Edit Tree Distance alignments for Semantic Role Labelling

Hector-Hugo Franco-Penya
Trinity College Dublin
Dublin, Ireland.
francoph@cs.tcd.ie

Abstract

"Tree SRL system" is a Semantic Role Labelling supervised system based on a tree-distance algorithm and a simple k-NN implementation. The novelty of the system lies in comparing the sentences as tree structures with multiple relations instead of extracting vectors of features for each relation and classifying them. The system was tested with the English CoNLL-2009 shared task data set where 79% accuracy was obtained.

1 Introduction

Semantic Role Labelling (SRL) is a natural language processing task which deals with semantic analysis at sentence-level. SRL is the task of identifying arguments for a certain predicate and labelling them. The predicates are usually verbs. They establish "what happened". The arguments determine events such as "who", "whom", "where", etc, with reference to one predicate. The possible semantic roles are pre-defined for each predicate. The set of roles depends on the corpora.

SRL is becoming an important tool for information extraction, text summarization, machine translation and question answering (Màrquez, et al, 2008).

2 The data

The data set I used is taken from the CoNLL-2009 shared task (Hajič et al., 2009) and is part of Propbank. Propbank (Palmer et al, 2005) is a hand-annotated corpus. It transforms sentences into propositions. It adds a semantic layer to the Penn TreeBank (Marcus et al, 1994) and defines a set of semantic roles for each predicate.

It is difficult to define universal semantic roles for all predicates. That is why PropBank defines a set of semantic roles for each possible sense of each predicate (frame) [See a sample of the frame "raise" on the Figure 1 caption].

	Sentences	predicates	arguments	Predicates per sentence	arguments per sub-tree	File size in Mb
Tra	39279	179014	393699	4.55	2.20	56.2
Dev	1334	6390	13865	4.79	2.17	1.97
Evl	2399	10498	23286	4.38	2.22	3.41

Table 1: The data

The data set is divided into three files: training (Tra), development (Dev) and evaluation (Evl). The following table describes the number of sentences, sub-trees and labels contained in them, and the ratios of sub-trees per sentences and relations per sub-tree.

The core arguments are labelled by numbers. Adjuncts, which are common to all predicates, have their own labels, like: AM-LOC, TMP, NEG, etc. The four most frequent labels in the data set are: A1:35%, A0:20.86%, A2:7.88% and AM-TMP: 7.72%

Propbank was originally built using constituent tree structures, but here only the dependency tree structure version was used. Note that dependency tree structures have labels on the arrows. The tree distance algorithm cannot work with these labelled arrows and so they are moved to the child node as an extra label.

The task performed by the Tree SRL system consists of labelling the relations (predicate arguments) which are assumed to be already identified.

3 Tree Distance

The tree distance algorithm has already been applied to text entailment (Kouylekov & Magnini, 2005) and question answering (Punyakanok et al, 2004; Emms, 2006) with positive results.

The main contribution of this piece of work to the SRL field is the inclusion of the tree distance algorithm into an SRL system, working with tree structures in contrast to the classical "feature extraction" and "classification". Kim et al (2009) developed a similar system for Information Extraction.

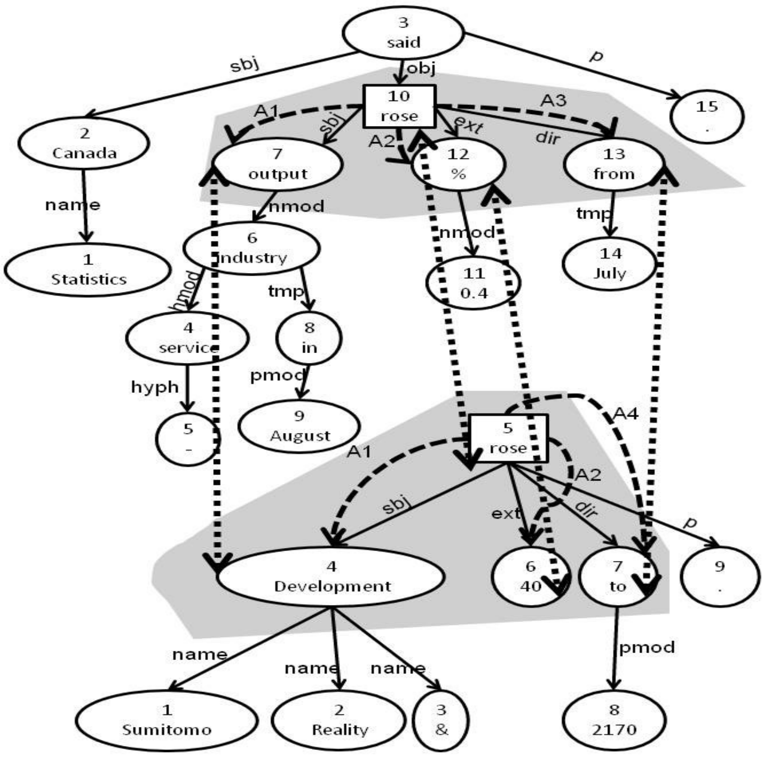

Figure 1: Alignment sample
A two sentence sample, in a dependency tree representation. In each node, the word form and the position of the word in the sentence are shown. Straight arrows represent syntactic dependencies. The label of the dependency is not shown. The square node represent the predicate that is going to be analyzed, (there can be multiple predicates in a single sentence). Semi-dotted arrows between a square node and an ellipse node represent a semantic relation. This arrow has a semantic tag (A1, A2, A3 and A4).

The grey shadow contains all the nodes of the sub tree for the "rose" predicate.
The dotted double arrows between the nodes of both sentences represent the tree distance alignment for both sub-trees. In this particular case every single node is matched.

Both predicate nodes are samples of the frame "raise" sense 01 (which means "go up quantifiably") where the core arguments are:
A0: Agent, causer of motion **A1**: Logical subject, patient, thing rising
A2: EXT, amount raised **A3**: Start point **A4:** End point **AM**: Medium

Tai (1979) introduced a criterion for matching nodes between tree representations (or converting one tree into another one) and (Shasha & Zhang, 1990; Zhang & Shasha, 1989) developed an algorithm that finds an optimal matching tree solution for any given pair of trees. The advantage of this algorithm is that its computational cost is low. The optimal matching depends on the defined atomic cost of matching two nodes.

4 Tree SRL system architecture

For the training and testing data set, all possible sub-trees were extracted. Figure 3 and Figure 5 describe the process. Then, using the tree distance algorithm, the test sub-trees are labelled using the training ones. Finally, the predicted labels get assembled on the original sentence where the test sub-tree came from. Figure 2 describes the process.

A sub-tree extracted from a sentence, contains a predicate node, all its argument nodes and all the ancestors up to the first common ancestor of all nodes. (Figure 1 shows two samples of sub-tree extraction. Figure 3 describes how sub trees are obtained)

```
**Input**: training data set (labelled)
**Input**: testing data set (unlabelled)
**Output**: testing data set (labelled)
Load training and testing data;
Adapt the trees for the tree distance algorithm;
**foreach** sentence (training & testing data) **do**
    obtain each minimal sub-tree for each predicate;
**end**
**foreach** sub-tree T from the testing data **do**
    calculate the distance and the alignment
    from T to each training sub-tree;
    sort the list of alignments by ascending
    tree distance;
    use the list to label the sub-tree T;
    Assemble T labels on the original sentence
**End**
```

Figure 2: Tree SRL system pseudo code

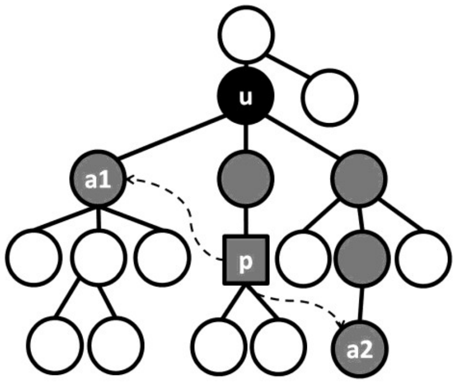

Figure 3: Sub-tree extraction sample.
Assuming that "p" (the square node) is a predicate node and the nodes "a1" and "a2" are its arguments (the arguments are defined by the semantic relations. In this case, the semi-doted arrows.), the sub-tree extracted from the above sentence will contain the nodes: "a1", "a2", "p", all ancestors of "a1","a2" and "p" up to the first common one, in this case node "u", which is also included in the sub-tree.
All of the white nodes are not included in the sub-tree. The straight lines represent syntactic dependency relations.

5 Labelling

Suppose that in Figure 1, the bottom sentence is the query, where the grey shadow contains the sub-tree to be labelled and the top sentence contains the sub-tree sample chosen to label the query. Then, an alignment between the sample sub-tree and the query sub-tree suggests labelling the query sub-tree with A1, A2 and A3, where the first two labels are right but the last label, A4, is predicted as A3, so it is wrong.

```
**Input**: A sub-tree to be labelled
**Input:** list of alignments sorted by ascending
        tree distance
**Output:** labelled sub-tree
**foreach** argument(a) in T **do**
  **foreach** alignment (ali) in the sorted list **do**
     **if** there is a semantic relation
         (ali.function(p),ali.function(a))
         **Then** break loop;
     **end**
  **end**
  label relation p-a with the label of the
  relation (ali.function(p),ali.function(a));
**end**
**p** is the node predicate.
**a** is a node argument.
**ali** is an alignment between the sub-tree that
has to be labelled and a sub-tree in the training dataset.
The method function is explained in Figure 3.
```

Figure 4: Labelling a relation. (approach A)

```
**Input**: T: tree structure labelled in post order
        traversal
**Input**: L: list of nodes to be on the sub-tree in
        post order traversal
**Output**: T: Sub-Tree
**foreach** node x in the list **do**
  mark x as part of the sub-tree;
**end**
**while** L contains more than 2 unique values **do**
  [minValue , position]=min(L);
  Value = parent(minValue);
  Mark value as part of the sub-tree;
  L[position] = value;
**end**
Remove all nodes that are not marked as part
of the sub-tree;
```

Figure 5: Sub-tree extraction

It is not necessary to label a whole sub-tree (query) using just a single sub-tree sample. However, if the whole query is labelled using a single answer sample, the prediction is guaranteed to be consistent (no repeated argument labels).

Some possible ways to label the semantic relation using a sorted list of alignments (with each sub-tree of the training data set) is discussed ahead. Each sub-tree contains one predicate and several semantic relations, one for each argument node.

5.1 Treating relations independently

In this sub-section, the neighbouring sub-trees for one relation of a sub-tree T refers to the nearest sub-trees with which the match with T produces a match between two predicate nodes and two argument nodes. A label from the nearest neighbour(s) can be transferred to T for labelling the relation.

The current implementation **(Approach A),** described in more detail in Figure 4, labels a relation using the first nearest neighbour from a list ordered by ascending tree distance. If there are several nearest neighbours, the first one on the list is used. This is a naive implementation of the k-NN algorithm where in case of multiple nearest neighbours only one is used and the others get ignored.

A negative aspect of this strategy is that it can select a different sub-tree based on the input order. This makes the algorithm indeterministic. A way to make it deterministic can be by extending the parameter "k" in case of multiple cases at the same distance or a tie in the voting **(Approach B)**.

5.2 Treating relations dependently

In this section, a sample refers to a sub-tree containing all arguments and its labels. The arguments for a certain predicate are related.

Some strategies can lead to non-consistent structures (core argument labels cannot appear twice in the same sub-tree). **Approach B** treats the relations independently. It does not have any mechanism to keep the consistency of the whole predicate structure.

Another way is to find a sample that contains enough information to label the whole sub-tree **(Approach C)**. This approach always generates consistent structures. The limitation of this model is that the required sample may not exist or the tree distance may be very high, making those samples poor predictors. The implemented method **(Approach A)** indirectly attempts to find a training sample sub-tree which contains labels for all the arguments of the predicate.

It is expected for tree distances to be smaller than other sub-trees that do not have information to label all the desired relations.

The system tries to get a consistent structure using a simple algorithm. Only in the case when using the nearest tree does not lead to labelling the whole structure, labels are predicted using multiple samples, thereby, risking the structure consistency.

Future implementations will rank possible candidate labels for each relation (probably using multiple samples).

A "joint scoring algorithm", which is commonly used (Marquez et al, 2008), can be applied for consistency checking after finding the rank probability for all the argument labels for the same predicate **(Approach D)**.

6 Experiments: the matching cost

The cost of matching two nodes is crucial to the performance of the system. Different atomic measures (ways to measure the cost of matching two nodes) that were tested are explained ahead. Results for experiments using these atomic measures are given in Table 2.

6.1 Binary system

For Binary system, the atomic cost of matching two nodes is one if label POS or dependency relations are different, otherwise the cost is zero. The atomic cost of inserting or deleting a node is always one. Note that the measure is totally based on the syntactic structure (words are not used).

6.2 Ternary system

The next intuitive measure is how the system would perform in case of a ternary cost (ternary system). The atomic cost is half if POS **or** dependency relation is different, one if POS **and** dependency relation are different or zero in all other case. For this system, Table 2 shows a very similar accuracy to the binary one.

6.3 Hamming system

The atomic cost of matching two nodes is the sum of the following sub costs:

0.25	if POS is different.
0.25	if dependency relation is different.
0.25	if Lemma is different.
0.25	if one node is a predicate but the other is not or if both nodes are predicates but with different lemma.

The cost to create or delete nodes is one. Note that the sum of all costs cannot be greater than one.

6.4 Predicate match system

The analysis of results for the previous systems shows that the accuracy is higher for the sub-trees that are labelled using sub-trees with the same predicate node. Consequently, this strategy attempts to force the predicate to be the same.

In this system, the atomic cost of matching two nodes is the sum of the following sub costs:

0.3	if POS is different.
0.3	if dependency relation is different.
1	if one is a predicate and the other node is not or both nodes are predicates but with different lemma.

The cost to create or delete nodes is one.

6.5 Complex system

This strategy attempts to improve the accuracy by adding an extra label to the argument nodes and using it.

The atomic cost of matching two nodes is the sum of the following sub costs:

0.1	for each different label (dependency relation or POS or lemma).
0.1	for each pair of different labels (dependency relation or POS or lemma).
0.4	if one node is a predicate and the other is not.
0.4	if both nodes are predicates and lemma is different.
2	if one node is marked as an argument and the other is not or one node is marked as a predicate and the other is not.

The atomic cost of deleting or inserting a node is: two if the node is an argument or predicate node and one in any other case.

7 Results

Table 2 shows the accuracy of all the systems. The validation data set is added to the training data set when the system is labelling the evaluation data set. This is a common methodology followed in CoNLL2009 (Li et al, 2009).

System	Evaluation	Development
Binary	64.36%	61.12%
Ternary	64.88%	61.28%
Hamming	78.01%	
Predicate Match	76.98%	
Complex	78.98%	

Table 2: System accuracy

Accuracy is measured as the percentage of semantic labels correctly predicted.

The implementation of the Tree SRL system takes several days to run a single experiment. It makes non viable the idea of using the development data set for adjusting parameters and that is why, for the last three systems (Hamming, Predicate Match and Complex), the accuracy over the development data set is not measured. The same reason supports adding the development data set to the training data set without over fitting the system, because the development data set is not really used for adjusting parameters.

However, the observations of the system on the development data set shows:

1. If the complexity gets increased (Ternary), the number of cases having the multiple nearest sub-trees gets reduced.
2. The output of the system only contains five per cent of inconsistent structures (Binary and Ternary), which is lower than expected. 0.5% of inconsistent sub-trees were detected in the training data-set.
3. Higher accuracy for the relations where a sub-tree is labelled using a sub-tree sample which has the same predicate node. This has led to the design of the "predicate match" and the "complex" systems.
4. Some sub-trees are very small (just one node). This resulted in low accuracy for they predicted labels due to multiple nearest neighbours.

It is surprising that the hamming measure reaches higher accuracy than the "predicate match", which uses more information, and is also surprising that the accuracies for "Hamming", "Predicate Match" and "Complex" systems are very similar.

The CoNLL-2009 SRL shared task was evaluated on multiple languages: Catalan, Chinese, Czech, English, German, Japanese and Spanish. Some results for those languages using "Tree SRL System Binary" are shown in Table 3.

Language	Accuracy on evaluation	Training data set size in Mb
English	64.36%	56
Spanish	57.86%	46
Catalan	58.49%	43
Japanese	50.71%	8
German	These languages had been excluded from the experiments because some of the sentences did not follow a dependency tree structure.	
Czech		
Chinese		

Table 3: Accuracy for other languages (Binary system)

The accuracy results for multiple languages suggest that the size of the corpora has a strong influence on the results of the system performance.

The results are not comparable with the rest of the CoNLL-2009 systems because the task is different. This system does not identify arguments and does not perform predicate sense disambiguation.

8 Conclusion

The tree distance algorithm has been applied successfully to build a SRL system. Future work will focus on improving the performance of the system by: a) trying to extend the sub-trees which will contain more contextual information, b) using different approaches to label semantic relations discussed in Section 5. Also, the system will be expanded to identify arguments using a tree distance algorithm.

Evaluating the task of identifying the arguments and labelling the relations separately will assist in determining which systems to combine to create an hybrid system with better performance.

Acknowledgments
This research is supported by the Science Foundation Ireland (Grant 07/CE/I1142) as part of the Centre for Next Generation Localisation (www.cngl.ie) at Trinity College Dublin.

Thanks are due to Dr Martin Emms for his support on the development of this project.

References

Martin Emms. 2006. Variants of Tree Similarity in a Question Answering Task. *In Proceedings of the Workshop on Linguistic Distances, held in conjunction with COLING 2006,* 100—108, Sydney, Australia, Asociation for Computational Linguistics.

Jan Hajič, Massimiliano Ciaramita, Richard Johansson, Daisuke Ka-wahara, Maria Antonia Martí, Luis Màrquez, Adam Meyers, Joakim Nivre, Sebastian Padó, Jan Štěpánek, Pavel Stravnák, Mihai Surdeanu, Nianwen Xue and Yi Zhang. 2009. The CoNLL-2009 shared task: syntactic and semantic dependencies in multiple languages. *In CoNLL '09: Proceedings of the Thirteenth Conference on Computational Natural Language Learning* (pp. 1-18). Morristown, NJ, USA: Association for Computational Linguistics.

Seokhwan Kim, Minwoo Jeong and Gary Geunbae Lee. 2009. A Local Tree Alignment-based Soft Pattern Matching Approach for Information Extraction. *Proceedings of NAAACL HLT*, 169-172. Boulder, Colorado, June 2009

Milen Kouylekov and Bernardo Magnini. 2005. Recognizing textual entailment with tree edit distance algorithms. *In Recognizing Textual Entailment* (pp. 17-20). Southampton, U.K.

Baoli Li, Martin Emms, Saturnino Luz and Carl Vogel. 2009. Exploring multilingual semantic role labeling. *In CoNLL '09: Proceedings of the Thirteenth Conference on Computational Natural Language Learning* (pp. 73-78). Morristown, NJ, USA: Association for Computational Linguistics.

Mitchell Marcus, Beatrice Santorini and Mary Ann Marcinkiewicz. 1994. Building a large annotated corpus of Eng-lish: The Penn Treebank. *Computational linguistics*, 19(2), 313–330.

Alessandro Moschitti, Daniele Pighin and Roberto Basili. 2008. Tree kernels for semantic role labeling. *Computational Linguistics,* 34(2), 193-224. Cambridge, MA, USA: MIT Press.

Lluis Màrquez, Xavier Carreras, Kenneth. C. Litkowski and Suzanne Stevenson. 2008. Semantic Role Labeling: An Introduction to the Special Issue. *Computational Linguistics*, 34(2), 145-159.

Martha Palmer, Paul Kingsbury and Daniel Gildea. 2005. The Proposition Bank: An Annotated Corpus of Semantic Roles. *Computational Linguistics*, 31(1), 71-106.

Vasin Punyakanok, Dan Roth and Wen-tau Yih. 2004. Mapping dependencies trees: An application to question answering. *In Proceedings of AI\&Math* 2004 (pp. 1-10). Ford.

Dennis Shasha and Kaizhong Zhang. 1990. Fast algorithms for the unit cost editing distance between trees. *J. Algorithms*, 11(4), 581-621. Duluth, MN, USA: Academic Press, Inc.

Kuo-Chung Tai. 1979. The Tree-to-Tree Correction Problem. *J. ACM,* 26(3), 422-433. New York, NY, USA: ACM.

Kaizhong Zhang and Dennis Shasha. 1989. Simple fast algorithms for the editing distance between trees and related problems. SIAM *J. Comput.*, 18(6), 1245-1262. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics.