to some advanced techniques described in [2], [6], and [9].

Our contribution in this paper includes: (1) a design of an extensible story comprehension architecture, (2) a richer and more unified story comprehension dataset than the one used in [3], and (3) a technique to integrate advanced scoring functions using training examples and linear regression.

We have performed evaluation and analysis of this system on the dataset that we have enhanced from [3]. Level 2 and 5 are used as training set, and level 3 and 4 are used for test set. On level 3 and 4, we have achieved an accuracy of 42.27%, comparing to our baseline system's 39.25%. The combined average accuracy for all four levels is 49.02%.

2 System Architecture

The enhanced Bag of Word (BOW) approach, as described in [4], has shown to be extensible to the new features that we would like to implement in [2], [6] and [9]

Figure 1 shows the modified architecture that we have used in this paper. On top of the feature sets in [4], we have added the following:

1. An Expected Answer Filter that tries to match the expected answer for a question to the sentence type
2. A trainable composite ranking function that has the ability to train its weights on combination of scorers
3. Implementation of eight different scorers that can be used in the composite ranking function in any combination
4. Additional scorers using Okapi, tree edit-distance, and expected answer.

Here is an overview of the system flow:

1. Our data preprocessing module would parse from the original data set and construct a multi-column format used internally in our system.
2. For each story, we would construct sentence models and question models. In our case, our model consists of a BOW representation and the unmodified original attributes that tagged each word.
3. The models are then passed to the filters. See Section 3 for more details on filters.
4. The system then passes the filtered question and sentence models to a trained composite ranking function. Given a question, the ranking function will sort the sentences by their relevance based on several scorers. Section 4 describes this trained ranking function and scorers.
5. With the ranked list of sentences, the Postprocessing module then picks the most likely answer. In some question-specific rules, the sentence with the best score may not be the optimal one to pick.
6. The Evaluation module would match generated answers with real answers and returns the final results.

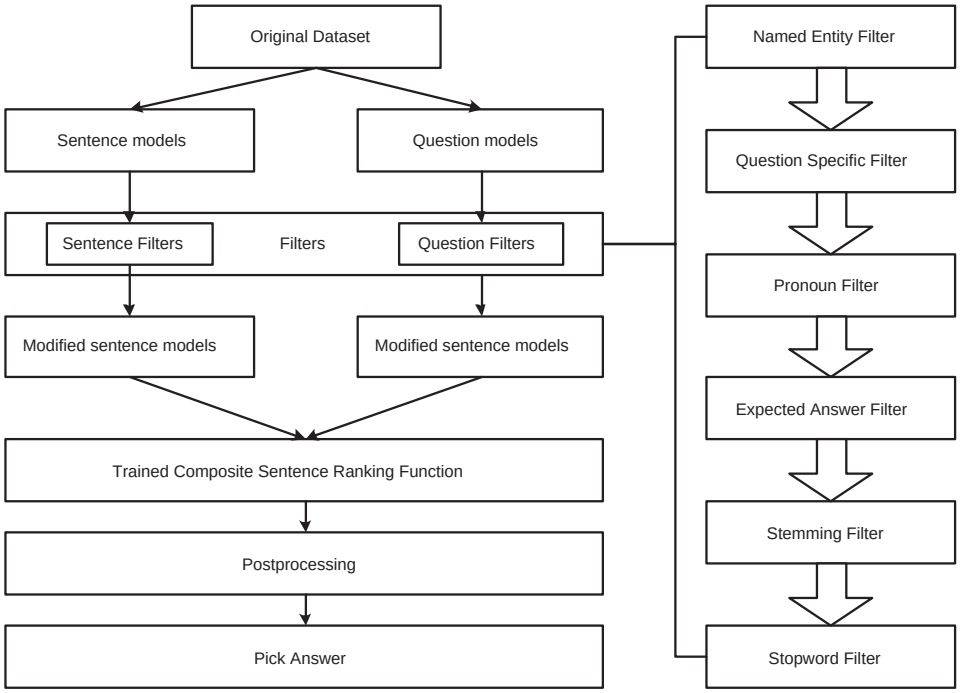

Figure 1: A Modular Story-Comprehension System Architecture

3 Filters

The filters are modifiers to the sentence and question BOW models as described in Section 2. Based on the tagged information such as as name for each sentence, different filters can eliminate words in the bag, add special tokens with given weight, or merge two tokens together.

We have implemented the following filters.

3.1 Pronoun Coreference Filter

Of course, a necessity for our question answering system is the ability to make sense out of pronoun references. We are fortunate enough to have pronoun coreference information provided to us by the MITRE corporation. The data comes in an xml format associating an identification id along with a reference id for each entity in the story. How exactly to take advantage of this information, though, is not particularly clear. Three out of the five systems cited in [1] make use of pronoun coreference information, but the paper fails to list any implementation specifics.

We make use of this information by adding all such identification numbers to our question and answer bags. Essentially, the identification numbers are treated just like other words, such that sentences making a reference to the same entities as a particular question will be scored higher, just as sentences sharing words with the question would.

3.2 Named Entity Filter

Named entity data is also provided by the MITRE corp. Entities are tagged as one of person, location, or organization. The named entity filter adds to each of the question and answer bags special tags indicating when entity data of a specific type is in a sentence. This way, sentences that contain the same type of entity information as is referred to in the question will receive a bonus by the scoring module. In addition, the question specific filter contains rules which are based on the presence on certain named entity data in the question and answer sentences.

3.3 Stopword Filter

In compliance with the Brown paper [1], we have used the same stopword classes. The stopword filters, when turned on, will filter out the following words from both the question and sentence BOW:

```
be am is are were was
have had
do did done
and or to in at of a
the this that which
```

3.4 Stemming Filter

Stemming has shown to improve the performance in [3], [8], and [1]. Since we were unable to get Steven Abney's stemmer, as used in [3], to work properly, we settle for the simple Porter stemmer [5] instead. We use the default Martin Porter's Perl implementation in our experiments. The stemmer filter applies stemming on both question and sentence BOW.

3.5 Question-Specific Filters

For the implementation of question-specific rules, we more or less followed the proposed rules of the Utah system [8] and adapted them to our modular architecture. Thus, we have the Question-Specific Filter, to process the question and answer bags, adding special tags corresponding to individual question specific rules, and the Question-Specific Scorer to recognize and assign an overall score

based on these weights. Descriptions of the specific rules, and some implementation details can be found in the Question-Specific Scorer section.

3.6 Expected Answer Filter

The expected answer filter makes use of additional data provided by the Cognitive Computation Group (CogComp in the sequel) at UIUC. This data indicates for each question one or more expected answer types depending on the question structure. These types can for example be "location", "person", "description", "entity" etc. In addition, these categories were organized in a hierarchical manner, i.e. each main category had several subcategories associated with it.

Also, the candidate answer sentences were tagged with more fine-grained named entity tags, using a software provided by CogComp. Unfortunately, the question-classification tags and the named entity tags were disjoint, i.e. they used different hierarchies and ontologies.

Our approach therefore was to learn correlations between these tags by looking at the correct answer sentences in stories at levels 2 and 5. For each pair of type $< q, n >$, where q is a question-specific tag and n is a named entity tag, we count the number of occurrences in the correct answer sentences of levels 2 and 5.

In section 4, we explain how these statistics were used to assign scores to each sentence during testing.

4 Scorers

Scorers are used to score each sentence for ranking. We have implemented several scorers in our project.

4.1 Baseline Scorers

For our baseline system [4], we have implemented two scorers: the bag of word (BOW) scorer used by DeepRead [3] and the TF/IDF (term frequency/inverse document frequency) scorer described in [1].

Given the vocabulary $W = \{w_1, ..., w_n\}$, we define the bag of word for the sentence as a set S such that $S = \{s_1, s_2, ..., s_n\}$, where s_m represents the frequency for word w_m in the sentence. Similarly, we define BOW of question as $Q = \{q_1, ..., q_n\}$, where q_n represents the frequency for word w_m in the question. We then can define the size of the intersection and union between S and Q as

$$\begin{aligned} |Q \cap S| &= \sum_{i=1}^{n} min(s_i, q_i) \\ |Q \cup S| &= \sum_{i=1}^{n} s_i + q_i \end{aligned}$$

. Given the union and the intersection, we define the scoring function for BOW as

$$score_{BOW}(Q, S) = \frac{|Q \cap S|}{|Q \cup S|}$$

.

For the TF/IDF scorer, let $tf(w, S)$ be the term frequency for w in sentence model S and $df(w, B)$ be the the count of sentences in the background model B that contains the term w. It

is worth nothing that the background model B, in our case, represents one particular story. In addition, let w_i be the i^{th} term in the sentence S. The scoring function then becomes

$$score_{TFIDF}(Q, S, B) = \sum_{i=1, tf(w_i,Q)\neq 0}^{n} \frac{tf(w_i, S)}{df(w_i, B)}$$

. Essentially, this is the same as giving a weight for each word so its score will be discounted if it appears very often in the story.

4.2 Trained Composite Ranking Function

The baseline scorers contain hand-tuned parameters for adding scores for various features such as pronoun coreference, named entities, and question specific rules. Those metrics generally return scores with different weights. Therefore, there is no systematic way to combine the scores.

Harabagiu [2] describes an approach to accurately retrieve the most relevant paragraphs using a Perceptron-trained function. The training problem is first set up as training an comparator function that tests the relevance for two sentence models S_1 and S_2 given the question model Q. Intuitively, the problem can be thought of training a function like:

$$moreRelevant(S_1, S_2, Q) = \begin{cases} 1 & \text{if } S_1 \text{ is more relevant than } S_2 \text{ given } Q \\ -1 & \text{otherwise} \end{cases}$$

. Like Harabagiu, if each scorer i has a scoring function $score_i$ and a weight w_i, we can use a linear equation to combine the scores for different scorers:

$$f(S_1, S_2, Q) = \sum_i w_i(score_i(S_1, Q) - score_i(S_2, Q))$$

We then use $f(S_1, S_2, Q)$ in our comparator function $moreRelevant()$:

$$moreRelevant(S_1, S_2, Q) = \begin{cases} 1 & \text{if } f(S_1, S_2, Q) > 0 \\ -1 & \text{otherwise} \end{cases}$$

We have modified this approach to fit the context of our story comprehension system (Figure 2). It is important to note that Harabagiu's features are simple word matching counts. We have built and used more sophisticated scorers as the basis for the composite ranking function.

We have reimplemented the scorers used in our baseline system so they work in the new system. The old scorers include the BOWScorer, TFIDFScorer, NamedEntityScorer, PronounScorer, and QuestionSpecificScorer. BOWScorer and TFIDFScorer are the same as the baseline scorer. We have enhanced the NamedEntityScorer and PronounScorer. See Section 4.2.1.

We use the corpus for level 2 and 5 for training. For each story in those levels, we create training examples in the form (Y, S_1, S_2, Q) for all combinations of (S_1, S_2, Q) in the story.

For each triplet (S_1, S_2, Q), $Y = 1$ if S_1 is the answer to question Q, and $Y = -1$ otherwise.

We then convert (Y, S_1, S_2, Q) into f. We can do so by running the scoring function for each scorer on (S_1, Q) and (S_2, Q) in order to calculate delta in the linear equation.

Finally, we feed the examples to a linear regression engine and learned the weight w_i for each scorer.

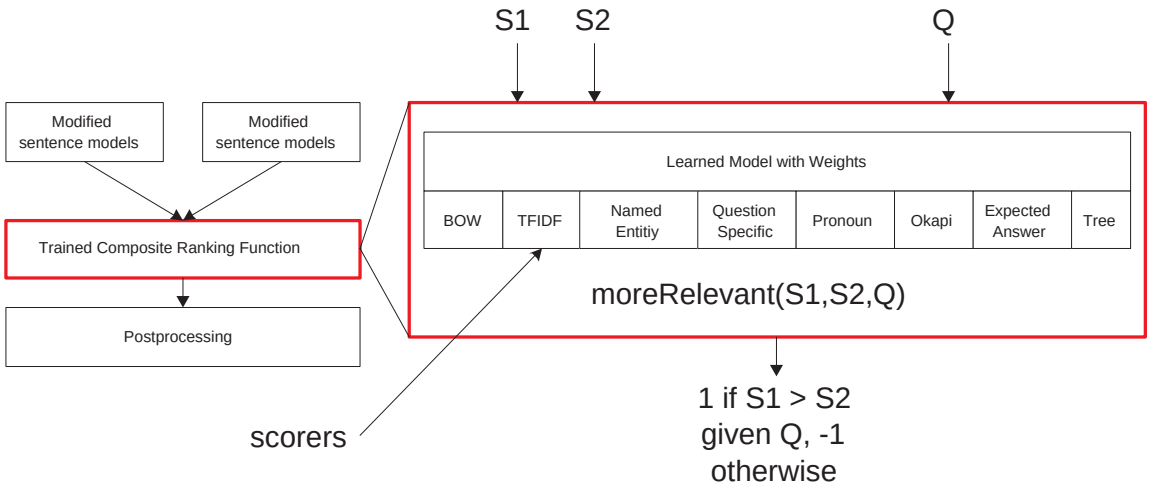

Figure 2: Trained Composite Ranking Function

4.2.1 NamedEntityScorer and PronounScorer

We have also changed the NamedEntityScorer and PronounScorer to follow the TFIDF scheme. We downweight the named entities or pronoun references if they appear in many sentences. More specifically, if we define function $tw(tag, S)$ be the term weight function that returns the sum of all the tag's weight (the tag can be either a named entity or a pronoun reference), we can derive the scoring functions as:

$$score_{namedentity}(Q,S,B) = \sum_{i=1,tw(n_i,Q)\neq 0}^{n} \frac{tf(n_i,S)}{df(n_i,B)}$$

$$score_{pronoun}(Q,S,B) = \sum_{i=1,tw(p_i,Q)\neq 0}^{n} \frac{tf(p_i,S)}{df(p_i,B)}$$

4.2.2 Okapi Scorer

Borrowing from the traditional Information Retrieval model, the Okapi formula has been used successfully in document retrieval. Okapi has the advantage in incorporating the length normalization and the query frequency factor on top of the traditional TFIDF model. In our context, we treat each sentence as a document and a story as a collection of document. Thus, the formula is given as:

$$score_{okapi}(Q,S,B) = \sum_{i=1,tf(w_i,Q)\neq 0}^{n} \frac{(k_1+1)tf(w_i,S)}{k_1((1-b)+b\frac{len(S)}{avsl(B)})} \cdot \frac{(k_3+1)tf(w_i,Q)}{k_3 tf(w_i,Q)} \cdot \ln \frac{(N-df(w_i,B)+0.5)}{df(w_i,B)+0.5}$$

where:

- $len(S)$ is the length of the sentence S, and $avsl$ is the average sentence length for the background model.
- The k_1, b, and k_3 are Okapi constants. Initial tweaking indicates the values of $k_1 = 1.1$,$b = 0.3$, and $k_3 = 0$ (not making any bias toward the the query frequency) seem to work best.

4.2.3 Question-Specific Scorer

Due to the differences in flow of control of ours and the Utah system, we were unable to use a few specific rules. The following subsections provide a detailed explanation of the rules that we used, and also mention which rules could not be implemented within our system.

Before a question-specific filter can be applied, one obviously first needs to figure out which type of question the current question sentence belongs to. Due to the nature of the questions though, this is almost trivial, since the first word of the question sentence indicated its type with 100% accuracy. After determining the type of question, the sentence is then passed on to the filter associated with this question type.

In general, these filters look at the question sentence and each sentence in the story and check whether a certain predicate is true (e.g. "The question contains a MONTH and the sentence contains either *today, yesterday, tomorrow,* or *last night*"). If it is true, a certain number of points is added to the overall score of the sentence, depending on the importance of the predicate. Points are discretized into 4 levels: **clue** (+3), **good clue** (+4), **confident** (+6) and **slam dunk** (+20). These discretization levels were also adapted from the Utah system. A possible improvement for the next part of this project would be to learn these weights from training data.

For almost all question types, the Utah system starts off by assigning points to sentences according to a *WordMatch* function. In the implementation of the question specific rules filter, we completely ignored this rule. As other parts of the system are based on these word matches, and we get the same effect using this different approach.

WHERE questions

The simplicity of the WHERE rules seems to imply that this should be one of the easiest question types to answer. The Utah system only uses two basic rules:

1. If the sentence contains a location preposition (such as "in", "at" or "near"), then a **good clue** score is added to the total score of the sentence.

2. If a sentence contains a LOCATION (the Utah system uses a predefined list of locations, we use the available named entity data), then a score of **confident** is added to the total score of the sentence.

Implementation of these two rules in our system is straightforward. For rule 1, we define a list of location prepositions (since the Utah system did not specify the entire list, we came up with our own). For each of these prepositions, we check whether it is in the question bag. If yes, we add a **good clue** score to the word. If it doesn't exist in the bag yet, we add it to the bag and initialize it with **good clue** score. Similarly for rule 2, where we use a special tag ":QS:NE:LOCATION:" which is then added to the bag. Then, sentences that are found to have Location information according to our named entity data are given a score of **confident**.

WHO questions

The rules for the WHO questions are as follows:

1. If the question does not contain a name and the sentence does contain a name, then it gets a score of **confident**.

   This rule was implemented using the same approach as for most other rules: Check if the question bag contains a name (indicated by NE tag). If it doesn't, then add the special tag "QS:NE:PERSON" to the bag and initialize it with a score of **confident**.

2. If the question does not contain a name and the sentence contains the word "name" then it gets a score of **good clue**.

   The implementation of this rule is analogous to the one above.

3. If the sentence contains either a name or the reference to a person (e.g. an occupation such as "writer"), then it gets a score of **good clue**.

   As we don't explicitly have the list of human related words that the Utah system authors had, we check WordNet to find if any words in the sentence are a descendant of "HUMAN" sense as defined by *WordNet*. If so, the sentence is given the bonus points associated with **good clue**.

WHAT questions

These are the rules used for WHAT questions by the Utah system:

1. If the question contains a MONTH expression (the 12 months of the year) and the sentence contains *today, yesterday, tomorrow,* or *last night*, then a **good clue** score is added to the sentence.

   We implemented this rule by checking for each MONTH expression whether it existed in the question. If it did, we checked for each of the above mentioned words whether they existed in the question bag. If they did, we added a **good clue** score to them, otherwise we added them and initialized them with the same score.

2. If the question contains the word "kind" and the sentence contains either "call" or "from", the score of the sentence is increased by **good clue**. If the question contains the word "name" and the sentence contains either "name", "call", or "known", a **slam dunk** score is added to the sentence.

   We implemented these rules by checking for the existence of the respective words in the question bag and then adding the respective score to the sentence words in the question bag.

3. If the question contains a phrase like "name of ¡x¿" and the sentence contains a proper noun whose head noun matches with "¡x¿", the sentence is awarded an additional score of **slam dunk**.

   We were unable to implement this rule since we did not have PP attachment data available.

WHEN questions

For the WHEN questions, the Utah system only applies the general word match filter if the sentence contains a TIME expression, where TIME expressions are years from years from 1400-1999, the 12 months of the year, and some more unspecified ones. Since we did not have a similar list of time expressions available, we modified this rule to apply in all cases, regardless of whether the sentence contained such an expression or not.

The remaining rules for the WHEN questions are:

1. If the question contains the string "the last" and the sentence contains either "first", "last", "since", or "ago", then it is assigned a score of **slam dunk**.

2. Similarly, if the question contains either "start" or "begin" and the sentence contains either "start", "begin", "since", or "year" then it is assigned a score of **slam dunk**.

The implementation of these rules is analogous to the implementations for other types of questions as described above.

WHY questions

WHY questions are handled differently in the Utah system. Instead of just assigning points to each sentence in one iteration, the system first performs a pure word match scoring and then isolates the best x sentences according to this score. Referring to this set of sentences as BEST, the following rules are then applied to all sentences in the story:

- If a sentence is in BEST, assign it a score of **clue**.
- If a sentence immediately precedes a sentence in BEST, assign it a score of **clue**.
- If a sentence immediately follows a member of BEST, assign a score of **good clue**.
- If the sentence contains "want", then increment its score by **good clue**.
- If the sentence contains either "so" or "because", then increment its score by **good clue**.

Due to the two iterations, this algorithm didn't fit well into our architecture and so we had to tweak it a bit to make this work. First we apply the scoring module as usual to associate a score with each question. From there, the rest of these rules are implemented in the postprocessing module. In the next phase of this project, we might think of a cleaner approach to WHY questions.

Dateline rules

As explained in the Utah paper, the dateline often contains the answer to WHERE or WHEN questions, so we need to take it into account for these types of questions as well.

Following the Utah system, we implemented the following rules:

1. If the question contains the word "happen", then the dateline's score is incremented by **good clue**.
2. If the question contains both "take" and "place" then the dateline's score is incremented by **good clue**.
3. If the question contains "this", then the dateline's score is incremented by **slam dunk**.
4. If the question contains "story", then the dateline's score is incremented by **slam dunk**.

Implementation of these rules follows the general pattern described above. A special tag is added to the questions and dateline bags, such that if any of these conditions are satisfied, the scorer will assign to the dateline the corresponding number of points. In the current version of the data, it should be noted that the dateline is not separated from the first sentence of the story, potentially enhancing results a small amount.

4.2.4 Expected Answer Scorer

As mentioned in the Expected Answer Filter section, each question has an associated set of expected answer types, and each potential answer has associated a set of named entity types. The goal of the Question Type Scorer is to assign high scores to sentences that contain the same named entity types as would be expected based on the question's set of expected answer types. Precisely, we wish to assign to each sentence a score based on the log likelihood that this sentence answers a question with the prescribed set of expected answers. Let **QC** denote a boolean vector indicating

the presence and absence of each expected answer type within the question, and let **NE** denote a similar vector indicating the named entity types that appear in the potential answer sentence, we assign to the candidate answer the score:

$$S = log\ p(\mathbf{QC}|\mathbf{NE}) \tag{1}$$

By Bayes rule, this equals,

$$S = log\frac{p(\mathbf{NE}|\mathbf{QC}) * p(\mathbf{QC})}{p(\mathbf{NE})} \tag{2}$$

$p(\mathbf{QC})$ is constant over all candidate answers, and thus,

$$S - log\ p(\mathbf{QC}) = S' = log\frac{p(\mathbf{NE}|\mathbf{QC})}{p(\mathbf{NE})} \tag{3}$$

Making the assumption that the named entity data is independent given the set of question classes,

$$S' = log \prod_{n \in \mathbf{NE}} \frac{p(n)|\mathbf{QC})}{p(n)} = \sum_{n \in \mathbf{NE}} log\frac{p(n|\mathbf{QC})}{p(n)} \tag{4}$$

If we make the assumption that the question classes occur independently, both unconditionally, and conditioned on the presence or absence of any one named entity type, several applications of Bayes rule yields,

$$S' = \sum_{n \in \mathbf{NE}} ((\sum_{q \in \mathbf{QC}} log\frac{p(n|q)}{p(n)}) + log\ p(n)) \tag{5}$$

$$S' \sum_{n \in \mathbf{NE}} ((\sum_{q \in \mathbf{QC}} log\ p(n|q)) - (log\ p(n))^{|\mathbf{QC}|-1}) \tag{6}$$

Statistics to approximate both of the probabilities, $p(n|q)$ and $p(n)$ are gathered by our Expected Answer Filter, and thus a score is assigned to each question, candidate answer pair.

4.2.5 Tree Scorer

The baseline scores, both the BOW scorer and the TF/IDF scorer, focus on exploiting semantic information like pronoun coreferences and named entities. However, we can also exploit the syntactic information. For example, assume that we have a following question that "Who lives in Greenland?"(Story2-20) The correct answer is "**Eskimos** live in Greenland," which has the exactly same sentence structure as the question. To exploit this structural information, we implemented a tree scorer based on [7].

This scorer exploits both syntactic and semantic information. The scorer is implemented in three phases. First, we transformed the given column-format data into the tree-format data based on the dependency graphs. Then, we implemented the basic tree matching algorithm described in [7]. Finally, we introduced a new heuristic to add more semantic information.

The preprocessing is the most expensive part in this scorer. First, we transformed the given fdg data into our column-format. There were some inconsistencies between these two column-formats. Since we splitted some words or combined them, the given dependency graph indices were screwed up. We checked every dependency graph index and made sure that it is reasonable and has one root, which is the main verb. Then, we constructed dependency trees based on these modified graphs. However, we have found that sometimes the original dependency graphs are not correct. In this case, if there were certain corrections, we corrected them manually. But, if there were not, we just leave them as they were. Because of this, we cannot guarantee that the tree scorer will show its optimal performance.

To match a question only with parts of the story sentence, the tree edit distance was introduced as a distance measure for matching between the tree representations. We achieved this by employing an approximate tree matching approach in ([7]).

Following [7], we consider ordered labeled trees in which each node is labeled by some information and the order from left to right of its children is important. There are three operations that can transform an ordered labeled tree to another: *deleting* a node, *inserting* a node, and *changing* a node.

The associated cost for each operation is mainly following the definition in [7], and new heuristic is tried in our implementation to use more semantic information, which will be explained later.

Our Tree Scorer returned the score of each question-sentence-pair by negating the minimum cost sequence of operations from among those that can be used to map the question tree into sentence tree. We use the SUBTREE REMOVAL algorithm developed in Zhang and Shasha ([10]), which is an efficient dynamic programming based algorithm. Since in our experiments we didn't restrict the cost function to satisfy the triangularity property, the modification introduced in [7] was used to compute the approximate tree matching cost.

Because the structure of a sentence might be quite different from that of a question, for each question, we flipped the first child, which was usually the question word (*who, when, where, what, why*), with the rest of the other children, and took the minimum cost of tree matching with or without flipping.

We also tried to use the information from the question classifier and named-entity provide. The statistics from the Expected Answer Filter was used to measure the match between the question type and the expected name entity for that type. Currently, for each question, both the main category and subcategories for its classified type were looked up in the statistics table. This is done so it can get the maximum probability for matching the named entity of the node in the story sentence tree.

The following is the definition of our cost functions:

1. delete:
   - if a is a stop word, $\gamma(a \rightarrow \Lambda) = 5$
   - else $\gamma(a \rightarrow \Lambda) = 200$
2. insert:
   - if a is a stop word, $\gamma(\Lambda \rightarrow a) = 5$
   - else $\gamma(\Lambda \rightarrow a) = 200$
3. change:
   - if a is a question word,

  - if a is "who" and b has named entity "PERSON", $\gamma(a \rightarrow b) = 5$
  - else if a is "where" and b has named entity "LOCATION", $\gamma(a \rightarrow b) = 5$.
  - else if a is "when" and b has named entity "DATE"/"TIME", $\gamma(a \rightarrow b) = 5$
  - else
    * get the maximum probability $p(a, b)$ of seeing expected question class for a and named entity for b
    * $\gamma(a \rightarrow b) = 200 - 100 * p(a, b)$
- else
  - if word a is identical to word b, $\gamma(a \rightarrow b) = 0$
    * if a is the main verb, $\gamma(a \rightarrow b) = -100$
    * else $\gamma(a \rightarrow b) = 1$
  - else if a and b have the same stemmed form
    * if a is the main verb, $\gamma(a \rightarrow b) = -99$
    * else $\gamma(a \rightarrow b) = 1$
  - else $\gamma(a \rightarrow b) = 200$

According to our observation, we noticed that the structure of the sentences in the story is usually simple. In addition, the shorter the sentence is, the less it will take advantage of the approximate tree matching approach. However, generally the main verb of the question is the same as the answer. So we gave extra score for the matching of the main verb between the question and the story sentence, which is implemented as less cost for the *change* operation in the cost function. The experiments on using tree scorer only actually indicated the advantage of this heuristic. For example, for the overall performance on level 3, we achieved 0.34 accuracy without using question type classification and the verb-match heuristic, and climbed to 0.37 and 0.36 with using either question type classification or the verb-match heuristic, got 0.39 finally with using both of them.

5 Evaluation and Results

We have run our system on all four grade levels. We reported on the baseline performance in our previous report [4] with various filters turned off and on.

In this paper, we would like to experiment with different combinations of scorers and how they can affect the overall performance. All the filters described in 3 are turned on for each experiment.

We report how often the program answers a question by choosing a correct sentence as judged in the answer mark-ups. This is known as **HumSent** (human annotated sentence). If a question does not have a correct sentences, we ignore that question. Thus, we do not consider the "no answer" case.

We have experimented with the following scorers:

Scorer ID	Meaning
BASELINE	The baseline case in our previous experiment, where TFIDF is used with stemming, stopwords removal, question specific rules, and the named entity filter
TF	TFIDFScorer
Qu	QuestionSpecificScorer
Pr	PronounScorer
Na	NamedEntityScorer
BO	BOWScorer (Bag of Words)
Ok	OkapiScorer
Ex	ExpectedAnswerScorer
Tr	TreeScorer

For each combination of the scorers, we first train the weights for the composite ranking function using level 2 and 5. we then run the experiment on all 4 levels. The result you see below are the top 10 combinations of scorers on the average score of level 3 and 4 ($Avg34$). That average score is calculated by adding the accuracy of the two levels together and divided by 2. Thus, this number may be different than the number of questions the system gets correctly divided by the total number of questions that we have answered.

The Avg is the average result of all 4 levels, it is calculated by summing the accuracy of the 4 levels and divided by 4.

The $Avg25$ is the average result of level 2 and 5, it is calculated in the same way.

The number (n/m) in each cell indicates the number of questions we get right (n) over the number of questions we answered (m). Our system answers all the questions.

The baseline performance is given below each of the table. In some cases, the baseline configuration actually is one of the top 10 configuration. In such case, it is shown twice.

Here are the overall results:

Scorers	**level2**	**level3**	**level4**	**level5**	**Avg**	**Avg25**	**Avg34**
TFQuNaBOEx	53.08% (69/130)	42.22% (57/135)	42.31% (55/130)	58.46% (38/65)	49.02% (219/460)	55.77% (107/195)	42.26% (112/265)
TFQuPrNaBOOkExTr	51.54% (67/130)	42.22% (57/135)	42.31% (55/130)	60.00% (39/65)	49.02% (218/460)	55.77% (106/195)	42.26% (112/265)
TFQuBOEx	52.31% (68/130)	42.22% (57/135)	40.77% (53/130)	58.46% (38/65)	48.44% (216/460)	55.38% (106/195)	41.50% (110/265)
TFNaBO	48.46% (63/130)	44.44% (60/135)	37.69% (49/130)	49.23% (32/65)	44.96% (204/460)	48.85% (95/195)	41.07% (109/265)
BOOkEx	48.46% (63/130)	40.74% (55/135)	40.77% (53/130)	53.85% (35/65)	45.95% (206/460)	51.15% (98/195)	40.75% (108/265)
QuOkEx	48.46% (63/130)	40.74% (55/135)	40.77% (53/130)	53.85% (35/65)	45.95% (206/460)	51.15% (98/195)	40.75% (108/265)
TFBOEx	52.31% (68/130)	42.22% (57/135)	39.23% (51/130)	55.38% (36/65)	47.29% (212/460)	53.85% (104/195)	40.73% (108/265)
QuBOOkEx	48.46% (63/130)	42.96% (58/135)	38.46% (50/130)	55.38% (36/65)	46.32% (207/460)	51.92% (99/195)	40.71% (108/265)
TFQuBOOkEx	48.46% (63/130)	42.96% (58/135)	38.46% (50/130)	55.38% (36/65)	46.32% (207/460)	51.92% (99/195)	40.71% (108/265)
TFBO	47.69% (62/130)	44.44% (60/135)	36.92% (48/130)	49.23% (32/65)	44.57% (202/460)	48.46% (94/195)	40.68% (108/265)
BASELINE	40.77% (53/130)	39.26% (53/135)	39.23% (51/130)	47.69% (31/65)	41.74% (188/460)	44.23% (84/195)	39.25% (104/265)

For the WHAT question type:

Scorers	**level2**	**level3**	**level4**	**level5**	**Avg**	**Avg25**	**Avg34**
TFPr	50.00% (12/24)	37.93% (11/29)	37.04% (10/27)	46.15% (6/13)	42.78% (39/93)	48.08% (18/37)	37.48% (21/56)
TFPrNa	45.83% (11/24)	37.93% (11/29)	37.04% (10/27)	46.15% (6/13)	41.74% (38/93)	45.99% (17/37)	37.48% (21/56)
PrNaOk	45.83% (11/24)	41.38% (12/29)	33.33% (9/27)	46.15% (6/13)	41.67% (38/93)	45.99% (17/37)	37.36% (21/56)
TFPrNaOk	45.83% (11/24)	41.38% (12/29)	33.33% (9/27)	46.15% (6/13)	41.67% (38/93)	45.99% (17/37)	37.36% (21/56)
TFPrBO	54.17% (13/24)	37.93% (11/29)	33.33% (9/27)	53.85% (7/13)	44.82% (40/93)	54.01% (20/37)	35.63% (20/56)
TFPrNaBO	54.17% (13/24)	37.93% (11/29)	33.33% (9/27)	53.85% (7/13)	44.82% (40/93)	54.01% (20/37)	35.63% (20/56)
TFQuPr	50.00% (12/24)	37.93% (11/29)	33.33% (9/27)	53.85% (7/13)	43.78% (39/93)	51.92% (19/37)	35.63% (20/56)
TFQuPrNa	50.00% (12/24)	37.93% (11/29)	33.33% (9/27)	53.85% (7/13)	43.78% (39/93)	51.92% (19/37)	35.63% (20/56)
PrBOOk	54.17% (13/24)	41.38% (12/29)	29.63% (8/27)	53.85% (7/13)	44.76% (40/93)	54.01% (20/37)	35.50% (20/56)
PrNaBOOk	54.17% (13/24)	41.38% (12/29)	29.63% (8/27)	53.85% (7/13)	44.76% (40/93)	54.01% (20/37)	35.50% (20/56)
BASELINE	54.17% (13/24)	34.48% (10/29)	29.63% (8/27)	61.54% (8/13)	44.95% (39/93)	57.85% (21/37)	32.06% (18/56)

For the WHERE question type:

Scorers	**level2**	**level3**	**level4**	**level5**	**Avg**	**Avg25**	**Avg34**
QuBOOkEx	37.04% (10/27)	71.43% (20/28)	48.00% (12/25)	66.67% (10/15)	55.78% (52/95)	51.85% (20/42)	59.71% (32/53)
TFQuBOOkEx	37.04% (10/27)	71.43% (20/28)	48.00% (12/25)	66.67% (10/15)	55.78% (52/95)	51.85% (20/42)	59.71% (32/53)
QuBOOk	40.74% (11/27)	71.43% (20/28)	44.00% (11/25)	66.67% (10/15)	55.71% (52/95)	53.70% (21/42)	57.71% (31/53)
TFQuBO	37.04% (10/27)	71.43% (20/28)	44.00% (11/25)	60.00% (9/15)	53.12% (50/95)	48.52% (19/42)	57.71% (31/53)
TFQuNaBO	37.04% (10/27)	71.43% (20/28)	44.00% (11/25)	60.00% (9/15)	53.12% (50/95)	48.52% (19/42)	57.71% (31/53)
TFQuPrNaBOOkExTr	40.74% (11/27)	64.29% (18/28)	48.00% (12/25)	66.67% (10/15)	54.92% (51/95)	53.70% (21/42)	56.14% (30/53)
BOOkEx	40.74% (11/27)	67.86% (19/28)	44.00% (11/25)	60.00% (9/15)	53.15% (50/95)	50.37% (20/42)	55.93% (30/53)
QuNaBOOkEx	37.04% (10/27)	67.86% (19/28)	44.00% (11/25)	66.67% (10/15)	53.89% (50/95)	51.85% (20/42)	55.93% (30/53)
QuOkEx	40.74% (11/27)	67.86% (19/28)	44.00% (11/25)	60.00% (9/15)	53.15% (50/95)	50.37% (20/42)	55.93% (30/53)
TFBOOkEx	40.74% (11/27)	67.86% (19/28)	44.00% (11/25)	66.67% (10/15)	54.82% (51/95)	53.70% (21/42)	55.93% (30/53)
BASELINE	22.22% (6/27)	32.14% (9/28)	32.00% (8/25)	46.67% (7/15)	33.26% (30/95)	34.44% (13/42)	32.07% (17/53)

For the WHEN question type:

Scorers	**level2**	**level3**	**level4**	**level5**	**Avg**	**Avg25**	**Avg34**
BOOkEx	71.43% (20/28)	51.85% (14/27)	69.23% (18/26)	46.15% (6/13)	59.67% (58/94)	58.79% (26/41)	60.54% (32/53)
QuOkEx	71.43% (20/28)	51.85% (14/27)	69.23% (18/26)	46.15% (6/13)	59.67% (58/94)	58.79% (26/41)	60.54% (32/53)
TFBOEx	75.00% (21/28)	55.56% (15/27)	65.38% (17/26)	53.85% (7/13)	62.45% (60/94)	64.42% (28/41)	60.47% (32/53)
TFNaBOEx	75.00% (21/28)	55.56% (15/27)	65.38% (17/26)	53.85% (7/13)	62.45% (60/94)	64.42% (28/41)	60.47% (32/53)
TFQuEx	75.00% (21/28)	55.56% (15/27)	65.38% (17/26)	53.85% (7/13)	62.45% (60/94)	64.42% (28/41)	60.47% (32/53)
TFQuNaEx	75.00% (21/28)	55.56% (15/27)	65.38% (17/26)	53.85% (7/13)	62.45% (60/94)	64.42% (28/41)	60.47% (32/53)
NaBOOkEx	71.43% (20/28)	48.15% (13/27)	69.23% (18/26)	46.15% (6/13)	58.74% (57/94)	58.79% (26/41)	58.69% (31/53)
QuNaOkEx	71.43% (20/28)	48.15% (13/27)	69.23% (18/26)	46.15% (6/13)	58.74% (57/94)	58.79% (26/41)	58.69% (31/53)
TFNaBOOkEx	71.43% (20/28)	48.15% (13/27)	69.23% (18/26)	46.15% (6/13)	58.74% (57/94)	58.79% (26/41)	58.69% (31/53)
TFQuNaOkEx	71.43% (20/28)	48.15% (13/27)	69.23% (18/26)	46.15% (6/13)	58.74% (57/94)	58.79% (26/41)	58.69% (31/53)
BASELINE	67.86% (19/28)	59.26% (16/27)	53.85% (14/26)	53.85% (7/13)	58.70% (56/94)	60.85% (26/41)	56.55% (30/53)

For the WHY question type:

Scorers	level2	level3	level4	level5	Avg	Avg25	Avg34
TFNaBOEx	33.33% (8/24)	18.18% (4/22)	34.62% (9/26)	27.27% (3/11)	28.35% (24/83)	30.30% (11/35)	26.40% (13/48)
TFQuNaBOEx	37.50% (9/24)	18.18% (4/22)	34.62% (9/26)	27.27% (3/11)	29.39% (25/83)	32.39% (12/35)	26.40% (13/48)
TFQuPrNaBOOkExTr	33.33% (8/24)	22.73% (5/22)	26.92% (7/26)	27.27% (3/11)	27.56% (23/83)	30.30% (11/35)	24.83% (12/48)
QuNaBOOkEx	37.50% (9/24)	18.18% (4/22)	30.77% (8/26)	27.27% (3/11)	28.43% (24/83)	32.39% (12/35)	24.48% (12/48)
TFBOEx	33.33% (8/24)	18.18% (4/22)	30.77% (8/26)	27.27% (3/11)	27.39% (23/83)	30.30% (11/35)	24.48% (12/48)
TFNaBO	37.50% (9/24)	18.18% (4/22)	30.77% (8/26)	27.27% (3/11)	28.43% (24/83)	32.39% (12/35)	24.48% (12/48)
TFQuBOEx	37.50% (9/24)	18.18% (4/22)	30.77% (8/26)	27.27% (3/11)	28.43% (24/83)	32.39% (12/35)	24.48% (12/48)
TFNaEx	29.17% (7/24)	13.64% (3/22)	34.62% (9/26)	27.27% (3/11)	26.17% (22/83)	28.22% (10/35)	24.13% (12/48)
TFQuNaEx	29.17% (7/24)	13.64% (3/22)	34.62% (9/26)	27.27% (3/11)	26.17% (22/83)	28.22% (10/35)	24.13% (12/48)
BASELINE	25.00% (6/24)	22.73% (5/22)	23.08% (6/26)	18.18% (2/11)	22.25% (19/83)	21.59% (8/35)	22.90% (11/48)
BASELINE	25.00% (6/24)	22.73% (5/22)	23.08% (6/26)	18.18% (2/11)	22.25% (19/83)	21.59% (8/35)	22.90% (11/48)

For the WHO question type:

Scorers	level2	level3	level4	level5	Avg	Avg25	Avg34
QuNaEx	25.93% (7/27)	44.83% (13/29)	57.69% (15/26)	61.54% (8/13)	47.50% (43/95)	43.73% (15/40)	51.26% (28/55)
BASELINE	33.33% (9/27)	44.83% (13/29)	57.69% (15/26)	53.85% (7/13)	47.42% (44/95)	43.59% (16/40)	51.26% (28/55)
NaBOEx	29.63% (8/27)	48.28% (14/29)	53.85% (14/26)	69.23% (9/13)	50.25% (45/95)	49.43% (17/40)	51.06% (28/55)
TFQu	37.04% (10/27)	55.17% (16/29)	42.31% (11/26)	46.15% (6/13)	45.17% (43/95)	41.60% (16/40)	48.74% (27/55)
TFQuNa	37.04% (10/27)	55.17% (16/29)	42.31% (11/26)	46.15% (6/13)	45.17% (43/95)	41.60% (16/40)	48.74% (27/55)
NaBO	29.63% (8/27)	37.93% (11/29)	57.69% (15/26)	76.92% (10/13)	50.54% (44/95)	53.28% (18/40)	47.81% (26/55)
QuEx	29.63% (8/27)	37.93% (11/29)	57.69% (15/26)	53.85% (7/13)	44.77% (41/95)	41.74% (15/40)	47.81% (26/55)
BOEx	33.33% (9/27)	41.38% (12/29)	53.85% (14/26)	61.54% (8/13)	47.52% (43/95)	47.44% (17/40)	47.61% (26/55)
QuNaBO	40.74% (11/27)	48.28% (14/29)	46.15% (12/26)	76.92% (10/13)	53.02% (47/95)	58.83% (21/40)	47.21% (26/55)
BOOk	40.74% (11/27)	51.72% (15/29)	42.31% (11/26)	46.15% (6/13)	45.23% (43/95)	43.45% (17/40)	47.02% (26/55)
BASELINE	33.33% (9/27)	44.83% (13/29)	57.69% (15/26)	53.85% (7/13)	47.42% (44/95)	43.59% (16/40)	51.26% (28/55)

The best overall performance on the testing data is 42.26%. This result is produced by two seperate scorers, which also tie for the top overall score on the training data, 55.77%. It is interesting to see how, although the implementation including all 8 scorers ties for the top spot, many of the

other top results come from combinations of just 3 or 4 scorers. With the very limited amount of training data available, this could imply that implementations involving a larger set of scorers brought with them a hypothesis space far too large, and the resulting scorer was overfit to the training data.

The best results we see come on the WHEN and WHERE questions. Results for these types hovers around 60%. The inclusion of named entity data regarding these types is likely a source of success. Not surprisingly, the worst performance comes on the abstract WHY questions, scoring about 26%. We also see that the top scorers for each question types vary quite a bit. This could imply that modifying our system to learn separate scorers for each question type could have potentially improved our results quite a bit. Unfortunately time constraints prohibited us from exploring this option.

In general the expected answer modules improve our performance quite a bit, especially on WHEN and WHERE questions, as the direct mapping from DATE and LOCATION expected answer data to their corresponding named entity data makes for easy recognition of likely answers. Also, it is worth noting that the domain of expertise of the expected answer scorer appears to overlap quite a bit with the question-specific rules scorer. This can be seen from the fact that adding either of these scorers to just the Bag of Words Scorer yields an improvement of several points (3-5%). However, adding either the question specific scorer or expected answer scorer to a system already utilizing the other yields very little improvement.

Also interesting is the fact that in comparing an implementation that includes the expected answer scorer to one that does not, even if they score roughly the same on levels 3 and 4, the implementation with Expected Answer data generally performs several points better on levels 2 and 5. This is likely due to the fact that statistics regarding conditional probabilities of named entity and question classification data are taken over levels 2 and 5. As the weights for the composite scorer are also learned based on sets 2 and 5, the composite scorer is likely assigning larger weights to the expected answer scorer than is deserved, and as a result the system is overfit on levels 2 and 5. This problem could likely be avoided by splitting the data into three sets, a training set from which conditional and unconditional probabilities are learned by the expected answer scorer, a developmental set used by the composite scorer to train the scorer weights, and a separate training set. Unfortunately, our training set is quite limited, and supplying each of the expected answer and composite scorer modules with even less data would likely lead to significantly degraded performance.

It is not clear how the tree scorer actually helps the overall system when we combine it with several scorers. However, when we only use the tree scorer, it shows better performance on higher levels, especially for level 5.

Flag(s)	**level2**	**level3**	**level4**	**level5**	**Average**
org	0.346153846	0.340740741	0.353846154	0.446153846	0.371723647
org+verb	0.353846154	0.37037037	0.353846154	0.507692308	0.396438746
org+NE	0.330769231	0.362962963	0.353846154	0.461538462	0.377279202
org+verb+NE	0.361538462	0.392592593	0.353846154	0.492307692	0.400071225

verb: verbal match heuristic, **NE**: expected question type and named entity match

Since the Approximate Tree Matching approach tries to match a question only with parts of the story sentence, if the sentence is very short and the dependency tree is simple, this approach will have almost the same ability as the bag-of-words approach on measuring the similarity between the question and the story sentence. This might be the main reason why the Tree Matching approach did not get as much improvement on the whole system as we expected. Another thing affected the best performance of the tree scorer might be the accuracy of the question classification. Looking at the staticstics for expected answer types, parts of them do not seem reasonable. For example,

for "where" question (7:74), the probability of seeing "DATE" is 0.56, but 0.45 for "Loc", and for most subcategories of "Loc", it is less than 0.1. If we could have more accurate date for expected answer type, the tree scorer might have better performance.

6 Conclusion

We have built upon our question answering system from phase 1, yielding better performance. We have added an expected answer module to score sentence, question pairs based on their associated named entity and question classification data. We have implemented a Tree Scorer to assign points based on the structure of the question and answer sentences. Finally, we have designed a composite scorer function, learning how much weight to associate to each individual scorer.

These modifications have improved our performance on the testing set by approximately 3%, increasing from 39.25% to 42.27%. On the entire data set, performance has improved by about 8%, reaching 49.02%. Unfortunately, our final results are still considerably lower than [2]. Analysis indicates that potential means by which we could improve our results further would be to (a) use a larger training set, possibly split into training and development sections, and (b) learn different composite scorers for each question type.

7 Division of Work

William Lee set up the basic architecture for the story comprehension system, wrote the the Stemming and Stopword filter, implemented the BOW, TF/IDF, and Okapi scoring modules, wrote the trained composite ranking function, and created reporting scripts to combine the data for evaluation.

Geoffrey Levine and Benjamin Liebald worked on data preprocessing, statistics gathering, and the Expected Answer modules.

Yoonkyong Lee implemented the Evaluation module. Xu Ling implemented the semantic classes generation module by looking up WordNet. Then, both Xu and Yoonkyong implemented the tree mapping algorithm.

References

[1] Eugene Charniak, Yasemin Altun, and Rodrigo de Salvo Braz. Reading comprehension programs in a statistical-language-processing class.

[2] Sanda M. Harabagiu, Steven J. Maiorano, and Marius A. Pasca. Open-domain textual question answering techniques. *Natural Language Engineering*, 1(1-38), 2003.

[3] Lynette Hirschman, Marc Light, Eric Breck, and John D. Burger. Deep read: A reading comprehension system.

[4] William Lee, Geoffrey Levine, Xu Ling, Yoonkyong Lee, Benjamin Liebald, and Emily Coogan. Cs 598dnr project 2 (team 4): A baseline story comprehension system.

[5] M. F. Porter. An algorithm for suffix stripping. pages 313–316, 1997.

[6] Vasin Punyakanok, Dan Roth, and Wen tau Yih. Natural language inference via dependency tree mapping: An application to question answering.

[7] Vasin Punyakanok, Dan Roth, and Wen tau Yih. Natural language inference via dependency tree mapping: An application to question answering.

[8] Ellen Riloff and Michael Thelen. A rule-based question answering system for reading comprehension tests.

[9] Kuo-Chung Tai. The tree-to-tree correction problem.

[10] Kaizhong Zhang and Dennis Shasha. Simple fast algorithms for the editing distance between trees and related problems. *SIAM*, 18(1245-1262), 1989.