Tree-banking with parsing aids: an effort assessment using Boardedit

Nicolas AUCLERC & Yves LEPAGE
{nicolas.auclerc, yves.lepage}@slt.atr.co.jp
ATR 音声言語通信研究所

1 Introduction

We see the process of building a treebank as a sequence of edition and search operations. We have proposed a tool which incorporates a tree editor (Lepage & Auclerc 00) with parsing aids `Boardedit`. The aim of this paper is to estimate the effort in edit operations (not in time) needed under this tool to augment a treebank.

2 Parsing aids

The construction of a treebank is a very cumbersome and time-consuming process and different techniques have been proposed to alleviate it, *e.g.* (Brants & Crocker 00). We propose to use the following steps in the construction of a treebank:

1. Look whether the input sentence already exists in the treebank. For that, apply an exact match research method. If there is no result, then continue.
2. Apply completion by analogy (Lepage 99): this is more than a searching method (it is based on analysis by analogy and builds a candidate for the sentence, from the treebank). Adapt the tree if necessary. If there is no result then continue.
3. Look for a similar sentence. For that, apply approximate matching. If there are results, retrieve their associated structures and adapt them to obtain the new linguistic structure. If there is no relevant result, then continue.
4. Build the tree by hand from scratch.

3 Tree editor

Our tree editor is intuitive to use thanks to a parallel with text editing. Moreover, it is economic in the sense that drawing a tree requires less operations than typing in the corresponding parenthesised representation.

3.1 Tree editing model

The tree editor specification uses a very special case of trees: nodes bear only labels, and no further information. This leads to a parallel between nodes and subtrees on the one hand, and words and lines on the other hand (See Table 1). Thus, any editing function (click, select, insert, cut, copy, paste, etc.) for trees will have exactly the same behaviour as in text-editing.

Table 1: Parallel between tree- and text-editing

Tree	Text
label of node	word
node	–
complete subtree	lines

Although the parallel clearly shows that a node is different from a label. This distinction is usually not intuitive for a naïve user. People usually think that "a label is a node." To make our tool intuitive to users, we do not contradict this way of thinking.

There are three ways to manipulate a tree:

- The keyboard: Table 2 shows that each key operation costs only one operation.
- The mouse: it allows the user to create a node by clicking on a sensitive area and to make selection of nodes and complete subtrees.
- The clipboard: it allows the user to transfer a complete subtree in two keystrokes (one for copy, one for paste).

Table 2: Equivalences of edit functions in trees and text

Click	Effect	Place		Cost
		Text	Tree	
single (sensitive area)	position cursor in...	–	new node	1
single (on a selection)	start drag and drop	text	tree	1
single (on a node)	position cursor in...	word	node	1
double (on a node)	select the...	word	node	2
triple (on a node)	select the...	line	complete subtree	3
Keystroke	Effect	Place		Cost
		Text	Tree	
`<sp>`	start a new...	word	node as right sister	1
`<ret>`	start a new...	line	node as daughter	1
`<↑>`	move cursor to...	line above	mother node	1
`<↓>`	move cursor to...	line below	leftmost daughter node	1

3.2 Gain in using the tree editor

The upper part of Table 3 shows the correspondence between the keystrokes listed in Table 2 and separators used in parenthesised representations of trees. It makes explicit that our tree editing model follows the text editing model closely.

Table 3: Operations corresponding to separators

Separator	Keystroke
(	`<ret>`
,	`<sp>`
)	`<↑>`
Separator	Click
[ )]*,	`<single>`

As a result, a tree like A(B(C,D)),E will be inputted in our tree editor by simply replacing the separators with their corresponding keystrokes. With this, the effort to input a tree under our tree editor would be the same as inputting the parenthesised representation under a text editor. However, we do not need any closing parentheses to get a tree drawn. By using one click we can reduce the number of keystrokes: any sequence of closing parentheses followed by a comma can be replaced by a single click of the mouse to create a new node at the right position on the drawing panel (see lower part of Table 3). The previous tree can thus be inputted by the sequence: A `<ret>` B `<ret>` C `<sp>` D `<single click>` E. In this way, the number of needed operations, *i.e.* the effort, is just twice the number of nodes to create the structure plus the total number of characters in the labels.

In all cases, inputting or modifying a tree under our editor takes less effort, measured by operations (clicks and keystrokes), than inputting or modifying its parenthesised representation under a text editor.

4 Experiments

4.1 Data

We augmented a tree-bank of 5 000 sentences (the base set) with 1 553 new sentences (the test set). Our data come from ATR-NEC dependency tree-bank (Lepage & al. 98). The sentences consist of Japanese hotel reservation dialogues. The tree structures use dependency representations.

We used the steps presented in section 2 to insert the 1 553 new sentences into the treebank. In fact, for each of these 1 553 sentences, we already know the corresponding linguistic structures which we must get at the end. By comparing these structures with the results obtained by exact matching, completion by analogy or approximate matching, we can measure the effort needed to obtain the exact structure counted in number of operations (keystrokes and clicks).

4.2 Baselines

The first baseline is obtained by doing as if we would have inputted all the new trees under a text editor in their parenthesised representations. This would have cost us 88 394 keystrokes (total number of characters in labels and separators).

The second baseline is obtained by using the tree editor. If we would have built the structures of the 1 553 trees by hand, this would have cost us 13 006 clicks under the tree editor to design the structures and 41 789 characters to type in all nodes. In total, this is 54 795 operations. As said above, we see that using the tree editor easies the task. The effort is reduced by (88 394 – 54 795)/88 394 = 38%.

4.3 Proposed method with parsing aids

4.3.1 Exact matching

First, we apply exact matching. Of the 1553 sentences (sequences of syntactic classes), 493 are already in the treebank. In 465 of the cases, the treebank provides an exact parse, which means no effort to do (0 operations). In the remaining 28 cases, a different tree than the exact one is retrieved, due to different representational choices. For instance, the representations of "すぐにできますよ。" and "三号車にありますよ。" are not the same, although the sequences of categories are the same, but すぐに is adverbial whereas 三号車に is a location complement.

We computed the effort needed if we would have edited the 28 wrong structures. This effort is the effort needed to modify by hand these structures to make them equal to the exact structures. We counted a total of 64 operations needed using our tree editor.

4.3.2 Completion by analogy

In the cases where the sentences were not exactly parsed by exact matching (1553 – 465 = 1088 sentences), sentences, we now apply the technique of completion by analogy.

Table 4: Results of completion by analogy

	number of sentences	percentage	
total	1 088		100%
parsed	701	100%	64%
exactly parsed	337	48%	30%

The outputs of completion by analogy are compared to the exact answers[1]. In 64% of the cases (701 sentences), we got at least one structure proposed. In almost half of the cases (337 sentences), one of the structures proposed is the exact one, so that no effort to edit is needed in those cases. However, scanning through the answers is needed: in the worst case, this is 5 clicks. In total, this would be $337 \times 5 = 1\ 685$.

If we wanted to edit the structures proposed in the 701 – 337 = 364 other cases, the effort needed is measured by the edit distance on trees. In average, there are 3.95 tree structures proposed, so that, in the worst case, we need $3.95 \times 364 = 1438$ clicks to scan until the best tree proposed. We counted 1891 clicks needed to transform the best structure proposed into the exact structure. As a total this yields $1891 \times (1 + 3.21) = 7\ 961$ operations (clicks and keystrokes) to perform the transformations[2].

4.3.3 Approximate matching

In the cases where the sentences were not exactly parsed by completion by analogy (1 088 – 337 = 751 sentences), we apply approximate matching. This consists in finding the closest sentences in the base (recall that we look for category sequences, not for word sequences), and then proposing the tree corresponding to the sentence found. In average the tree proposed is at a distance of 5.75 nodes from the desired tree. In total, transforming those trees into the exact ones would cost 4 319 operations (clicks and keystrokes).

Compared with the figure of 7 961 operations obtained in the previous subsection, this shows that, when completion by analogy does not deliver an exact result, it is preferable to use approximate matching than trying to adapt by hand a tree proposed by completion by analogy.

4.3.4 Drawing trees from scratch

In this experiment we never need to draw any tree from scratch, because editing a tree obtained by approximate matching is always faster than redrawing the complete tree from scratch.

[1]We took only the first 10 results coming out. We could have taken any number. However, experiments have shown that the increase in quality is not really significant (only 1% increase in the number of exactly parsed sentences, *i.e.* 9 sentences, from 10 to 90 results).

[2]One single click to position the cursor, plus average length of nodes in characters (3.21).

5 Analysis of the experiment

We now summarise the experiment by comparing the number of operations (clicks and keystrokes) in three settings: Using the proposed methods (parsing aids plus tree editor), using a text editor to input parenthesised representations of the trees (1st baseline) and using our tree editor only (2nd baseline). The results are summarised in Table 5. We added the number of times the user has to click menu buttons to activate the parsing aids at each step.

Table 5: Summary

method	nbr of operations	nbr of sentences
menu buttons	$3 \times 1\,553$	
exact matching	0	465
compl. by analogy	1 685	337
approx. matching	4 319	751
proposed method (parsing aids)	(total) 10 663	(total) 1 553
2nd baseline (tree editor)	54 795	1 553
1st baseline (text editor)	88 394	1 553

Clearly, the use of the four steps proposed in Section 2 is extremely benefic: it reduces the number of keystrokes by (54 795 − 11 156)/54 795 = 81% compared with the use of the tree editor alone, and by (88 394 − 11 156)/88 394 = 88% with the use of a text editor to input the parenthesised representations.

6 Future work

The previous computations are worst cases of use of the parsing aids. It would be better to refine by separating into deletions, insertions and replacement to compute the effort more precisely. An insertion is one click plus the average number of characters per node. A deletion is a double click to select the node (or triple click to select a complete subtree) and one keystroke on the `<del>` key. A replacement counts as a double click to select the node (or triple click for a complete subtree), followed by the number of characters in the new node. Also, we did not use copy/paste, which may still reduce the number of operations.

In this experiment, we did not take the time into account. As a matter of fact, changing from the keyboard to the mouse (and vice-versa) takes time. Thus, although we would expect a measure of time to be always in favour of the use of the proposed steps, the gap between the different methods may be reduced from that point of view.

In the future, we want to integrate more the parsing aids together. As a first step, already done, we incorporated exact matching with completion by analogy, so that the user does not need in fact to separate the first two steps described here. A second step is to integrate approximate matching after a failed try with completion by analogy.

7 Conclusion

To speed up tree banking, we proposed a tool, a tree editor with parsing aids. We have shown that this can enormously reduce the number of operations (clicks and keystrokes) needed to augment a treebank of 5000 sentences by 1553 new sentences. The reduction has been shown to be more than 4/5 of the effort.

References

Thorsten BRANTS & Matthew CROCKER
Probabilistic Parsing and Psychological Plausibility
Proceedings of COLING 2000, vol 1, Saarbrücken, July-August 2000, pp. 111–117.

Yves LEPAGE, ANDO Shin-Ichi, AKAMINE Susumu, IIDA Hitoshi
An annotated corpus in Japanese using Tesnière's structural syntax
ACL-COLING Workshop on Processing of Dependency-Based Grammars, Montréal, August 1998, pp. 109–115.

Yves LEPAGE
Open Set Experiments with Direct Analysis by Analogy
Proceedings of NLPRS-99, Beijing, November 1999, pp 363-368.

Yves LEPAGE & Nicolas AUCLERC
A tool to build a tree bank for conversational Chinese
Proceedings of ICSLP 2000, vol IV, Beijing, October 2000, pp. 985–988.