Adapting Tree Distance to Answer Retrieval and Parser Evaluation

Martin Emms

Dept of Computer Science, Trinity College, Dublin, Ireland

Abstract

The results of experiments on the application of tree-distance to an answer-retrieval task are reported. Various parameters in the definitions of tree-distance are considered, including whole-vs-sub tree, node weighting, wild cards and lexical emphasis. The results show that improving parse-quality maps to improved performance on this tree-distance answer-retrieval task. It also shown that one of the parametrisations of tree-distance performs better than the more familiar string-distance measure.

keywords: *tree-distance, question-answering, retrieval*

1. Introduction

This concern of this paper is the use of *tree-distance* in an answer retrieval task. Given a question eg.

> *what does malloc return ?*

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

> *the malloc function returns a null pointer*

The paper presents results on this answer retrieval task in which the strategy adopted is to compare syntactic structures assigned to the question and the potential answers, to see how much editing of the answer structure (deletion,insertion, substitution) is needed to derive the question structure. The less editing the answer structure needs, the lower its *tree-distance* from the question structure is deemed to be, and the higher it is rated as an answer to the question.

Two results are presented. Structures are assigned to the questions and potential answers automatically; they are not hand-crafted. The first result is that improving the structures which the parser generates does improve performance on the answer retrieval task. This is not a given, as one could be very sceptical that similarity of syntactic structure could be a guide to answerhood. Also this has significance if seen the other way around: performance on the answer retrieval task, using tree-distance, has potential for use as an evaluation metric for parsers. The second result concerns a comparison of *tree-distance* with the better known *string-distance*, which can also be used to quantify the amount of editing (deletion, insertion, substitution) needed to transform the answer into the question. If string distance is used, the answers and questions are represented simply by word sequences and not by tree structures. We show that with a particular choice of how to compute tree-distance, tree-distance does out-perform string-distance.

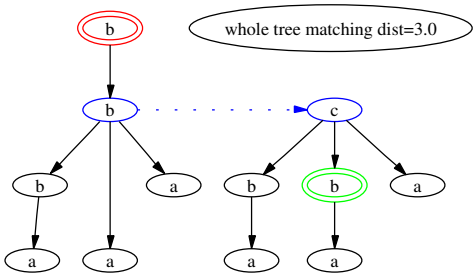
2. Tree Distance

The *tree distance* [1] between two trees can be defined by considering all the possible 1-to-1 partial maps, σ, between source and target trees S and T, which preserve left to right order and ancestry: if S_{i_1} and S_{i_2} are mapped to T_{j_1} and T_{j_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}.

Nodes of S which are not in the domain of σ are considered *deleted*, with an associated

cost. Nodes of T which are not in the range of σ are considered *inserted*, with an associated cost. Otherwise, where $T_j = \sigma(S_i)$, and $T_j \neq S_i$, there is a *substitution* cost. The least cost belonging to a possible mapping between the trees defines the tree distance.

In the following example, a 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):



Nodes which are mapped unaltered are displayed at the same height but with no linking arrow.

In the work reported in this paper the definition of tree distance is varied along a number of dimensions.

Sub-tree: in this variant, the *sub*-tree distance is the cost of the least cost mapping from a sub-tree of the source. Figure 1 shows an example. The source tree nodes which do not belong to the chosen sub-tree are shown in grey. The lowest vp sub-tree in the source is selected, and mapped to the vp in the target. This leaves the remaining target nodes to be inserted, but this costs less than starting the match higher in the source and doing some deletions and substitutions.

Sub-traversal: the least cost mapping from a sub-traversal of the left-to-right post-order traversal of the source.

Structural weights: the outcome in Figure 1 is not completely intuitive, because 4 of the nodes in the source represent the use of an auxiliary verb, which should be negligible. In the weighted version nodes have a weight between 0 and 1, assigned according to the syntactic structure. In Figure 2, the nodes associated with the auxiliary have a low weight, changing

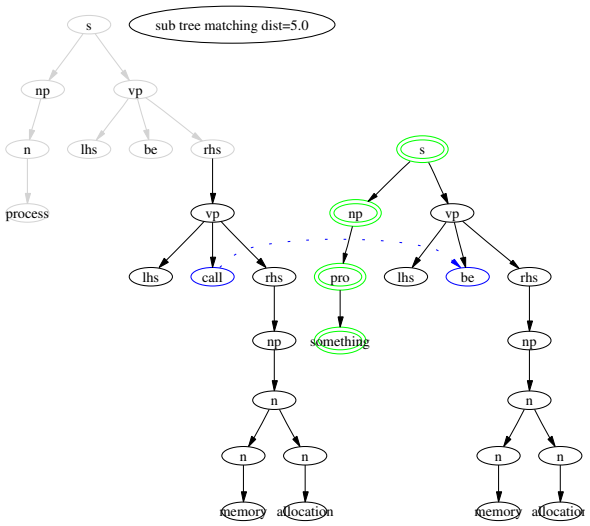


Fig. 1. Sub tree example

the optimum match to one covering the whole source tree. There is some price paid in matching the dissimilar subject nps. The procedure

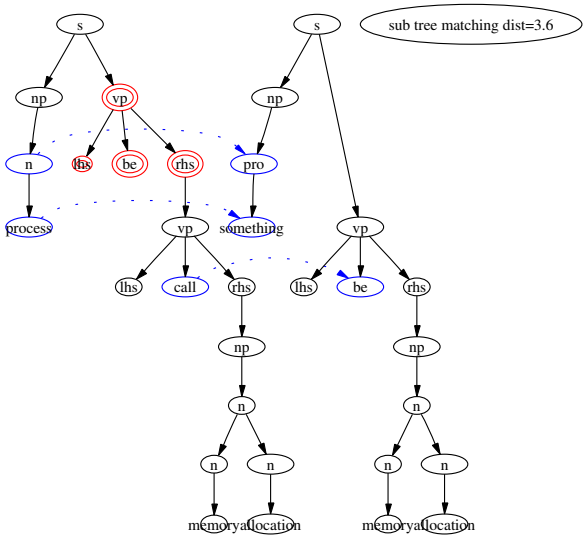


Fig. 2. Weighted example

for structural weighting depends on classifying 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.

Target wild cards: in this variant, marked target sub-trees can have zero cost matching with sub-trees in the source. Such wild card trees 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.

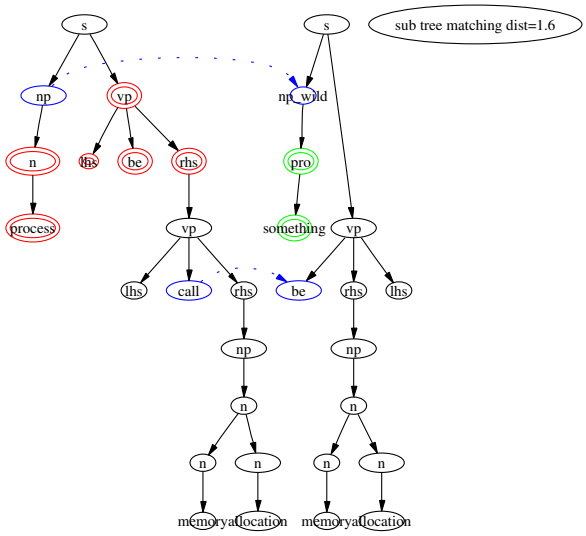


Fig. 3. Wild example

Target disjunctions: in this variant, marked target sub-trees compete as alternatives, which can be used for example to allow *what is x* to match both *x is y* and *y is x*.

Source 'self effacers': in this variant, marked source sub-trees (such as optional adjuncts) can be deleted in their entirety for no cost.

Lexical Emphasis: in this variant, the leaf nodes have weights which are scaled up in comparision to tree-internal nodes.

String Distance: if you code source and target word sequences as vertical trees, the string distance [2] between them coincides with the the tree-distance, and the sub-string distance coincides with the sub-traversal distance.

The basis of the algorithm used is the *Zhang-Shasha algorithm* [1] to compute the cost of the least cost mapping. The Appendix summarises it. The implementation is an adaptation of [3], allowing for the above-mentioned variants, and which generates human-readable displays of the chosen alignments (such as seen in figures 1,2,3).

3. The Answer Retrieval Task

Queries were formulated whose answers are single sentences in the manual of the GNU C Library [1]. For example

Q *what is a page fault*
A *When a program attempts to access a page which is not at that moment backed by real memory , this is known as a page fault*

Q *what does the free function do*
A *- Function : void free (void * ptr) The free function deallocates the block of memory pointed at by ptr*

In the retrieval task, each sentence in the manual is assigned a distance rating, measuring the distance from it, to the question. The *correct cutoff* is the proportion of the answers whose distance is less than or equal to the distance for the correct answer. Thus the lower this number is for a given query, the better the system performs on that query.

There are 88 queries. The text from which the answers come was turned into a part-of-speech tagged version which contains 360326 tokens, split into 31625 units. Usually the units are sentences, but sometimes they may be section titles.

4. Parse Quality vs Retrieval Performance

Mostly linguists agree that semantics is some kind of function of, or has some dependency on, syntactic structure. But on what semantics is, and the transparency of the relation between syntactic and semantic structures, there is a wide range of opinion. Although most would agree that one of the roles of syntactic structure is to group items which are semantically related, there are rival aims which designers of syntactic structures are striving to achieve, and these may pull syntactic structures away from semantics.

Because of this, its hard to know how successful retrieving answers to questions by means of their structural similarity could be expected to be. On the one hand there is an optimistic

[1] `www.gnu.org`

view. If thematic roles can be determined fairly easily from syntactic structure, then similar syntactic structures should be expected to lead to similar thematic role assignments and similar semantics. So, if a question has *the free function* as its subject np, other sentences which also have this as their subject, or something similar, might be expected to be a likely answers. On the other hand, there is a pessimistic view, according to which semantic structure is mostly very difficult to discern from syntactic structure, that sees syntactic structure as more of an encryption of semantics than a useful gateway into it.

This section addresses this question, and provides some evidence that one need not be overly pessimistic. What we look at is whether improvements to parse quality lead to improvements in retrieval performance.

First the performance of a particular parsing system, making uses of particular linguistic knowledge bases (see *full* in the tables below) was determined. For this discussion, the details of this parsing system are not important: briefly, it combines 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. For ease of reference let this be known as the *trinity parser*. The parsing system is imperfect, and there is much that could be objected to in the structures produced. But it could be worse. If we randomly remove 50% of the linguistic knowledge base, we could expect the structures produced to be worse (see *thin50*). Also if we manually strip out parts, again we should expect the structures to be worse (see *manual*). Worst of all, we could move to entirely *flat* parses, simply attaching unanalysed words to a top-most node. In each of these cases, we checked whether these intuitively worse parses actually diminish retrieval performance.

Damaging the grammar is easy, improving it more difficult. To try to get a picture of what might happen to retrieval performance if the parse quality improved, we hand-corrected the parses of the queries and their correct answers – see *gold* in the tables below.

Tables 1 and 2 give the results using the sub-tree and weighted sub-tree distances. The numbers describe the distribution of the *correct cutoff* over the queries, so lower numbers are better.

Table 1. Correct Cutoff *in different parse settings, ranking by sub-tree distance*

Parsing	1st Qu.	Median	Mean	3rd Qu.
flat	0.1559	0.2459	0.2612	0.3920
manual	0.0349	0.2738	0.2454	0.3940
thin50	0.01936	0.1821	0.2115	0.4193
full	0.0157	0.1195	0.1882	0.2973
gold	0.00478	0.04	0.1450	0.1944

Table 2. Correct Cutoff *in different parse settings, ranking by weighted sub-tree distance*

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

Clearly as the parsing improves there is tendency for the *correct cutoff* of a query to go down, or equivalently for the correct answer to be placed higher in the ranking. Figure 4 shows the empirical cumulative density function (ecdf) of *correct cutoff* obtained with the weighted sub-tree with wild cards measure. For each possible value of *correct cutoff*, it plots the percentage of queries whose cut-off is less than or equal to it.

I said one might be sceptical of the extent to which syntactic structures could be used as a substitute for semantic structures. An extreme sceptic would expect the strategy to produce a diagonal line in Figure 4, with the ranking placing the correct answer at a basically random point in the ordering. Instead of that, what we see is that improved parse quality pulls the line more and more towards the top left hand corner, as the number of queries answered cor-

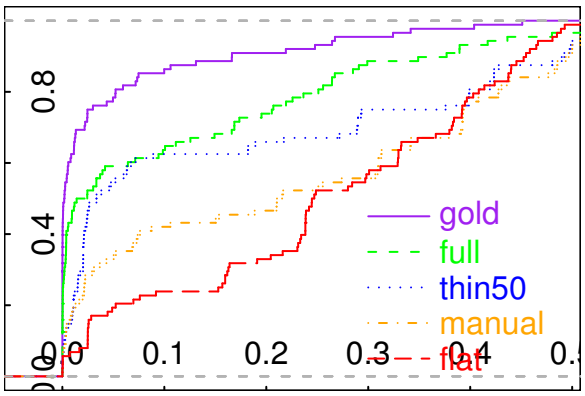


Fig. 4. *Success vs Cut-off for different parse settings: x = correct cutoff, y = proportion of queries whose correct cutoff $\leq x$ (ranking by weighted subtree with wild cards)*

rectly for a given cutoff goes steadily up.

Besides pointing in a direction in which to go to improve retrieval performance, this is also at least suggestive of a way in which the retrieval task could be used as an evaluation metric for a parser. Such an application could have some attractive features. The raw materials for the retrieval task are questions and answers, as plain text, not syntactic structures. Also, all the parser has to supply to the tree distance code is a post-order traversal of a tree, with node-types identified by positions in a symbol table. These 2 features mean that it should be fairly straightforward to use the retrieval task to evaluate *any* parser.

This is not the case if parsers are evaluated against a set of gold-standard parses – either the parser has to work directly in the notation of the gold-standard parses or there must be some kind of mapping between the notations of the parse and the gold standard.

5. Distance Measures Compared

In section 2 some parameters of variation in the definition of tree-distance were introduced: such as sub-tree vs whole tree, weights, wild cards, and lexical emphasis. The performance of some of these variants is reported in this section, including the performance of the *substring* distance. Because the tree-distance algorithm is a generalisation of the string-distance algorithm (see Appendix) it seemed natural to make a comparison. Table 3 below summarises the distribution of the *correct cut-off* over the queries for the different settings, whilst in Figure 5 the ecdf plots are given (for the distance type column -we = structural weights, -wi = wild cards, -lex = lexical emphasis, sub = subtree).

Table 3. Correct CutOff *for different distance measures*

distance type	1st Qu.	Median	Mean
sub-we-wi-lex	9.414e-05	1.522e-03	4.662e-02
substring	2.197e-04	3.609e-03	5.137e-02
sub-we-wi	7.061e-04	1.919e-02	1.119e-01
sub-we	3.891e-03	4.216e-02	1.308e-01
sub	1.517e-02	1.195e-01	1.882e-01
whole	0.040710	0.159600	0.284600

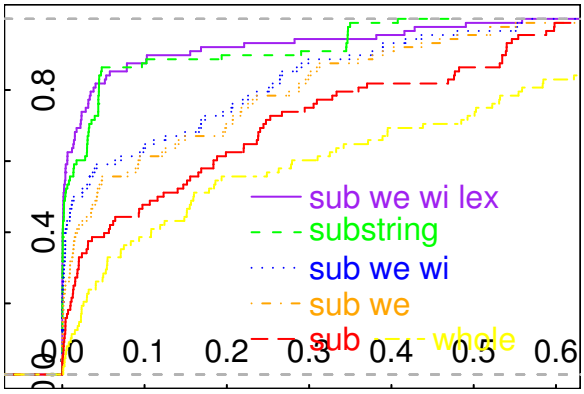


Fig. 5. *Success vs Cut-off for different distance measures: x = correct cutoff, y = proportion of queries whose correct cutoff $\leq x$*

What the data show is that the version of tree distance which uses sub-trees, weights, wild-cards and lexical emphasis, performs better than the sub-string distance, and that each of the parameters make a contribution to improved performance.

Lexical emphasis makes a large contribution to the performance. Without it, what can happen is that a structurally similar but lexically completely dissimilar false answer can be rated about the same as a lexically more similar but structurally less similar correct answer.

6. Using a different parser

Some experimentation has been done with a different parser – the Collins parser [4] (*Model 3* variant). This is a probabilistic parser, using a model of trees as built top-down with a repertoire of moves, learnt from the Penn Treebank. We performed experiments[2] in which the repertoire of moves was truncated, see Table 4. The trend seems to replicate the findings of section 4. We also compared results obtained[3] with the *trinity* parser and Collins parser – see Table 5

Table 4. *Reducing possible parser moves: n in column c is number of queries whose correct cut-off is* $\leq c$

% moves	1	10	50	100	1000
65	6	9	17	31	68
85	11	21	45	51	81
100	11	22	49	58	82

Table 5. *Collins and Trinity parser: n in column c is number of queries whose correct cut-off is* $\leq c$

parser	1	10	50	100	1000
collins	4	20	41	54	84
trinity	8	27	49	59	79

These are preliminary results, starting a process of substantiating the claim made in section 4, that the tree-distance approach should be fairly easily portable to any parser. The poorer performance with the Collins parser could be due to the way that it infers a part-of-speech for any words it has encountered in the Penn Treebank, rather than refering to the input part-of-speech tagging, which leads to some miscategorisations.

[2]for these experiments, there was a preselection of those answers with a non-empty bag-of-words overlap with the query, and then within these, ranking was by weighted sub-trees with self-effacing answer adjuncts.

[3]The same preselection was applied, and then answers were sorted by the weighted sub-traversal distance.

7. Conclusion and Future Work

We have given evidence that the tree-distance cousin of the string-distance measure can perform better than the string-distance measure on an answer retrieval task. We have also shown that improved parse quality leads to better performance by the tree-distance measure, and suggested that this shows that this kind of retrieval task could be used an evaluator for parsers. Used as an evaluator of parsers, the absolute level of retrieval performance is not particularly important, only that it increase with improvements to the parser. To use practically for answer retrieval, there are some areas to look at increase performance. We deliberately made no use of any statistics concerning the likely significance of a word, the kind of statistics used in many text retrieval approaches, focusing exclusively instead on features intrinsic to a single answer-query pair. Use of corpus-derived statistics is a topic for further work, but we anticipate such features are equally applicable to any of the distance measures considered here, and that our findings will persist in systems using such features, albeit at a higher performance level.

8. Appendix

This appendix briefly summarises the algorithm to compute the tree-distance, based on [1] (see Section 2 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 obtain the structurally weighted versions defined in section 2, each node in the tree traversal was assigned a weight between 0 and 1, in a top-down fashion, with the weight depending on the head vs. complement vs. adjunct vs. other status of the node and upon the weight of the mother node. Insertion and deletion costs = node weight, and swap costs = the heavier node weight. Space precludes giving details of the weighting procedure. For the *trinity* parser, 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 *collins* parser gives the distinctions directly.

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 $for + tree\_del = \mathcal{F}[l(i_s) - 1, j_t]$. To accommodate disjunctive target trees, the final distance is the minimum of the distances to each disjunct.

9. References

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

[2] V.I.Levenshtein, "Binary codes capable of correcting insertions and reversals," *Sov. Phys. Dokl*, vol. 10, pp. 707–710, 1966.

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

[4] Michael Collins, *Head-driven statistical models for natural language parsing*, Ph.D. thesis, 1999.