Sentence diagrams: their evaluation and combination

Jirka Hana and **Barbora Hladká** and **Ivana Lukšová**
Charles University in Prague, Faculty of Mathematics and Physics
Institute of Formal and Applied Linguistics
Prague, Czech Republic
{hana,hladka,luksova} (at) ufal.mff.cuni.cz

Abstract

The purpose of our work is to explore the possibility of using sentence diagrams produced by schoolchildren as training data for automatic syntactic analysis. We have implemented a sentence diagram editor that schoolchildren can use to practice morphology and syntax. We collect their diagrams, combine them into a single diagram for each sentence and transform them into a form suitable for training a particular syntactic parser. In this study, the object language is Czech, where sentence diagrams are part of elementary school curriculum, and the target format is the annotation scheme of the Prague Dependency Treebank. We mainly focus on the evaluation of individual diagrams and on their combination into a merged better version.

1 Introduction

Syntactic parsing has been an attractive topic for both theoretical and computational linguists for many years. In combination with supervised machine learning techniques, several corpus-based parsers have been implemented (e.g., (Nivre et al., 2007), (de Marneffe et al., 2006), (McDonald et al., 2005)), combined (e.g., (Surdeanu and Manning, 2010)), and adapted (e.g., (McClosky et al., 2010),(Zhang and Wang, 2009)). The performance of such techniques directly correlates with the size of training data: the more annotated data, the better. However, the annotation process is very resource consuming, thus we have been seeking for alternative ways of faster and cheaper annotation. Namely, we have been inspired by the solution of crowdsourcing, see e.g. (Brabham, 2013).

In Czech schools, practicing morphology and syntax is an obligatory part of the curriculum. Schoolchildren draw sentence diagrams similar to syntactic trees in dependency grammar theories (Hudson, 1984; Sgall et al., 1986; Mel'čuk, 1988), with labeled nodes and edges. Our goal is to collect such diagrams and transform them into the annotation scheme of the Prague Dependency Treebank (Hajič et al., 2006). Thereby we enlarge training data for taggers and parsers of Czech. Traditionally, diagrams that we need are only in students' notebooks so they are not accessible to us at all. Since we require diagrams electronically, we have been developing a sentence diagram editor *Čapek*. We have designed it both as a CALL (Computer-Assisted Language Learning) system for practicing morphology and dependency-based syntax and as a crowdsourcing system for getting annotated data. In addition, the editor can be used for drawing sentence diagrams in any natural language. On the other hand, transformation rules have to be specified with respect to a particular target annotation scheme. We introduced this approach in (Hana and Hladká, 2012).

Data quality belongs to the most important issues related to crowdsourcing, see e.g. (Sabou et al., 2012), (Wang et al., 2010), (Hsueh et al., 2009). We discuss the data quality from two aspects: (i) evaluation of students' diagrams against teachers' and/or other students' diagrams, i.e. we consider how diagrams are similar; (ii) combination of students' diagrams of one sentence to get a better diagram, i.e. we deal with multiple, possibly noisy, annotations and we study if they are useful.

Our paper is organized as follows: in Section 2, we describe Czech sentence diagrams and how they differ from the PDT annotation scheme. We introduce the Čapek editor in Section 3. Section 4 introduces

LAW VIII - The 8th Linguistic Annotation Workshop, pages 38–47,
Dublin, Ireland, August 23-24 2014.

a tree edit distance metric we use to quantify the difference between diagrams. Section 5 discusses an algorithm combining alternative diagrams into a single structure. Finally, some initial evaluation and other statistics are presented in Section 6.

2 Czech sentence diagrams

In the Czech sentence diagrams (hence SDs), a sentence is represented as a type of dependency structure.[1] The structure is a directed acyclic graph (roughly a tree) with labeled nodes. The nodes correspond to words: one (most common), multiple (auxiliary words are considered markings on their heads, e.g. preposition and noun, or a complex verb form share a single node) or none (in case of dropped subjects). The edges capture the dependency relation between nodes (e.g., between an object and its predicate). The node label expresses the type of dependency, or syntactic function.
Formally, a sentence diagram over a sentence $s = w_1\ w_2\ \ldots\ w_n$ is a directed acyclic graph $D = (Nodes, Edges)$, where $Nodes$ is a partition of s. Moreover, the $Nodes$ set might contain a dummy node corresponding to a dropped subject. The first node N_1 of an edge $E = (N_1, N_2)$ is a child node of the second node N_2.

For illustration, let's consider the sentence in (1) and its diagram in Figure 1:

(1) (—) Ráno půjdu se svým kamarádem na houby.
I in the morning will go with my friend mushrooming.
'I will go mushrooming with my friend in the morning.'

Since our goal is to get more data annotated according to the PDT schema (the so-called a-layer or surface syntax), we characterize certain aspects of SD with respect to the PDT conventions depicted in Figure 2:

- **Tokenization.** There is a 1:1 correspondence between tokens and nodes in PDT; all punctuation marks have their corresponding nodes. Cf. 8 tokens and 8 nodes in Example 1 and Figure 2. In SDs, there is an N:1 correspondence between tokens and nodes (N can be 0 for dropped subjects); punctuation is mostly ignored. Cf. 8 tokens and 6 nodes in Example 1 and Figure 1.

- **Multi-token nodes.** SDs operate on both single-token (*půjdu* 'will go') and multi-token nodes (*se kamarádem* 'with friend', *na houby* 'for mushrooms'). The tokens inside each multi-token node are ordered in accordance with their surface word order. Auxiliary words, auxiliary verbs, prepositions, modals etc. do not have their own nodes and are always part of a multi-token node. PDT handles single-token nodes only.

- **Subject and predicate.** In PDT, predicate is the root and the subject depends on it; in Figure 1, they are on the same level; cf. the nodes for *(já) půjdu* 'I will go'.

- **PRO subject.** SDs introduce nodes for elided subjects (see the --- node in Figure 1), which are common in Czech. PDT does not represent them explicitly.

- **Morphological tags.** We adopt the system of positional tags used in PDT to capture morphological properties of words. Tags are assigned to each token in the sentence, not to the nodes.

- **Syntactical tags (functors)**. Our SDs use 14 syntactical tags (Subject, Predicate, Attribute, Adverbial of time/place/manner/degree/means/cause/reason/condition/opposition, Verbal Complement). PDT distinguishes significantly higher number of functors, but most of the additional tags are used in rather specific situations that are captured by different means in school syntax (parenthesis, ellipsis), are quite technical (punctuation types), etc. In the vast majority of cases, it is trivial to map SD functors to PDT functors.

[1] For expository reasons, in this paper, we ignore complex sentences consisting of multiple clauses. Their SD is a disconnected graph where each component is an SD of a single clause. Such sentences and graphs are however part of the evaluation in Section 6.

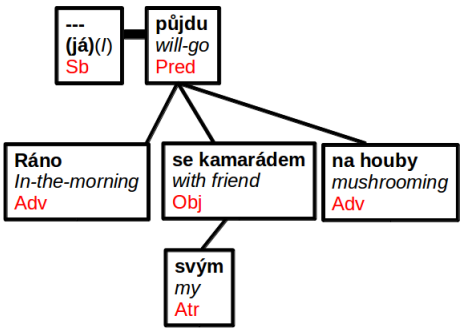

Figure 1: A sample of sentence diagram

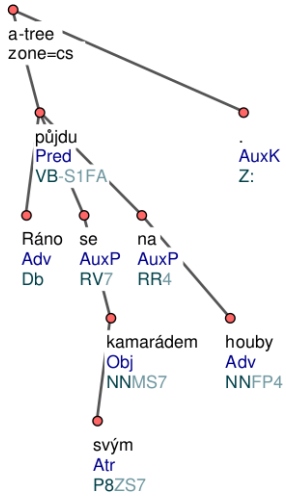

Figure 2: A sample of PDT tree

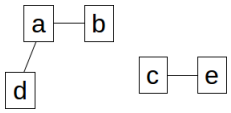

Figure 3: A possible sentence diagram draw in Čapek

3 Čapek editor

Since we wanted to provide students with a sentence diagram editor that is easy to use, we have decided not to use the TrEd editor,[2] a versatile, flexible but also complex tool, which is used as the main annotation tool of the Prague Dependency Treebanks. Instead, we decided to implement *Čapek*, a new system. It exists as a desktop application, written in Java on top of the Netbeans Platform, [3] and as a web application.[4]

Students use the editor in a similar way as they are used to use chalk/pen at school. A simple and intuitive GUI supports the following operations:

- **`JOIN`** Merge two nodes into a single multi-token node.
- **`SPL`** Divide a multi-token into nodes corresponding to single tokens.
- **`INS`** Create a node for elided subject.
- **`LINK`** Link a node to its governing parent node.
- **`LAB`** Label a node with syntactic function.
- **`MLAB`** Label a token with morphological function.

Intentionally, we did not make Čapek to perform any consistency checks, except acyclicity of the graph. Thus students can create a graph with several components, all nodes can be a subject, etc.

4 Similarity of sentence diagrams

We compute the similarity between sentence diagrams using a tree edit distance. Our definition is based on a tree edit distance in (Bille, 2005). It assumes two trees T_1, T_2 and three edit operations: relabeling a node, deleting a non-root node, and inserting a node. T_1 is transformed into T_2 by a sequence of edit

[2] http://ufal.mff.cuni.cz/tred
[3] http://platform.netbeans.org
[4] http://capek.herokuapp.com/

operations S. Each operation has a particular cost, the cost of the sequence S is simply the sum of the cost of individual operations. Then *tree edit distance* between two trees is the cost of a cheapest sequence of operations turning one tree into another.

Our situation is similar, however:

- the compared sentence diagrams are always over the same sentence, i.e. over the same set of tokens
- diagrams are not trees: they are acyclic graphs but unlike trees they might consist of several components (either because they capture complex sentences, or because the students did not finish them). In addition, a diagram usually has two "roots": one for the subject and one for predicate. However, it is trivial to transform them into the corresponding tree, considering the subject to be the daughter of the predicate.

Thus, we modify the distance from (Bille, 2005). For an example, see Figure 4 with nodes of two particular diagrams over a 6-token sentence. The arrows show a token-node mapping specified by the annotator of D_1:

- Let D_1 and D_2 be sentence diagrams; we are turning D_2 into D_1.
- We consider the following operations:
  - `SPL` – detaching a token from a node
  - `JOIN` – adding a token to a node
  - `INS` – adding an empty node (used for elided subjects)
  - `LINK` – linking a node with its parent and removing all inconsistent edges. If manipulating a non-root node, relink the node to its new parent and remove the edge to its former parent. If manipulating a root node, like *a* in Figure 5 a), link the node to its new parent, e.g. to *e*, see Figure 5 b). Then the diagram consists of a single cycle. Thus remove the edge from *e* to its former parent *c* and *e* becomes a root, see Figure 5 c).
  - `SLAB` – change node syntactic label

  All operations are assumed to have the cost of 1. Without loss of generality, we can assume that operations are performed in stages: first all `SPL`s, then all `JOIN`s, etc. In Figure 4, first we apply `SPL` twice on the nodes $[b, c]$, $[d, e, f]$ and then `JOIN` also twice on the nodes $[a]$, $[b]$ and $[e]$, $[f]$.
- Finally, the measure is normalized by sentence length. Thus, we redefine the tree edit distance $TED(D_1, D_2, n)$ for diagrams D_1, D_2 and sentence of n tokens as follows:

$$TED(D_1, D_2, n) = (\#SPL + \#JOIN + \#INS + \#LINK + \#SLAB)/n$$

  .
- We define the tree edit distance for annotators A_1, A_2 and a set of sentences S ($s_i \in S$) as the average tree distance over those sentences:

$$\overline{TED}(A_1, A_2, S) = \frac{1}{|S|} \sum_{i=1}^{|S|} TED(D^i_{A_1}, D^i_{A_2}, |s_i|)$$

  .

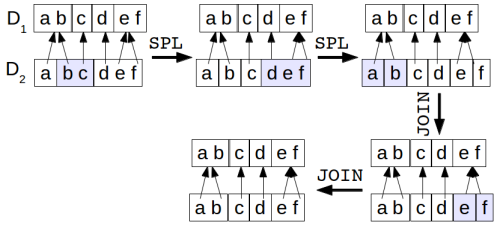

Figure 4: Turning nodes of D_2 into nodes of D_1

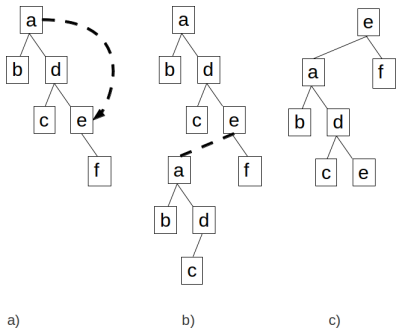

Figure 5: Linking a root node

5 Combination of sentence diagrams

We deal with sentence diagrams and their differences before transformation into a target annotation scheme. We propose a majority-voting method to combine m multiple diagrams $D_1, \ldots, D_m$ created by m different users over the sentence $s = w_1\ w_2\ \ldots\ w_n$. In some sense, our task is similar to the task of combination independently-trained syntactic parsers. However, at least to our knowledge, the experiments performed so far, e.g. (Surdeanu and Manning, 2010), are based on the assumption that all input parsers build syntactic structures on the same set of nodes. Given that, we address a significantly different task. We approach it using the concept of assigning each candidate node and edge a score based on the number of votes it received from the input diagrams. The votes for edges are weighted by a specific criterion.

To build a final diagram, we first create its set of nodes $FinalNodes$, then its set of edges $FinalEdges$ linking nodes in $FinalNodes$, and finally extend the set of nodes by any empty nodes. The method can produce both nodes and edges that do not occur in any of the input diagrams.

Building $FinalNodes$

1. $\forall t, u \in s\ .\ v(t,u) = \sum_{k=1}^{m} \delta([t,u], D_k)$, where $\delta([t,u], D) = 1$ if the tokens t and u are in the same node in the diagram D, and 0 otherwise. We compute the number of votes $v(t,u)$ to measure user preferences for having token pair t, u in one node. In total, there are $\binom{|s|}{2}$ token pairs.

2. The set $FinalNodes$ is formed as a partition over tokens induced by the $v(t,u)$ equivalence relation:

   $FinalNodes = s/eq$ where $eq(t,u) \Leftrightarrow v(t,u) > m/2$

For illustration, we start with the sentence $a\ b\ c\ d$ and three diagrams with nodes displayed in Figure 6. All of them consist of two nodes, namely $Nodes_1 = \{[a,b,c],[d]\}$, $Nodes_2 = \{[a],[b,c,d]\}$, $Nodes_3 = \{[a,b],[c,d]\}$. First, we calculate the votes for each possible token pairs, see Table 1. There are two candidates with a majority of votes, namely (a,b) and (b,c), both with two votes. Thus, $FinalNodes = \{[a,b,c],[d]\}$. A final diagram consists of n nodes $[w_1],\ \ldots\ ,[w_n]$ if there is no candidate with majority of votes, see Figure 7 and Table 2.

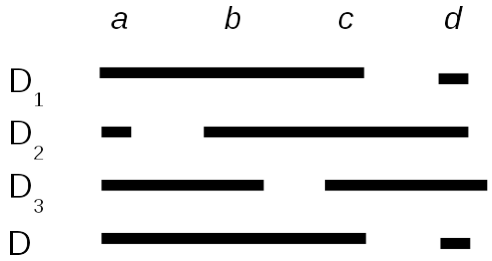

Figure 6: Sentence $a\ b\ c\ d$ and nodes in three diagrams

	a	b	c	d
a	x	**2**	1	0
b	x	x	**2**	1
c	x	x	x	1
d	x	x	x	x

Table 1: Two candidates for joining

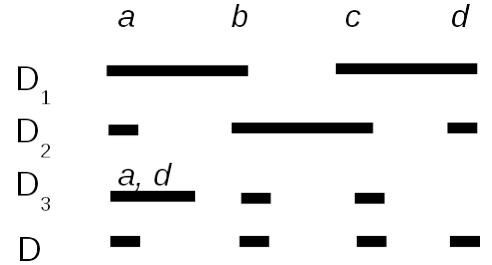

Figure 7: Sentence $a\ b\ c\ d$ and nodes in three other diagrams

	a	b	c	d
a	x	1	0	1
b	x	x	1	0
c	x	x	x	1
d	x	x	x	x

Table 2: No candidates with the great majority of votes

Building $FinalEdges$

1. $fn = |FinalNodes|$

2. $\forall D_{k=1,\dots,m}$, $\forall E = (N_1, N_2) \in Edges_k$, $\forall (t,u) \in tokens(N_1) \times tokens(N_2) : v^k(t,u) = 1/(|tokens(N_1)||tokens(N_2)|)$. We compute $v^k(t,u)$ to measure user preference for having token t in a node dependent on a node containing u. We take it proportionally to the number of tokens in two particular nodes.

3. We initialize a set of potential edges as a set of all possible edges over the final nodes. I.e. $PotentialEdges$ is formed as a variation of fn nodes choose 2. Let $p = |PotentialEdges| = fn(fn-1)$. Then weights are assigned to the potential edges:

   $\forall E = (N_1, N_2) \in PotentialEdges : v_E = \sum_{k=1}^{m} v^k(t,u), (t,u) \in tokens(N_1) \times tokens(N_2)$

4. Sort $PotentialEdges$ so that $v_{E_1} \geq v_{E_2} \geq \cdots \geq v_{E_p}$

5. $FinalEdges := \emptyset$

6. until $PotentialEdges = \emptyset$

   - $FinalEdges := FinalEdges \cup E_1$
   - $PotentialEdges := PotentialEdges \setminus E_1$
   - $PotentialEdges := PotentialEdges \setminus -E_1$
   - $PotentialEdges := PotentialEdges \setminus \{E : E \cup FinalEdges \text{ has a cycle}\}$

For illustration, we assume three diagrams D_1, D_2, D_3 displayed in Figure 8. We compute weights of token pairs proportionally to the number of tokens in nodes identifying a given edge, e.g. the edge $([a,b],[c])$ in D_1 determines two token pairs (a,c) and (b,c), each of them with the weight $1/2$. See Table 3 for other weights. Let $FinalNodes = \{[a,b],[c],[d]\}$. There are six possible edges connecting the final nodes, namely $([a,b],[c])$,$([c],[a,b])$,$([a,b],[d])$,$([d],[a,b])$,$([c],[d])$,$([d],[c])$. For each of them, we compute its weight, see Table 4. Then we sort them – $([a,b],[c])$, $([c],[d])$, $([a,b],[d])$, $([c],[a,b])$, $([d],[a,b])$, $([d],[c])$. Table 5 traces the algorithm for adding edges into a final diagram. Finally, we get the diagram D in Figure 8.

	$([a,b],[c])$	$([c],[a,b])$	$([a,b],[d])$	$([d],[a,b])$	$([c],[d])$	$([d],[c])$
weight	13/6	0	1/2	0	1	0

Table 4: Computing weights of edges-candidates to be added into a final diagram

1^{st}	$FinalEdges$						
	$PotentialEdges$	$([a,b],[c])$	$([c],[d])$	$([a,b],[d])$	$([c],[a,b])$	$([d],[a,b])$	$([d],[c])$
2^{nd}	$FinalEdges$	$([a,b],[c])$					
	$PotentialEdges$		$([c],[d])$	$([a,b],[d])$	~~$([c],[a,b])$~~	$([d],[a,b])$	$([d],[c])$
3^{rd}	$FinalEdges$	$([a,b],[c])$	$([c],[d])$				
	$PotentialEdges$			~~$([a,b],[d])$~~		~~$([d],[a,b])$~~	~~$([d],[c])$~~

Table 5: Adding edges into a final diagram

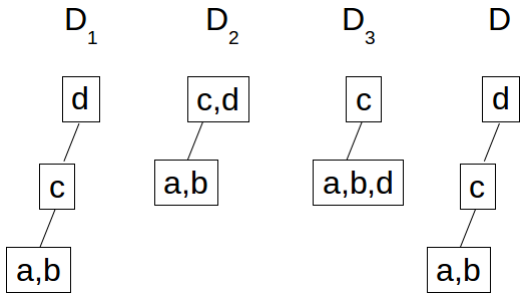

Figure 8: Input diagrams D_1, D_2, D_3 and final diagram D

D_1		D_2		D_3	
token pair	weight	token pair	weight	token pair	weight
(a,c)	1/2	(a,c)	1/4	(a,c)	1/3
(b,c)	1/2	(a,d)	1/4	(b,c)	1/3
(c,d)	1	(b,c)	1/4	(d,c)	1/3
		(b,d)	1/4		

Table 3: Assigning weights to token pairs

6 Data and initial experiments

We randomly selected a workbench of 101 sentences from a textbook of Czech language for elementary schools (Styblík and Melichar, 2005) with the average length of 8.5 tokens, for details see Figure 9. These sentences were manually analysed according to the school system with the emphasis placed on syntactic analysis. Namely, elementary school teachers T1 and T2 and secondary school students S1 and S2 drew school system diagrams using Čapek 1.0. Teachers T1 and T2 are colleagues from the same school but they were drawing diagrams separately. Students S1 and S2 study at different schools and they are students neither of T1 nor T2. In Table 6, we present $\overline{TED}$ for pairs of teachers and students. As we expected, the teachers' diagrams are the most similar ones and on the other hand, the students' diagrams are the most different one. Taking teacher T1 as a gold-standard data, student S1 made less errors that student S2. We analyzed differences in details considering two aspects:

- Do nodes closer to the root node cause more differences? A diagram D2 is transformed into a diagram D1 by a sequence of operations (`SPL`, `JOIN`, `INS`, `LINK`, `SLAB`) where the first operation

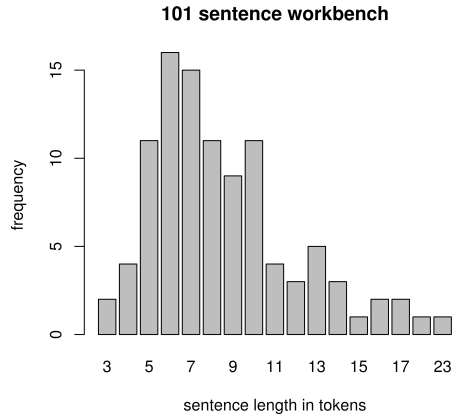

Figure 9: Length of sentences in the workbench

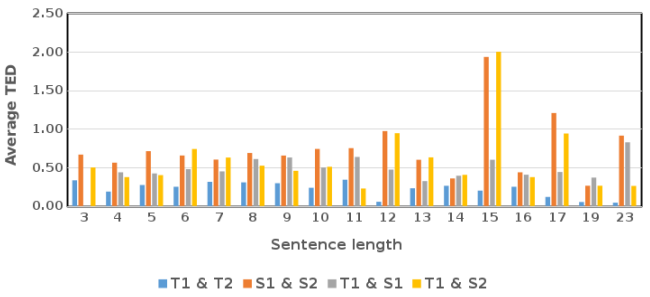

Figure 10: $\overline{TED}$ vs. Sentence length

	(T1,T2)	(T1,S1)	(T1,S2)	(S1,S2)	U1	U2	U3	U4	U5	U6	U7	MV
# of sentences	101	91	101	91	10	10	10	10	10	10	10	10
$\overline{TED}$	0.26	0.49	0.56	0.69	0.78	0.63	0.56	0.76	0.38	0.62	1.21	0.40

Table 6: $\overline{TED}$ for pairs of teachers and students, for pairs of teacher T1 and users U1,...,U7 and their combination MV

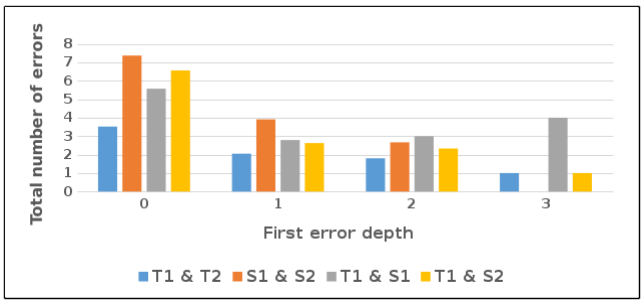

Figure 11: First Error Depth

is applied on the node in some depth of D2 (where the depth of a node is the length of the path from the root to that node). Figure 11 illustrates this depth for pairs of teachers and students. We observe that the very first operation is applied in the root nodes mostly. So we can suppose that recognizing predicate and its dependent nodes is the most difficult step for users.

- Do longer sentences cause more difficulties? In Figure 10, we observe that the sentence length does not influence discrepancies between teachers at all (measured by $\overline{TED}$). For students, we can see peaks for sentences of 12, 15, 17, 23 tokens. However, we suppose that longer sentences do not cause obstacles for them.

A group of 7 users U1, ..., U7, graduate and undergraduate students, drew diagrams for 10 (S_{10}) sentences randomly selected from the workbench using Čapek 2.0. We merged their analyses using the MV algorithm. When the final diagrams are compared to the diagrams by the T1 teacher, we get $\overline{TED}(T_1, MV(U1, \ldots, U8), S_{10}) = 0.4$. To see whether we built a better final diagram, we computed $\overline{TED}(T_1, U_i, S_{10})$ for each user – see columns U1,...,U7 in Table 6. One can see that only one user (U5) has a slightly better agreement with the T1 diagrams. The user U7 actually managed to have more than one error (differences from T1) per annotated token.

7 Conclusion

In this paper, we have shown our motivation for getting more syntactically annotated data by sentence diagrams transformation. We have implemented Čapek, a diagram editor, which allows students to perform sentence analysis electronically. We can then collect their diagrams easily. The editor is designed as both a CALL and crowdsourcing system for practicing morphology and syntax and for collecting diagrams over a given set of sentences. Both aspects have to deal with a quantitative measure of agreement, therefore we designed a tree edit distance metric comparing two or multiple diagrams. In addition, we have formulated an algorithm combining multiple crowdsourced diagrams into a single better diagram. Finally, we presented the results of a pilot study with promising results.

In the near future, to get more statistically significant results, we plan to address the following issues:

- evaluating the combination algorithm on complex sentences
- specifying the practice of crowdsourcing: how to distribute tasks, and how to assign voting weights to users based on their past results
- getting more diagrams

Acknowledgements

The authors would like to thank numerous persons for their sentence diagrams drawn using Čapek. We gratefully acknowledge support from the Charles University Grant Agency (grant no. 1568314), Charles University in Prague, Faculty of Mathematics and Physics. This work has been using language resources developed and/or stored and/or distributed by the LINDAT/CLARIN project of the Ministry of Education, Youth and Sports of the Czech Republic (project LM2010013).

References

Philip Bille. 2005. A survey on tree edit distance and related problems. *Theoretical computer science*, 337(1):217–239.

Daren C. Brabham. 2013. *Crowdsourcing*. MIT Press.

Marie-Catherine de Marneffe, Bill MacCartney, and Christopher D. Manning. 2006. Generating typed dependency parses from phrase structure parses. In *Proceedings of the 8th International Conference on Language Resources and Evaluation (LREC 2006)*, pages 449–454.

Jan Hajič, Eva Hajičová, Jarmila Panevová, Petr Sgall, Petr Pajas, Jan Štěpánek, Jiří Havelka, and Marie Mikulová. 2006. Prague Dependency Treebank 2.0. Number LDC2006T01. Linguistic Data Consortium.

Jirka Hana and Barbora Hladká. 2012. Getting more data: Schoolkids as annotators. In *Proceedings of the 8th International Conference on Language Resources and Evaluation (LREC 2012)*, pages 4049–4054, İstanbul, Turkey. European Language Resources Association.

Pei-Yun Hsueh, Prem Melville, and Vikas Sindhwani. 2009. Data quality from crowdsourcing: A study of annotation selection criteria. In *Proceedings of the NAACL HLT 2009 Workshop on Active Learning for Natural Language Processing*, HLT '09, pages 27–35, Stroudsburg, PA, USA. Association for Computational Linguistics.

Richard Hudson. 1984. *Word Grammar*. Blackwell.

David McClosky, Eugene Charniak, and Mark Johnson. 2010. Automatic domain adaptation for parsing. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, HLT '10, pages 28–36, Stroudsburg, PA, USA. Association for Computational Linguistics.

Ryan McDonald, Fernando Pereira, Kiril Ribarov, and Jan Hajič. 2005. Non-projective dependency parsing using spanning tree algorithms. In *Proceedings of the Conference on Human Language Technology and Empirical Methods in Natural Language Processing*, HLT '05, pages 523–530, Stroudsburg, PA, USA. Association for Computational Linguistics.

Igor Mel'čuk. 1988. *Dependency syntax: theory and practice*. State University of New York Press.

Joakim Nivre, Johan Hall, Jens Nilsson, Atanas Chanev, Glsen Eryigit, Sandra Kübler, Svetoslav Marinov, and Erwin Marsi. 2007. Maltparser: A language-independent system for data-driven dependency parsing. *Natural Language Engineering*, 13(2):95–135.

Marta Sabou, Kalina Bontcheva, and Arno Scharl. 2012. Crowdsourcing research opportunities: Lessons from natural language processing. In *Proceedings of the 12th International Conference on Knowledge Management and Knowledge Technologies*, i-KNOW '12, pages 17:1–17:8, New York, NY, USA. ACM.

Petr Sgall, Eva Hajičová, and Jarmila Panevová. 1986. *The Meaning of the Sentence and Its Semantic and Pragmatic Aspects*. Academia/Reidel Publishing Company, Prague, Czech Republic/Dordrecht, Netherlands.

Vlastimil Styblík and Jiří Melichar. 2005. *Český jazyk - Přehled učiva základní školy*. Fortuna.

Mihai Surdeanu and Christopher D. Manning. 2010. Ensemble models for dependency parsing: Cheap and good? In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, HLT '10, pages 649–652, Stroudsburg, PA, USA. Association for Computational Linguistics.

Aobo Wang, Cong Duy Vu Hoang, and Min-Yen Kan. 2010. Perspectives on crowdsourcing annotations for natural language processing.

Yi Zhang and Rui Wang. 2009. Cross-domain dependency parsing using a deep linguistic grammar. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 1 - Volume 1*, ACL '09, pages 378–386, Stroudsburg, PA, USA. Association for Computational Linguistics.