Mapping Dependencies Trees: An Application to Question Answering [*]

Vasin Punyakanok Dan Roth Wen-tau Yih
Department of Computer Science
University of Illinois at Urbana-Champaign
{punyakan, danr, yih}@cs.uiuc.edu

Abstract

We describe an approach for answer selection in a free form question answering task. In order to go beyond the key-word based matching in selecting answers to questions, one would like to incorporate both syntactic and semantic information in the question answering process. We achieve this goal by representing both questions and candidate passages using dependency trees, and incorporating semantic information such as named entities in this representation. The sentence that best answers a question is determined to be the one that minimizes the generalized edit distance between it and the question tree, computed via an approximate tree matching algorithm. We evaluate the approach on question-answer pairs taken from previous TREC Q/A competitions. Preliminary experiments show its potential by significantly outperforming common bag-of-word scoring methods.

1 Introduction

Open-domain natural language question answering (Q/A) is a challenging task in natural language processing which has received significant attention in the last few years [11, 12, 13]. In the Text REtrieval Conference (TREC) question answering competition, for example, given a free form query like "What was the largest

[*]This research is supported by NSF grants ITR-IIS-0085836, ITR-IIS-0085980 and IIS-9984168 and an ONR MURI Award.

crowd to ever come see Michael Jordan?" [13], the system can access a large collection of newspaper articles in order to find the exact answer, e.g. "62,046", along with a short sentence that supports it being the answer.

The overall tasks is very difficult even for fairly simple question of the type exemplified above. A complete Q/A, requires the ability to 1) analyze questions (question analysis) in order to determine what is the question about [7], 2) retrieve potential candidate answers from the given collection of articles, and 3) determine the final candidate that answers the question. This work concerns with the last stage only. That is, we assume that a set of candidate answers is already given, and we aim at choosing the correct candidate.

We view the problem as that of evaluating the *distance* between a question and each of their answer candidates. The candidate that has the lowest distance to the question is selected as the final answer. The simple bag-of-word technique does not perform well in this case as shown in the following example taken from [6].

```
What is the fastest car in the world?
```

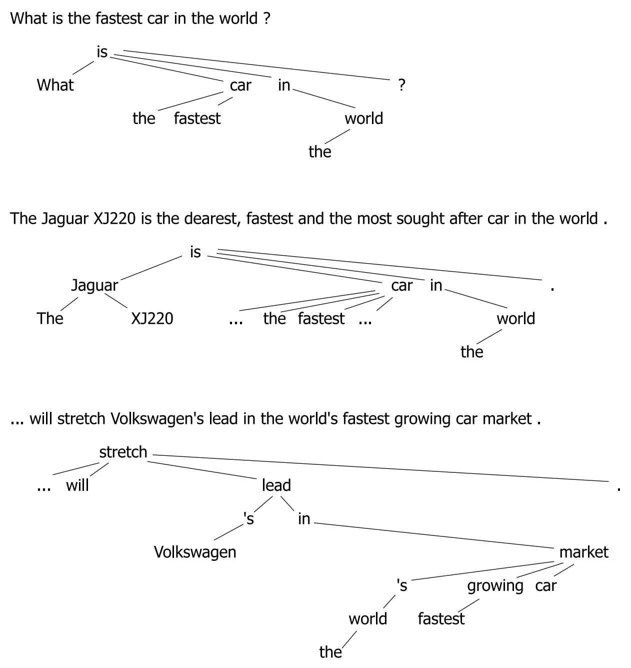
The candidate answers are:

1. `The Jaguar XJ220 is the dearest (415000 pounds), fastest (217mph) and most sought after car in the world.`

2. `...will stretch Volkswagen's lead in the world's fastest growing vehicle market.`

Without deep analysis of the sentences, one would not know that the "fastest" in the second candidate does not modify car as does in the first, and the bag-of-word approach would fail. Therefore, rather than defining distance measure on the raw representation of the sentence, we first represent the question and the answer using a dependency tree. Then we define a distance measure between dependencies trees, taking into account their structure and some semantics properties we infer. Figure 1 shows the dependency trees of the question and the candidate answers in the previous example. This information allows us to better match the question and its correct answer.

Tree matching has recently received attention in natural langauge processing community in the context of machine translation [3, 5, 2], but so far not in the Q/A task. We also presented here a different algorithmic approach from those used in machine translation. Our approach uses the edit distance with the approximate tree matching algorithm [14] to measure the distance between trees.



What is the fastest car in the world ?
is
What
car
in
?
the
fastest
world
the
The Jaguar XJ220 is the dearest, fastest and the most sought after car in the world .
is
Jaguar
car
in
.
The
XJ220
...
the
fastest
...
world
the
... will stretch Volkswagen's lead in the world's fastest growing car market .
stretch
...
will
lead
.
's
in
Volkswagen
market
's
growing
car
world
fastest
the

Figure 1: An example of dependency trees for a question and its candidate answer. Note that due to the comprehensibility we omit some parts of the tree that are irrelevant.

We test our approach on the questions given in the TREC-2002 Q/A track. The comparison between the performance of our approach and a simple bag-of-word approach clearly illustrates the advantage of using dependency trees in this task.

The next section describes our idea of using tree matching over the dependency trees. Then, we explained the edit distance measure and the tree matching method we use. After that we present our experimental results. The conclusion and future direction are given in the final section.

2 Dependency Tree Matching in Question Answering

Our problem concerns with finding the best sentence that contains the answer to any given question. In doing so, we need some mechanism that can measure how close the a candidate answer is to the question. This allows us the choose the final answer which is the one that matches the most closely to the question.

To achieve this, we look at the problem in two levels. First, we need a representation of the sentences that allows us to capture useful information in order to accommodate the matching process. Second, we need an efficient matching process to work on the chosen representation.

At the first level, the representation should be able to capture both the syntactic and semantic information of a sentence. To capture the syntactic information, we represent questions and answers with their dependency trees which allows us to see clearly the syntactic relations between words in the sentences. Using trees also allows us to flexibly incorporate other information including semantic knowledge. By allowing each node in the tree to contain more than just the surface form of its corresponding word, we can add semantic information, e.g. what type of named entities the word belongs, synonyms of the words, or other related words, to the node. Moreover, each node may be generalized to contain a larger unit than a word such as a phrase or a named entity.

With an appropriate representation, the only work left is to find the matching between nodes in the question and the answer in consideration. In doing so, we use the approximate tree matching which we explain in the next section. Formally speaking, we assume for each question q_i, a collection of candidate answers, $A_i = \{a_1, a_2, \ldots, a_{n_i}\}$, each of which is a sentence, is given. We output as the final answer for the q_i,

$$a_i = \arg\min_{a \in A_i} DR(q_i, a),$$

where DR returns the minimum approximate tree matching.

3 Edit Distance and Approximate Tree Matching

We use approximate tree matching [14] in order to decide how similar any given pair of trees are. We first introduce the edit distance [10] which is the distance measure used as the matching criteria. Then, we explain exactly how this measure is used in the approximate tree matching problem.

We recapture here the standard definition that was introduced by [10] and [14]. 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. Edit distance measures the cost of doing a sequence of operations that transforms an ordered labeled tree to another. The operations include deleting a node, inserting a node, and changing a node. Figure 2 illustrate what effect these operations have. Specifically, when a node n is deleted, its children will be attached to the parent of n. Insertion is the inverse of the deletion. Changing a node is to alter its label. Each operation is associated with some cost. A cost of a sequence of operations is the summation of the costs of each operation. We are interested in finding the minimum cost sequence that edits a tree to another.

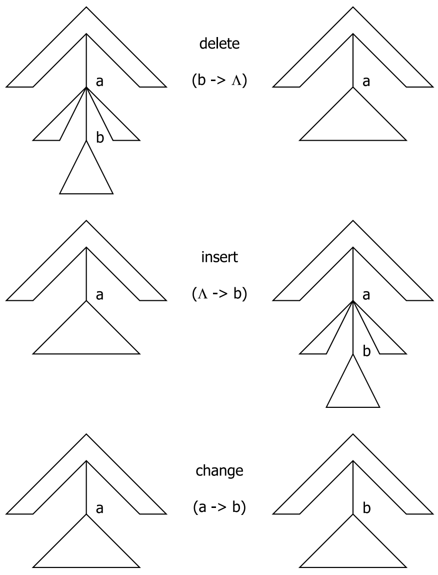


Figure 2: Effect of delete, insert, and change operations

Formally speaking, we represent an operation with a pair (a, b) where a represents the node to be edited and b is its result. We use (a, Λ) and (Λ, b) to represent delete and insert operation respectively. Each operation $(a, b) \neq (\Lambda, \Lambda)$ is associated with a nonnegative cost $\gamma(a \to b)$. The cost of a sequence of operations $S = \langle s_1, s_2, \ldots, s_k \rangle$ is $\gamma(S) = \sum_{i=1}^{k} \gamma(s_i)$. Given a tree T, we denote $s(T)$ as the tree resulting from applying operation s on T, and $S(T) = s_k(s_{k-1}(\ldots(s_1(T))\ldots))$. Given two trees T_1 and T_2, we would like to find

$$\delta(T_1, T_2) = \min\{\gamma(S) | S(T_1) = T_2\}$$

If the cost satisfies triangularity property, that is $\gamma(a \to c) \leqslant \gamma(a \to b) + \gamma(b \to c) \forall a, b, c$, then [10] showed that the minimum cost $\delta(T_1, T_2)$ is a minimum cost of a mapping. A mapping M from T_1 to T_2 is a set of integer pairs satisfying the following properties. Let $T[i]$ represent ith node of the tree T in any given order, N_1 and N_2 be the numbers of nodes in T_1 and T_2 respectively.

1. For any pair $(i, j) \in M$, $1 \leqslant i \leqslant N_1$ and $1 \leqslant j \leqslant N_2$.
2. For any pairs (i_1, j_1) and $(i_2, j_2) \in M$,
   (a) $i_1 = i_2$ if and only if $j1 = j_2$,
   (b) $T_1[i_1]$ is to the left of $T_1[i_2]$ if and only if $T_2[j_1]$ is to the left of $T_2[j_2]$,
   (c) $T_1[i_1]$ is to an ancestor of $T_1[i_2]$ if and only if $T_2[j_1]$ is an ancestor of $T_2[j_2]$.

The cost of a mapping M is

$$\gamma(M) = \sum_{(i,j)\in M} \gamma(T_1[i] \rightarrow T_2[j]) + \sum_{(i,j)\in I} \gamma(T_1[i] \rightarrow \Lambda) + \sum_{(i,j)\in J} \gamma(\Lambda \rightarrow T_2[j]),$$

where I is the set of index of nodes in T_1 that is not mapped by M and J is that of nodes in T_2.

In general, we can use edit distance to decide how similar any give pair of trees are. However, in matching question and answer sentences in the question answering domain, an exact answer to a question may reside only as a clause or a phrase in a sentence, not the whole sentence itself. Therefore, matching the question with the whole candidate sentence may result in poor match even though the sentence contain the correct answer. Approximate tree matching allows us to match question with only some parts of the sentence not a whole. Specifically, there is no additional cost if some subtrees of the answer are deleted.

Formally speaking, Let T_1 and T_2 be two trees to match. A forest S of a tree T is a set of subtrees in T such that all subtrees in S are disjoint, and $T \backslash S$ is the new tree resulting from cutting all subtrees in S from T. Let $\mathcal{S}(T)$ represent the set of all possible forests of T. The approximate tree matching between T_1 and T_2 is to find:

$$DR(T_1, T_2) = \min_{S \in \mathcal{S}(T_2)} \delta(T_1, T_2 \backslash S)$$

[14] gives an efficient dynamic programming based algorithm to compute the approximate tree matching.

We note here that although the cost functions that we use in our experiments do not satisfy the triangularity property, this does not affect the underlying theories of the algorithm. The property is needed only in the proof of the relation between the minimum distance edit operation sequence and the minimum cost mapping. Since we are directly interested in finding the mapping not the operation sequence, the algorithm correctly works for us.

4 Experiment

We experimented on 500 questions given in TREC-2002 Q/A competition. There were 46 questions that had no correct answer. The correct answers for each question, if any, were given along with the answers returned by all participants after the completion of the competition. We, therefore, built our candidate pool for each question from its correct answers and all answers returned by all participants to the question. In some sense, this made the problem harder for our answer selector. Normally, an answer selection process is evaluated based on the candidate pool built from the correct answer and the output from an information retrieval engine. However, our candidate pool contained those incorrect answers made by other systems; hence, we need to be more precise.

Since sentence structure might be quite different from the question, we reformulated the question in simple statement form using simple heuristics rules. In this transformation, the question word (e.g. *what*, *when*, or *where*) was replaced with a special token `*ANS*`. Below is an example of this transformation.

```
Where is Devil’s Tower?
Devil’s Tower is in *ANS*
```

Each sentence was preprocessed first by a SNoW-based part-of-speech tagger [4]. Then, the automatic full parser [1] was run to produce the parse trees. Since this parser also output the head word of each constituent, we could directly convert the parse trees to their corresponding dependency tree by simply taking the head word as the parent. Moreover, we extracted named-entity information with the named-entity recognizer used in [9]. In addition, for each question, we also ran a question classifier [7] which predicted the type of the answers expected by the question.

After the answer was found, the document id that contained the answer was returned. We counted as correct if the returned document id matched that of the correct answer.

We defined three types of cost functions, namely, delete, insert and change, as shown in Figure 3. The stop word list contained some of very common word that would not be very meaningful, e.g. the article such as “a”, “an”, “the”. The word lemma forms were extracted using WordNet [8].

We compared our approach with a simple bag-of-word strategy. In this simple approach, we measured the similarity between a question and a candidate answer with the number of common words, either in their surface forms or lemma forms,

```
1. delete:
       if $a$ is a stop word, $\gamma(a \to \Lambda)$ = 5,
       else $\gamma(a \to \Lambda)$ = 200.
2. insert:
       if $a$ is a stop word, $\gamma(\Lambda \to a)$ = 200,
       else $\gamma(\Lambda \to a)$ = 5.
3. change:
       if $a$ is *ANS*,
           if $b$ matches the expected answer type, $\gamma(a \to b) = 5$,
           else $\gamma(a \to b) = 200$,
       else
           if word $a$ is identical to word $b$, $\gamma(a \to b) = 0$,
           else if $a$ and $b$ have the same lemma form, $\gamma(a \to b) = 1$,
           else $\gamma(a \to b) = 200$.
```

Figure 3: The definition of cost functions

between the question and the answer divided by the length of that answer. The final answer was the one that produced the highest similarity.

Note that the evaluation method we used here is different from that in TREC-2002 Q/A competition. In TREC, an answer produced by a system consists of the answer key and the document that supports the answer. The answer is considered correct only when both the answer key and the supporting document are correct. Since our system does not provide the answer key, we relax the evaluation of our system by finding only the correct supporting document. However, this does not greatly simplify the task as the harder part of answer selection is to find the correct supporting document. The answer key may be extracted later with some heuristic rules. Also, in practice, a user who uses a Q/A system is very unlikely to believe the system without a correct supporting document. Even though the system does not provide a correct answer key, the user can easily find that given a correct supporting document at hand.

The result is shown in Table 1. It shows the large improvement of using dependency tree over the simple bag-of-word strategy.

Table 1: The comparison of the performance of the approximate tree matching approach and the simple bag-of-word. The last column shows the percentage over only the 454 questions that have an answer.

	Correct		
Method	#	%	%(454)
Tree Matching	183	36.60	40.31
Bag-of-Word	131	26.20	28.85

5 Conclusion

We develop an approach to apply the approximate tree matching algorithm [14] to the Q/A problem. This approach allows us to incorporate dependency trees, a useful syntactic information, in the decision process. In addition, we incorporate some semantic information such as named entity in our approach.

We evaluate our approach on the TREC-2002 questions, and the result clearly illustrate the potential of our approach over the common bag-of-word strategy.

In the future we plan to investigate how to use more semantic information such as synonyms and related words in our approach. Moreover, each node in a tree represents only a word in a sentence, and we believe that by appropriately combining nodes into a meaningful phrase may allow our approach perform better. Finally, we plan to use some learning technique to learn the cost functions which are manually defined now.

References

[1] M. Collins. Three generative, lexicalised models for statistical parsing. In *Proceedings of the 35th Annual Meeting of the Association of Computational Linguistics*, pages 16–23, Madrid, Spain, 1997.

[2] Y. Ding, D. Gildea, and M. Palmer. An algorithm for word-level alignment of parallel dependency trees. In *The 9th Machine Translation Summit of International Association of Machine Translation*, New Orleans, LA, 2003.

[3] J. Eisner. Learning non-isomorphic tree mappings for machine translation. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics (companion volume)*, Supporo, Japan, July 2003.

[4] Y. Even-Zohar and D. Roth. A sequential model for multi-class classification. In *Proceedings of 2001 Conference on Empirical methods in Natural Language Processing*, Pittsburgh, PA, 2001.

[5] D. Gildea. Loosely tree-based alignment for machine translation. In *Proceedings of the 41st Annual Meeting of the Association of Computational Linguistics (ACL-03)*, Supporo, Japan, 2003.

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

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

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

[9] D. Roth, G. K. Kao, X. Li, R. Nagarajan, V. Punyakanok, N. Rizzolo, W-T. Yih, C. Ovesdotter, and L. Moran. Learning components for a question-answering system. In *Proceedings of The Tenth Text REtrieval Conference (TREC 2001)*, Gaithesburg, Maryland, 2001.

[10] K. Tai. The tree-to-tree correction problem. *Journal of the Association for Computing Machinery*, 26(3):422–433, July 1979.

[11] E. Voorhees. Overview of the trec-9 question answering track. In *The Ninth Text Retrieval Conference (TREC-9)*, pages 71–80. NIST SP 500-249, 2000.

[12] E. Voorhees. Overview of the trec 2001 question answering. In *The Tenth Text Retrieval Conference (TREC 2001)*, pages 42–51. NIST SP 500-250, 2001.

[13] E. Voorhees. Overview of the trec 2002 question answering. In *The Eleventh Text Retrieval Conference (TREC 2002)*. NIST SP 500-251, 2002.

[14] K. Zhang and D. Shasha. Simple fast algorithms for the editing distance between trees and related problems. *SIAM Journal on Computing*, 18(6):1245–1262, December 1989.