Variants of tree similarity in a Question Answering task

Martin Emms
Dept of Computer Science
Trinity College
Ireland

Abstract

The results of experiments on the application of a variety of distance measures to a question-answering task are reported. Variants of tree-distance are considered, including whole-vs-sub tree, node weighting, wild cards and lexical emphasis. We derive string-distance as a special case of tree-distance and show that a particular parameterisation of tree-distance outperforms the string-distance measure.

1 Introduction

This paper studies the deployment in a question answering task of methods which assess the similarity of question and answer representations. Given questions such as

Q1 *what does malloc return ?*
Q2 *What year did poet Emily Dickinson die?*

and a collection of sentences (eg. a computer manual, a corpus of newspaper articles), the task is to retrieve the sentences that answer the question, eg.

A1 *the malloc function returns a null pointer*
A2 *In 1886 , poet Emily Dickinson died in Amherst , Mass*

One philosophy for finding answers to questions would be to convert questions and candidate answers into logical forms and to compute answerhood by apply theorem-proving methods. Another philosophy is to assume that the answers are *similar* to the questions, where similarity might be defined in many different ways. While not all answers to all questions will be similar, there's an intuition that most questions can be answered in a way which shares quite a bit with the question, and that accordingly with a large enough corpus, a similarity-based approach could be fruitful.

2 Distance Measures

In pursuing such a similarity-based approach to question-answering, the key decisions to be made are the representations of the questions and answers, and relatedly, distance measures between them.

We will primarily be concerned with measures which refer to a linguistic structure assigned to a word sequence – variants of *tree-distance*, but we will also consider *string-distance*.

2.1 Tree Measures

Following (Zhang and Shasha, 1989), one can arrive at *tree-distance* in the following way. Given source and target ordered, labelled trees, S and T, consider the set $\mathcal{H}(S,T)$ of all 1-to-1 *partial* maps, σ, from S into T, which are *homomorphism*s preserving left-to-right order and ancestry[1]. Let the *alignment*, σ', be the enlargment of the map σ with pairs (S_i, λ) for nodes $S_i \notin dom(\sigma)$ and (λ, T_j) for nodes $T_j \notin ran(\sigma)$. Let $\mathcal{D}$ define *deletion* costs for the (S_i, λ), $\mathcal{I}$ *insertion* costs for the (λ, T_j), and $\mathcal{R}$ *replacement* costs for the (S_i, T_j) which represent nodes with non-identical labels. Then a total cost for the alignment, $\mathcal{C}(\sigma')$ can be defined as the sum of these components costs, and the **tree distance** can then be defined as the cost of the least-cost map:

$$\Delta(S,T) = min(\{\mathcal{C}(\sigma') \mid \sigma \in \mathcal{H}(S,T)\})$$

For any 3 trees, T^1, T^2, T^3, the triangle inequality holds $\Delta(T^1,T^3) \leq \Delta(T^1,T^2) + \Delta(T^2,T^3)$.

[1] If $T_{j_1} = \sigma(S_{i_1})$ and $T_{j_2} = \sigma(S_{i_2})$ then (i) S_{i_1} is to the left of S_{i_2} iff T_{j_1} is to the left of T_{j_2} and (ii) S_{i_1} is a descendant of S_{i_2} iff T_{j_1} is a descendant of T_{j_2}, with descendency understood as the transitive closure of the daugher-mother relation.

Proceedings of the Workshop on Linguistic Distances, pages 100–108,
Sydney, July 2006.

Briefly the argument is as follows. Given mappings $\sigma \in \mathcal{H}(T^1, T^2)$, and $\tau \in \mathcal{H}(T^2, T^3)$, $\sigma \circ \tau \in \mathcal{H}(T^1, T^3)$[2], so $(\sigma \circ \tau)'$ is an alignment between T^1 and T^3, and $\Delta(T^1, T^3) \leq \mathcal{C}((\sigma \circ \tau)')$. The cost of the composition is less than the sum of the costs of the composed maps: σ's insertions and replacements contribute only if they fall in $dom(\tau)$, τ's deletions and replacements contribute only if they act on $ran(\sigma)$.

From this basic definition, one can depart in a number of directions. First of all, there is a **part-vs-whole** dimension of variation. Where $\Delta(S, T)$ gives the cost of aligning the *whole* source tree S with the target T, one can consider variants where one minimises over a set of *sub*-parts of S. This is equivalent to letting all but the nodes belonging to the chosen sub-part to delete at zero cost[3]. Let $\delta(S, T)$ be the **sub-tree** distance. Let $\vec{\delta}(S, T)$, be the **sub-traversal** distance, in which sub-traversals of the left-to-right, post-order traversal of S are considered. As for Δ, the triangle inequality holds for δ and $\vec{\delta}$ – one needs to extend the notion of alignment with a set of free deletions. Unlike Δ, δ and $\vec{\delta}$ are not symmetric.

All of Δ, δ and $\vec{\delta}$ are implicitly parametrised by the cost functions, $\mathcal{D}$, $\mathcal{I}$ and $\mathcal{R}$. In the work below 4 other parameters are explored

Node weighting $\mathcal{W}$: this is a function which assigns a real-number weight to each each node. The cost function then refers to the weights. In experiments reported below, $\mathcal{D}_w((S_i, w), \lambda) = w$, $\mathcal{I}_w(\lambda, (T_j, w)) = w$, $\mathcal{R}_w((S_i, w_s), (T_j, w_t)) = max(w_s, w_t)$, if S_i and T_j have unequal labels. The experiments reported below use 2 weighting function $\mathcal{STR}$, and $\mathcal{LEX}$. $\mathcal{STR}$ assign weights according to the syntactic structure, via a classification of nodes as heads vs. complements vs. adjuncts vs. the rest, with essentially adjuncts given 1/5th the weights of heads and complements, and other daughters 1/2, via essentially the following top-down algorithm:

$Str(node, rank)$:
assign weight $1/rank$ *to* $node$
for each daughter d
if (d *is head or complement)*{
assign weight = $1/rank$,
$Str(rank, d)$ }
else if (d is adjunct) {
assign weight = $1/(5 \times rank)$,
$Str(5 * rank, d)$}
else {
assign weight = $1/(2 \times rank)$
$Str(2 * rank, d)$ }

$\mathcal{LEX}$ is a function which can be composed with $\mathcal{STR}$, and scales up the weights of leaf nodes by a factor of 3.

Target wild cards $T(*)$: this is a function which classifies certain target sub-trees as *wild-card*. If source S_i is mapped to target T_j, and T_j is the root of a wild-card tree, all nodes within the S_i sub-tree can be deleted for 0 cost, and all those within the T_j sub-tree can be inserted for 0 cost. A wild card np tree might can be put in the position of the gap in wh-questions, allowing for example *what is memory allocation*, to closely match any sentences with *memory allocation* as their object, no matter what their subject – see Figure 3.

Source self-effacers S/λ: this is a function which classifies source sub-trees as *self-effacers*. Such trees can be deleted in their entirety for zero cost. If S/λ classifies *all* source sub-trees as self-effacing, then $\Delta(S/\lambda)$ will coincide with notion of 'tree-distance with Cut' given in (Zhang and Shasha, 1989).

Target self-inserters λ/T: this is a function which classifies certain target sub-trees as self-inserters. Such trees can be inserted in their entirety for zero cost. A candidate might be optional adjuncts.[4]

2.2 Sequence Measures

The tree-distance measures work with an elaboration of the original questions and answers. (Levenshtein, 1966) defined the 1 dimensional precursor of tree distance, which works directly on the 2 word sequences for the answer and question. For two sequences, s, t, and vertical (or horizontal) tree encodings $l\_tree(s)$ and $l\_tree(t)$, if

[2] $\forall x \in T_1 \forall z \in T_3((x, z) \in \sigma \circ \tau$ iff $\exists y \in T_2((x, y) \in \sigma, (y, z) \in \tau)$

[3] Note that if one minimises also over sub-parts of the target, you do not get an interesting notion, as the minimum will inevitably involve at most one node of source and target.

[4] Thus a target wild-card is somewhat like a target self-effacer, but one which also licenses the classification of a matched source sub-tree as a being self-effacer.

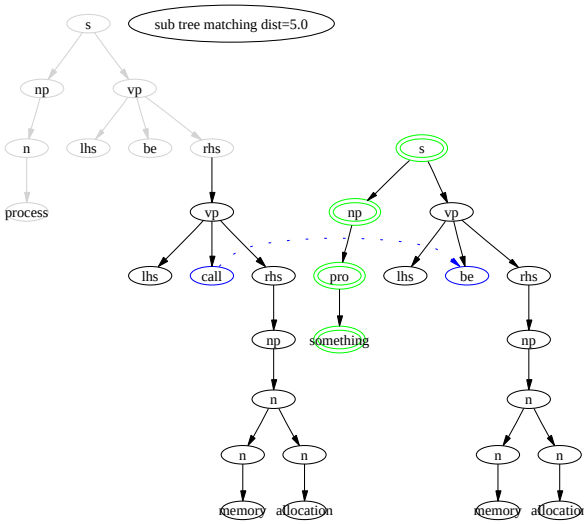
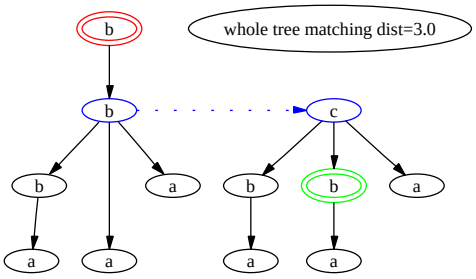


Figure 1: Sub tree example

we define $\Pi(s,t)$, as $\Delta(l\_tree(s), l\_tree(t))$, and $\pi(s,t)$, as $\vec{\delta}(l\_tree(s), l\_tree(t))$, then Π and π coincide with the standard **sequence edit distance** and **sub-sequence edit distance**. As special cases of Δ and δ, Π and π inherit the triangle inequality property.

To illustrate some of the tree-distance definitions, in the following example, a Δ distance of 3 between 2 trees is obtained, assuming unit costs for deletions (shown in red and double outline), insertions (shown in green and double outline), and substitutions (shown in blue and linked with an arrow):



Note also in this picture that nodes that are mapped without a relabelling are shown at the same horizontal level, with no linking arrow.

Figure 1 shows a sub-tree example – δ. The source tree nodes which do not belong to the chosen sub-tree are shown in grey. The lowest vp subtree in the source is selected, and mapped to the vp in the target. The remaining target nodes must be inserted, but this costs less than a match which starts higher and necessitates some deletions and substitutions.

Figure 2 shows a sub-tree example where the structural weighting $\mathcal{STR}$ has been used: size of a node reflects the weight. 4 of the nodes in the source represent the use of an auxiliary verb, and receive low weight, changing the optimum match to one covering the whole source tree. There is some price paid in matching the dissimilar subject nps.

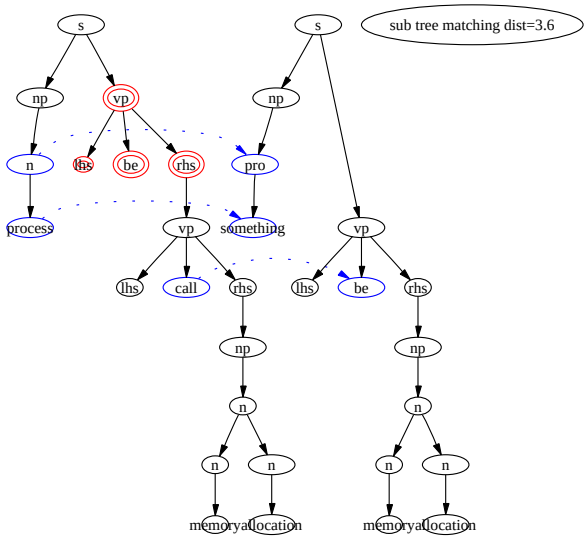


Figure 2: Structurally weighted example

Figure 3 continues the example, but this time in the subject position there is a sub-tree which is classified as a wild-card np tree, and it matches at 0 cost with the subject np in the source tree.

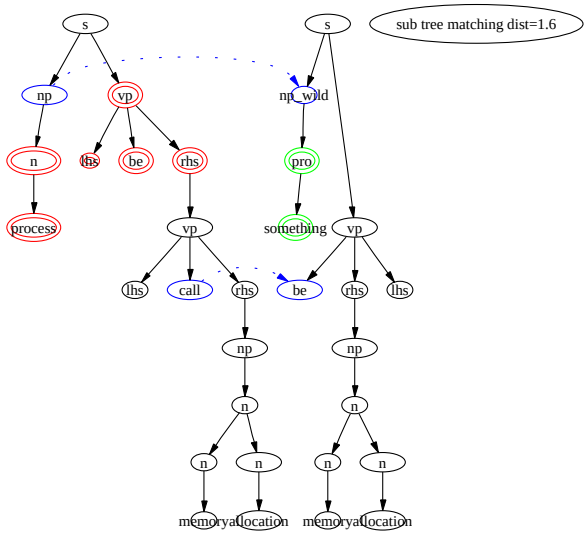


Figure 3: Wild-card example

The basis of the algorithm used to calculate Δ is the *ZhangShasha algorithm* (Zhang and Shasha, 1989): the Appendix summarises it. The im-

plementation is based on code implementing Δ (Fontana et al., 2004), adapting it to allowing for the δ and $\vec{\delta}$ variants and $T(*)$, S/λ, and λ/T parameters, and to generate the human-readable displays of the alignments (such as seen in figures 1,2 and 3).

2.3 Order invariant measures

Assessing answer/question similarity by variants of tree distance or sequence edit-distance, means that distance will not be word-order invariant. There are also measures which are word-order invariant, sometimes called *token-based* measures. These measures are usually couched in a *vector* representation of questions and answers, where vector dimensions are words from (some chosen enumeration) of words (see (Salton and Lesk, 1968)). In the simplest case the values on each dimensions are in $\{0, 1\}$, denoting presence or absence of a word. If $\bullet$ is vector product and a^w is the set of words in a sequence a, then $\vec{a} \bullet \vec{b} = |a^w \cap b^w|$, for the binary vectors representing a^w, b^w. Three well known measures based on this are given below, both in terms vectors, and for binary vectors, the equivalent formulation with sets:

Dice	$2(\vec{a} \bullet \vec{b})/(\vec{a} \bullet \vec{a}) + (\vec{b} \bullet \vec{b}))$
	$= 2(\|a^w \cap b^w\|)/(\|a^w\| + \|b^w\|)$
Jaccard	$(\vec{a} \bullet \vec{b})/(\vec{a} \bullet \vec{a}) + \vec{b} \bullet \vec{b} - \vec{a} \bullet \vec{b})$
	$= (\|a^w \cap b^w\|)/(\|a^w \cup b^w)$
Cosine	$(\vec{a} \bullet \vec{b})/(\vec{a} \bullet \vec{a})^{\cdot 5}(\vec{b} \bullet \vec{b})^{\cdot 5}$
	$= (\|a^w \cap b^w\|)/((\|a^w\|)^{0.5}(\|b^w\|)^{0.5})$

These measure *similarity*, not difference, ranging for 1 for identical a^w,b^w, to 0 for disjoint. In the binary case, Dice/Jaccard similarity can be related to the alignment-based, difference counting perspective of the edit-distances. If we define $\Pi^w(a, b)$ as $|a^w \cup b^w| - |a^w \cap b^w|$ – the size of the *symmetric difference* between a^w and b^w – this can be seen as a set-based version of edit distance[5], which (i) considers mappings on the sets of words, a^w, b^w, not the sequences a, b, and (ii) sets replacement cost to infinity. A difference measure (ranging from 0 for identical a^w,b^w to 1 for disjoint) results if $\Pi^w(a, b)$ is divided by $|a^w| + |b^w|$ (resp. $|a^w \cup b^w|$) and this difference measures will give the reverse of a ranking by Dice (resp. Jaccard) similarity.

The Cosine is a measure of the *angle* between the vectors $\vec{a}, \vec{b}$, and is not relatable in the binary-case to the alignment-based, difference-counting perspective of the edit-distances: dividing $\Pi^w(a, b)$, the symmetric difference, by $|a^w|^{\cdot 5}|b^w|^{\cdot 5}$ does not give a measure with maximum value 1 for the disjoint case, and does not give the reverse of a ranking by Cosine similarity.[6]

Below we shall use θ to denote the Cosine distance.

3 The Question Answering Tasks

For a given representation r (parse trees, word sequences etc.), and distance measure d, we shall generically take a Question Answering by Distance (QAD) task to be given by a set of queries, $\mathcal{Q}$, and for each query q, a corpus of potential answer sentences, $\mathcal{COR}_q$. For each $a \in \mathcal{COR}_q$, the system determines $d(r(a), r(q))$, the distance between the representations of a and q, then uses this to sort $\mathcal{COR}_q$ into $\mathcal{A}_q$. This sorting is then evaluated in the following way. If $a_c \in \mathcal{A}_q$ is the *correct* answer, then the *correct-answer-rank* is the rank of a_c in $\mathcal{A}_q$:

$$| \{a \in \mathcal{A}_q : d(r(a), r(q)) \leq d(r(a_c), r(q))\} |$$

whilst the *correct-answer-cutoff* is the proportion of $\mathcal{A}_q$ cut off by the correct answer a_c:

$$| \{a \in \mathcal{A}_q : d(r(a), r(q)) \leq d(r(a_c), r(q))\} | \ / \ | \mathcal{A}_q |$$

Lower values for this connote better performance. Another figure of merit is the *reciprocal correct-answer-rank. Higher* values of this connote better performance.

Note the notion of answerhood is not one requiring answers to be the sub-sentential phrases associated with wh-phrases in the question. Also not all the questions are wh-questions.

Note also that the set of candidate answers $\mathcal{COR}_q$ is sorted by the answer-to-query distance, $d(r(a), r(q))$, not the query-to-answer distance, $d(r(q), r(a))$. The intuition is that the queries are short and the answers longer, with sub-part that really contains the answer.

The performance of some of the above mentioned distance measures on 2 examples of QAD tasks has been measured:

GNU Library Manual QAD Task: in this case $\mathcal{Q}$ is a set of 88 hand-created

[5] $\Pi^w(a, b)$ could be equivalently defined as $|(\vec{a} - \vec{b})|^2$

[6] if the vectors are normalised by their length, then you can show $|(\vec{a}/|\vec{a}| - \vec{b}/|\vec{b}|)|^2$ reverses the Cosine ranking

queries, and $\mathcal{COR}_q$, shared by all the queries, is the sentences of the manual of the GNU C Library[7] ($|\ \mathcal{COR}_q\ | \approx 31,000$).

The TREC 11 QAD task: In this case $\mathcal{Q}$ was the 500 questions of the TREC11 QA track (Voorhees and Buckland, 2002), whose answers are drawn from a large corpus of newspaper articles. $\mathcal{COR}_q$ was taken to be the sentences of the top 50 from the top-1000 ranking of articles provided by TREC11 for each question ($|\ \mathcal{COR}_q\ | \approx 1000$). Answer correctness was determined using the TREC11 answer regular expressions.

For the tree-distance measures, 2 parsing systems have been used. For convenience of reference, we will call the first parser, the *trinity* parser. This is a home-grown parser combining a disambiguating part-of-speech tagger with a bottom-up chartparser, refering to CFG-like syntax rules and a subcategorisation system somewhat in a categorial grammar style. Right-branching analyses are prefered and a final selection of edges from all available is made using a leftmost/longest selection strategy – there is always an output regardless of whether there is a single input-encompassing edge. Preterminal node labels are a combination of a main functor with other feature terms, but the replacement cost function $\mathcal{R}$ is set to ignore the feature terms. Terminal node labels are base forms of words, not inflected forms. For the structural weighting algorithm, $\mathcal{STR}$, the necessary node distinctions are furnished directly by the parser for vp, and by a small set of structure matching rules for other structures (nps, pps etc). The structures output for wh-questions are essentially deep structures, re-ordering an auxiliary inversion, and placing a tree in the position of a gap.

The Collins parser (Collins, 1999) (*Model 3* variant) is a probabilistic parser, using a model of trees as built top-down with a repertoire of moves, learnt from the Penn Treebank. The preterminal node labels are a combination of a Penn Treebank label with other information pertaining to the head/complement/adjunct distinction, but the replacement cost function $\mathcal{R}$ is set to ignore all but the Penn Treebank part of the label. The terminal node labels are inflected forms of words, not base forms. For the structural weighting algorithm, $\mathcal{STR}$, the necessary node distinctions are furnished directly by the parser. For the question parses, a set of transformations is applied to the parses directly given by the parser, which comparable to the *trinity* parser, re-order auxiliary inversion, and place a tree in the position of a gap.

[7] `http://www.gnu.org`

4 Relating Parse Quality to Retrieval Performance

As a kind of sanity-check on the idea of the using syntactic structures in retrieving answers, we performed some experiments in which we varied the sophistication of the parse trees that the parsers could produce, the expectation being that the less sophisticated the parse, the less successful would be question-answering performance. The left-hand data in Table 1 refers to various reductions of the linguistic knowledge bases of the *trinity* parser(*thin50* = random removal of 50% subset, *manual* = manual removal of a subset, *flat* = entirely flat parses, *gold* = hand-correction of query parses and their correct answers). The right-hand data in Table 1 refers to experiments in which the repertoire of moves available to the Collins parser, as defined by its grammar file, was reduced to different sized random subsets of itself.

Figure 4 shows the empirical cumulative density function (ecdf) of the *correct-answer-cutoff* obtained with the weighted sub-tree with wild cards measure. For each possible value c of *correct-answer-cutoff*, it plots the percentage of queries with a *correct-answer-cutoff* $\leq c$.

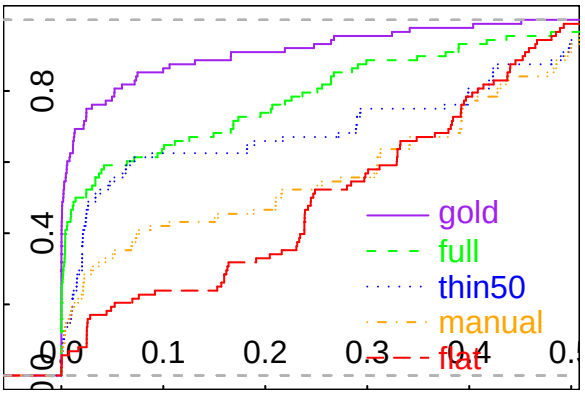


Figure 4: *Success vs Cut-off for different parse settings: x = correct-answer-cutoff, y = proportion of queries whose correct-answer-cutoff $\leq$ x (ranking by weighted sub-tree with wild cards) (Library task)*

What these experiments show is that the ques-

Table 1: *Distribution of* Correct Cutoff *across query set $\mathcal{Q}$ in different parse settings. Left-hand data = GNU task, trinity parser, right-hand data = TREC11 task, Collins parser*

Parsing	1st Qu.	Median	Mean	3rd Qu.
flat	0.1559	0.2459	0.2612	0.3920
manual	0.0215	0.2103	0.2203	0.3926
thin50	0.01418	0.02627	0.157	0.2930
full	0.00389	0.04216	0.1308	0.2198
gold	0.00067	0.0278	0.1087	0.1669

Parsing	1st Qu.	Median	Mean	3rd Qu.
55	0.3157	0.6123	0.5345	0.766400
75	0.02946	0.1634	0.2701	0.4495
85	0.0266	0.1227	0.2501	0.4380
100	0.01256	0.08306	0.2097	0.2901

tion answering performance is a function of the sophistication of the parses that the parsers are able to produce.

5 Comparing Distance Measures

Table 2 gives results on the Library task, using the trinity parser, for some variations of the distance measure.

Considering the results in 2, the best performing measure ($mrr = 0.27$) was the sub-traversal distance, $\vec{\delta}$, assigning weights structurally using $\mathcal{STR}$, with lexical emphasis $\mathcal{LEX}$, and treating a gap position as an np wild card. This slightly out performs the sub-tree measure, δ ($mrr = 0.25$). An alternative approach to discounting parts of the answer tree, allowing any sub-tree of the answer the option to delete for free ($\Delta(\mathcal{W} = Str \circ Lex, T(*) = np\_gap, S/\lambda = \forall)$) performs considerably worse ($mrr = 0.16$). Presumably this is because it is too enthusiastic to assemble the query tree from disparate parts of the answer tree. By comparison, $\vec{\delta}$ and δ can only assembly the query tree from parts of the answer tree that are more closely connected.

The tree-distance measures ($\vec{\delta}$, δ) using structural weights, lexical emphasis and wild cards ($mrr = 0.27$) out-perform the sub-sequence measure, π ($mrr = 0.197$). It also out-performs the cosine measure, θ ($mrr = 0.190$). But π and θ either out-perform or perform at about the same level as the tree-distance measure if the lexical emphasis is removed (see $\delta(\mathcal{W} = Str, T(*) = np\_gap)$, $mrr = 0.160$).

The tree-distance measure δ works better if structural weighting is used ($mrr = 0.09$) than if it is not ($mrr = 0.04$).

The tree-distance measure δ works better with wild-cards (see $\delta(\mathcal{W} = Str, T(*) = np\_gap)$, $mrr = 0.160$, than without (see $\delta(\mathcal{W} = Str)$, $mrr = 0.090$).

Table 3 gives some results on the TREC11 task, using the Collins parser. Fewer comparisons have been made here.

The sub-traversal measure, using structural weighting, lexical emphasis, and wild-cards performs better ($mrr = 0.150$) than the sub-sequence measure ($mrr = 0.09$), which in turn performs better than the basic sub-traversal measure, withouth structural weighting, lexical emphasis or wild-cards ($mrr = 0.076$). The cosine distance, θ, performed best.

6 Discussion

For the parsers used, you could easily have 2 sentences with completely different words, and very different meanings, but which would have the same pre-terminal syntactic structure: the preterminal syntactic structure is not a function of the meaning. Given this, it is perhaps not surprising that there will be cases that the sequence distance easily spots as dissimilar, but which the tree distance measure, without any lexical emphasis, will regard as quite similar, and this perhaps explains why, without any lexical emphasis, the tree-distance measure performs at similar level to, or worse than, the sub-sequence distance measure.

With some kind of lexical emphasis in place, the tree-distance measures out-perform the sub-sequence measures. We can speculate as to the reason for this. There are two kinds of case where the tree-distance measures could be expected to spot a similarity which the sequence-distance measures will fail to spot. One is when the question and answer are more or less similar on their head words, but differ in determiners, auxiliaries and adjuncts. The sequence distance measure will pay more of a price for these differences than the structurally weighted tree-distance. Another kind of case is when the answer supplies words which match a wild-card in the middle of the query tree, as might happen for example in:

> Q: what do child processes inherit from their parent processes
> A: a child process inherits the owner and permissions from the ancestor process

Table 2: *For different distance measures (Library task, trinity parser), distrution of correct-answer-cutoff, mean reciprocal rank* mrr

	cutoff			
distance type	1st Qu.	Median	Mean	mrr
$\vec{\delta}(\mathcal{W} = Str \circ Lex, T(*) = np\_gap)$	8.630-05	8.944-04	2.460-02	0.270
$\delta(\mathcal{W} = Str \circ Lex, T(*) = np\_gap)$	9.414e-05	1.428e-03	7.133e-02	0.255
π bases	1.569e-04	2.087e-03	5.181e-02	0.197
θ bases	1.569e-04	8.630e-04	1.123e-02	0.190
$\Delta(\mathcal{W} = Str \circ Lex, T(*) = np\_gap, S/\lambda = \forall)$	4.080e-04	9.352-03	5.853-02	0.160
$\delta(\mathcal{W} = Str, T(*) = np\_gap)$	3.923e-04	1.964e-02	1.162e-01	0.160
$\delta(\mathcal{W} = Str)$	5.060e-03	3.865e-02	1.303e-01	0.090
δ	1.324e-03	1.046e-01	1.852e-01	0.040
Δ	8.398e-02	2.633e-01	3.531e-01	0.003

Table 3: *For different distance measures (TREC task, collins parser) the distribution of correct-answer-cutoff and mean reciprocal rank (*mrr*)*

	cutoff			
distance type	1st Qu.	Median	Mean	mrr
θ forms	7.847e-03	2.631e-02	1.068e-01	0.167
$\vec{\delta}(\mathcal{W} = Str \circ Lex, T(*) = np\_gap)$	8.452e-03	4.898e-02	1.558e-01	0.150
π forms	2.113e-02	7.309-02	2.051e-01	0.092
$\vec{\delta}$	1.815e-02	1.030e-01	3.269e-01	0.076

The tree-distance measures will see these as similar, but the sub-sequence measure will pay a large price for words in the answer that match the gap position in the query. Thus one can argue that the use of structural weighting, and wild-card trees in the query analysis will tend to equate things which the sequence distance sees as dissimilar.

Another possible reason that the tree-distance measure out-performs the sub-sequence measure is that it may be able to distinguish things which the sequence distance will tend to treat as equivalent. A question might make the thematic role of some entity very clear, but use very few significant words as in:

what does malloc do ?

Using tree distance will favour answer sentences with *malloc* as the subject, such as *malloc returns a null pointer*. The basic problem for the sequence distance here is that it does not have much to work with and will only be able to partition the answer set into a small set of equivalence classes.

These are speculations as to why tree-distance would out-perform sequence distance. Whether these equating and discriminating advantages which theoretically should accrue to δ, $\vec{\delta}$ actually will do so, will depend on the accuracy of the parsing: if there is too much bad parsing, then we will be equating that which we should keep apart, and discriminating that which we should equate.

In the two tasks, the relationship between the tree-distance measures and the order-invariant cosine measure worked out differently. The reasons for this are not clear at the moment. One possibility is that our use of the Collins parser has not yet resulted in good enough parses, especially question parses – recall that the indication from 4 was that improved parse quality will give better retrieval performance. Also it is possible that relative to the queries in the Library task, the amount of word-order permutation between question and answer is greater in the TREC task. This is also indicated by the fact that on the TREC task, the sub-sequence measure, π, falls considerably behind the cosine measure, θ, whereas for the Library task they perform at similar levels.

Some other researchers have also looked at the use of tree-distance measures in semantically-oriented tasks. Punyakonok(2004) report work

using tree-distance to do question-answering on the TREC11 data. Their work differs from that presented here in several ways. They take the parse trees which are output by Collins parser and convert them into dependency trees between the leaves. They compute the distance from query to the answer, rather than from answer to query, using essentially the variant of tree-distance that allows arbitrary sub-trees of the target to insert for zero-cost. Presumably this directionality difference is not a significant one, and with distances calculated from answers to queries, this would correspond to the variant that allows arbitrary source sub-trees to delete with zero cost. The cost functions are parameterised to refer in the case of wild-card replacements to (i) information derived from Named Entity recognisers so different kinds of wh wild-cards can be given low-cost replacment with vocabulary categorised as belong to the right kind by NE recognition and (ii) base-form information.

There is no way to make a numerical comparison because they took a different answer corpus $\mathcal{COR}_q$ – the articles containing the answers suggested by TREC11 participants – and a different criterion of correctness – an answer was correct if it belonged to an article which the TREC11 adjudicators judges to contain a correct answer.

Their adaptation of cost functions to refer to essentially semantic annotations of tree nodes is an avenue we intend to explore in future work. What this paper has sought to do is to investigate intrinsic syntactic parameters that might influence performance. The hope is that these parameters still play a role in an enriched system.

7 Conclusion and Future Work

For two different parsers, and two different question-answering tasks, we have shown that improved parse quality leads to better performance, and that a tree-distance measure out-performs a sequence distance measure. We have focussed on intrinsic, syntactic properties of parse-trees. It is not realistic to expect that exclusively using tree-distance measures in this rather pure way will give state-of-the-art question-answering performance, but the contribution of this paper is the (start of an) exporation of the syntactic parameters which effect the use of tree-distance in question answering. More work needs to be done in systematically varying the parsers, question-answering tasks, and parametrisations of tree-distance over all the possibilities.

There are many possibilities to be explored involving adapting cost functions to enriched node descriptions. Already mentioned above, is the possibility to involve semantic information in the cost functions. Another avenue is introducing weightings based on corpus-derived statistics, essentially making the distance comparision refer to extrinsic factors. One open question is whether analogously to idf, cost functions for (non-lexical) nodes should depend on tree-bank frequencies.

Another question needing further exploration is the dependency-vs-constituency contrast. Interestingly Punyakonok(2004) themselves speculate:

> *each node in a tree represents only a word in the sentence; we believe that appropriately combining nodes into meaningful phrases may allow our approach to perform better.*

We found working with constituency trees that it was the sub-traversal distance measure that performed best, and it needs to be seen whether this holds also for dependency trees. Also to be explored is the role of structural weighting in a system using dependency trees.

A final speculation that it would be interesting to explore is whether one can use feed-back from performance on a QATD task as a driver in the machine-learning of probabilities for a parser, in an approach analogous to the use of the language-model in parser training.

References

Michael Collins. 1999. *Head-driven statistical models for natural language parsing*. Ph.D. thesis, University of Pennsylvania.

Walter Fontana, Ivo L. Hofacker, and Peter F. Stadler. 2004. Vienna rna package. `www.tbi.univie.ac.at/~ivo/RNA`.

V. I. Levenshtein. 1966. Binary codes capable of correcting insertions and reversals. *Sov. Phys. Dokl*, 10:707–710.

Vasin Punyakanok, Dan Roth, and Wen tau Yih. 2004. Natural language inference via dependency tree mapping: An application to question answering. *Computational Linguistics*.

G. Salton and M. E. Lesk. 1968. Computer evalation of indexing and text processing. *Journal of the ACM*, 15:8–36, January.

Ellen Voorhees and Lori Buckland, editors. 2002. *The Eleventh Text REtrieval Conference (TREC 2002)*. Department of Commerce, National Institute of Standards and Technology.

K. Zhang and D. Shasha. 1989. Simple fast algorithms for the editing distance between trees and related problems. *SIAM Journal of Computing*, 18:1245–1262.

Appendix

This appendix briefly summarises the algorithm to compute the tree-distance, based on (Zhang and Shasha, 1989) (see Section 2.1 for definition of tree-distance). The algorithm operates on the left-to-right post-order traversals of trees. Given source and target trees S and T, the output is a table $\mathcal{T}$, indexed vertically by the traversal of S and horizontally by the traversal of T, and position $\mathcal{T}[i][j]$ is the tree-distance from the S subtree rooted at i, to the T subtree rooted at j. Thus the bottom righthand corner of the table represents the tree distance between S and T.

If k is the index of a node of the tree, the *left-most leaf*, $l(k)$, is the index of the leaf reached by following the left-branch down. For a given leaf there is a highest node of which it is the left-most leaf. Let such a node be called a *key-root*. Let $KR(T)$ be the sequence of *key-roots* in T. The algorithm is a doubly nested loop ascending throught the key-roots of S and T, in which for each pair of key-roots (i, j), a routine $tree\_dist(i, j)$ updates the $\mathcal{T}$ table.

Suppose i is any node of S. Then for any i_s with $l(i) \leq i_s \leq i$, the subsequence of S from $l(i)$ to i_s can be seen as a *forest* of subtrees of S, denoted $F(l(i), i_s)$. $tree\_dist(i, j)$ creates a table $\mathcal{F}$, indexed vertically from $l(i)$ to i and horizontally from $l(j)$ to j, such that $\mathcal{F}[i_s][j_t]$ represents the distance between the forests $F(l(i), i_s)$ and $F(l(j), j_t)$. Also the F table should be seen as having an extra left-most column, representing for each i_s, $l(i) \leq i_s \leq i$, the $F(l(i), i_s)$ to $\emptyset$ mapping (pure deletion), and an extra uppermost row representing for each for each j_t, $l(j) \leq j_t \leq j$, the $\emptyset$ to $F(l(j), j_t)$ mapping (pure insertion).

$tree\_dist(i, j)\{$

initialize:

$\mathcal{F}[l(i)][\emptyset], \ldots, \mathcal{F}[i][\emptyset] = 1, \ldots, i - l(i) + 1$

$\mathcal{F}[\emptyset][l(j)], \ldots, \mathcal{F}[\emptyset][j] = 1, \ldots, j - l(j) + 1$

loop: $\forall i_s, l(i) \leq i_s \leq i, \forall j_t, l(j) \leq j_t \leq j$

{

case 1: $l(i_s) = l(i)$ and $l(j_t) = l(j)$

$\mathcal{T}[i_s][j_t] = \mathcal{F}[i_s][j_t]$ = min of $swap$, $delete$, $insert$, where

$swap$ = $\mathcal{F}[i_s - 1][j_t - 1]$ + $swap(i_s, j_t)$

$delete$ = $\mathcal{F}[i_s - 1][j_t]$ + $delete(i_s)$

$insert$ = $\mathcal{F}[i_s][j_t - 1]$ + $insert(j_t)$

case 2: either $l(i_s) \neq l(i)$ or $l(j_t) \neq l(j)$

$\mathcal{F}[i_s][j_t]$ = min of $delete$, $insert$, $for + tree$, where

$swap$, $delete$, $insert$ as before and

$for + tree$ = $\mathcal{F}[l(i_s) - 1][l(j_t) - 1]$ + $\mathcal{T}[i_s][j_t]$

}

}

In case 1, the 'forests' $F(l(i), i_s)$ and $F(l(j), j_t)$ are both single trees and the computed forest distance is transferred to the tree-distance table $\mathcal{T}$. In case 2, at least one of $F(l(i), i_s)$ or $F(l(j), j_t)$ represents a forest of more than one tree. This means there is the possibility that the final trees in the two forests are mapped to each other. This quantity is found from the $\mathcal{T}$ table.

This formulation gives the *whole-tree* distance between S and T. For the *sub-tree* distance, you take the minimum of the final column of $\mathcal{T}$. For the *sub-traversal* case, you do the same but on the final iteration, you set the pure deletion column of $\mathcal{F}$ to all 0s, and take the minimum of the final column of $\mathcal{F}$.

To accommodate wild-card target trees, **case 1** in the above is extended to allow $\mathcal{T}[i_s][j_t] = \mathcal{F}[i_s][j_t] = 0$ in case j_t is the root of a wild-card tree. To accommodate self-effacing source trees, **case 2** in the above is extended to also consider $for + tree\_del$ = $\mathcal{F}[l(i_s) - 1, j_t]$.