Tagging ICE Philippines and other ICE corpora – Sean Wallis

Initial tagging of ICE Philippines was carried out jointly in Manila and London. The principal tasks here were (a) to get the corpus recognised by the ICE tagger and (b) to build a 'flat' corpus that ICECUP could then browse.

This process was instructive and presented a number of issues to both teams. These are likely to recur with other corpora.

1. First steps

Character set. The tagger expects files to represent characters in 8-bit ASCII, with special characters marked in what we call 'ampersand notation', i.e. "`&<label>;`" so "`&aacute;`" means 'á' and "`&ampersand;`" is '&'. Unfortunately we found the tagger got confused or crashed a few times when it was presented with unexpected character sequences in the input file.

Possible solutions

1) Make the tagger more robust and report these errors clearly.
2) Develop a filter for other character set formats, e.g. unicode.

```
<#293:1:B> <sent> Las Pi&ntilde;as [ICE-PHI S1A-099]

<#19:1:B> <sent> But <indig> Bayan </indig> has also distanced itself from
calls for a coup <foreign> d' &eacute;tat </foreign> saying this goes
against the spirit of People Power [S2B-018]
```

Incomplete markup symbols. It is very easy to fail to type (or to delete) part of a markup symbol or pause. If the final closing angled bracket '>' is absent, the algorithm that identifies these symbols will get confused. It can lead to the tagger reporting an error at some point *after* the failure to close the bracket. Spot the error in the following line in ICE-PHI.

```
<#28:1:B> <sent> When you hear other people talk <{1> <[1> about it </[1>
because people in the Philippines are you know <{2> <[2> <, </[2> they 're
pretentious [S1A-063]
```

Interim solution

I've improved the detection of errors of this kind by limiting the amount of characters read and warning when spaces appear within a markup symbol this should stop more gracefully.

Full header [re]numbering. This is a new **option** in the tagger (switch on with 'C') which constructs full sentence headers, `<#1:1:A> <sent>`, from markup `<$A>`, `<I></I>`, `<X></X>` and `<#>` symbols. It replaces `<#>` markers with the full header.

The availability of this option means that ICE teams do not need to enumerate every sentence header but they do need to note speaker changes, subtext divisions, etc.

Automatic word splitting. This is the mapping: `company's` → `<w> company 's </w>`. This step is also performed in the pre-processor.

Tagging self-correction.
The tagger supports two processes to improve the tagging of self-correction.

1. It marks all 'removed' material as **ignored**. Thus `<-> pros </->` → `(ignore) {pros}`. This material is then skipped in the first phase of tagging.
2. In a second phase, self-corrected material is tagged by looking **forward**, prior to looking up terms in the database. Partial matches apply to incomplete terms. So `<.> pros </.> prospective` copies tag from "prospective" to "pros" (rather than look up "pros", i.e., abbreviation for *professionals*).

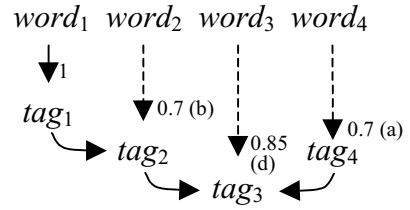

Figure 1: tagging a simple string (scheme)

2. The tagger

The more complex issues are to do with tagging proper. Before we discuss these, it would be useful to briefly summarise how the tagger works and what it currently does not do.

The tagger takes 'sentences' and applies tags in the manner shown in Figure 1.

If the word is a unique lexical type in the database, that is, only one tag exists for that lexical item, then the word is given that tag with probability 1. The tagger then scans the string back and forth, fixing those tags with highest probability first, taking the highest probability value of the following:

(a) the simple probability of a particular tag, e.g. $word_4$ has tag_4 with probability 0.7,
(b) the probability of the tag given a previous wordclass, e.g. $word_2$ has tag_2 given tag_1 before it,
(c) the probability of the tag given a following wordclass,
(d) the probability of the tag given both preceding and following wordclasses, e.g. $word_3$ has tag_3 given both tag_2 and tag_4 with probability 0.85.

This algorithm applies a 'threshold probability' which is reduced after every pass through the sentence. Only tags with a probability higher than that threshold are applied. This is a 'simulated annealing' algorithm which is quite efficient and will order the application of tags to the sentence according to the highest probability at the time. Note that although the probability of tag_3 is higher than the probabilities associated with tag_2 and tag_4, it is dependent on those tags being added beforehand.

The tagger applies some simple morphological rules if the word is not recognised at all. But it does not generalise the database to combine probabilities of, say, all *-ed* verbs in a particular position. This is a potential source of weakness when faced with new texts.

A further source of error is in then handling of ambiguous sequences, where the decision to add one tag is closely allied to the decision to add a neighbouring one, but neither are individually likely. The 'back and forth' algorithm summarised above applies a certain amount of 'look-ahead' to try to resolve these ambiguous sequences.

The database is currently generalised from ICE-GB. It is built in a similar way to the lexicon database in ICECUP, storing references to unique words and additional 'transition' frequencies for lexically ambiguous words. We have a process like the one shown in Figure 2 overleaf. When tagging is carried out on a second corpus, e.g., ICE-PHI, the corrected corpus (shown in bold), whole or part, may then be used to improve the database. The process starts to look more like Figure 3.

Compounds. ICE uses two types of compound tag:
1. **Compound nouns**, typically proper names and titles. Often these are identified pragmatically or by topic, although some simple recognition of common patterns e.g. "`Mr Smith`" is carried out.

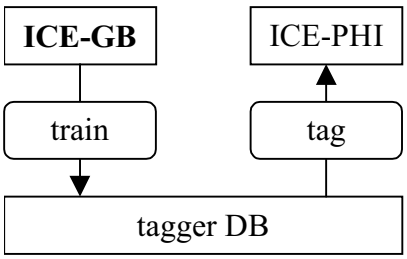

Figure 2: one corpus tagging a second

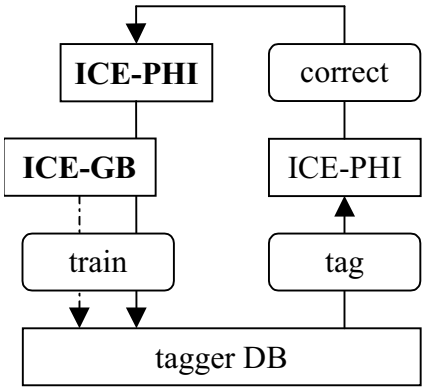

Figure 3: improving the tagging process

2. **Phrasal compounds**, from "`in order to`" (particle to, PRTCL(to)) to "`good morning`" (formulaic expression, FRM).

The tagger includes a simply structured ASCII data file, "ditto.dat" (in the 'icedata' directory), which you can edit with Notepad, etc. The format lists the tag in ICE notation, followed by the strings that should be tagged with that tag, for example:

```
*CONJUNC(subord)
so that
no matter
```

Entries in this file take priority over the database generated from the corpus, and are applied to words with probability 1. You can therefore manually insist that all references to "good morning", **or local equivalents**, are correctly tagged by simply editing this file. (Note: this file can now have single word entries.)

Language. ICE Philippines includes a significant proportion of text in languages other than English.

Where texts include longer sequences of non-English words, or speakers engage in significant code-switching, then we really need to extend the tagger to cope with more than one language. Of course the tagger is only trained (thus far) on British English. Our proposal is to develop alternative tagging databases for different languages integrated in the tagger.

Currently, the lexicon does not differentiate between words in one language or another. What we are proposing is that instead of simply storing the word and its tags, we store a pair of terms, i.e. {word, language}.

Extending the tagger is necessary for two reasons. First, because words are likely to recur, and therefore relying on manual correction alone will be inefficient. We could get around this to some extent by modifying the "ditto.dat" file

Second, as we have seen, the tagger relies on local proximity statistics when it selects between competing tags, and overall accuracy is dependent on the tagging of every word in the sentence. Unless the second language is tagged properly, code-switching can lead to ambiguity in the tagging of the English in the sentence.

In order to build up a basic tagging lexicon of, e.g., the Philippines Tagalog language, some initially substantial manual correction will be required. By default, unrecognised words tend to get tagged as **noun**, hence the uncorrected ICE-PHI contains long strings of Tagalog 'nouns'. When we discussed this, we agreed the following process to 'bootstrap' an ICE-PHI lexicon containing Tagalog.

1) The ICE-PHI team start working by manually correcting the tagging of, e.g. 5 texts (10,000 words or 1%) containing Tagalog and English. Local ICE teams are in the best position to determine the wordclass tag of a local word, whether or not these words are indigenous.
2) The ICE-PHI team then sends the Survey these 5 texts and we train the tagger to retag the remaining 495 texts.
3) The ICE-PHI team can then review the next 5 texts, see how much of the Tagalog material is incorrect, retag these and so forth.

The idea is that by working in this way we bootstrap the tagger and tag the corpus in parallel. Once a corpus has been pushed through the tagger and the corpus indexed for the first time, the problems in section 1 are unlikely to recur, and reapplying the tagger can be carried out quite quickly.

Ultimately we will need to ensure that the ICE teams themselves can rebuild corpora and retrain the tagger but at the moment this process still requires occasional source code adjustment.

3. Editing the tagged corpus

Once the corpus has been initially tagged it will need to be manually corrected. There are two basic ways of doing this. The first way is to edit the corpus files directly. However once this is done, the corpus will have to be 'rebuilt' for it to be viewed in ICECUP. This can also introduce new character set or markup symbol errors in the corpus (see Section 1). Mistakes in typing tags will not be spotted until the corpus is rebuilt.

The best way to work with the corpus is to 'build', i.e. index, the corpus, and **edit the indexed corpus** with ICECUP. ICECUP can be set up to allow the editing of sentences in the tree viewer, and supports multi-seat editing. This uses a system of 'sentence locks' to ensure that the same sentence is not being edited simultaneously by two or more people.

In particular ICECUP provides

- search facilities for any tag, word, combination or 'flat FTF', so that annotators can make corrections to a particular pattern across the corpus,
- text browsing, concordance and search, allowing you to review the annotation
- a browsable lexicon, so that spelling variations etc can be identified,
- online help for wordclass tags.

We recommend that the following tasks be applied in this order.

1) Use the Lexicon to identify *frequently mis-analysed* cases (see below). Make changes to the "ditto.dat" file rather than to the corpus. We can then retag the entire corpus quickly (see 4 below).
2) Use the Lexicon to identify *mis-entered* words and correct them in the corpus.
3) Correct 5 texts (say) manually and then retrain the tagger.
4) Reapply the amended tagger to the remaining texts.

4. Getting started editing with the Lexicon

The Lexicon provides a browsable list of words and their tags.

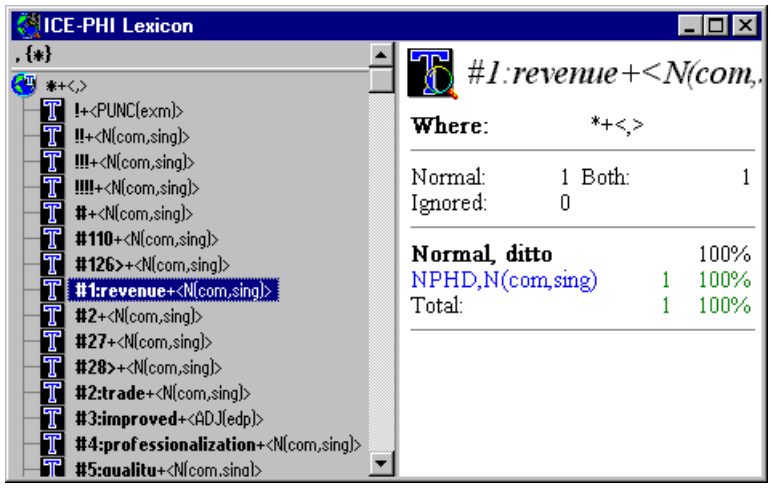

Figure 4: spotting a possible mis-entered 'word' using the lexicon (uncorrected ICE-PHI).

Here we are looking for two types of problem, exemplified in Figure 4 above.

1) 'Words' not in ICE-GB may be incorrectly termed '`N(com,sing)`'.

2) Strings without spaces may be treated as a single word: "`#126>`", "`#1:revenue`", when they should be split into multiple terms, etc.

What should you do with these?

- If a term is frequent and unambiguously wrong, you can edit the "ditto.dat" file to tell the tagger to re-tag it, and then move on, safe in the knowledge that next time the corpus is retagged, the tag will be changed. Despite its name you can add single words to the "ditto.dat" file. These can be removed after the corpus has been retagged.

- If the term is simply formatted incorrectly, you can double-click on the item to find the sentence, edit the tree and correct it.

Figure 5 (overleaf) shows how you can edit the underlying corpus text. Pressing F4 or double-clicking on the item in the Lexicon opens the Query window. This shows the results of the search. Double-clicking on the line opens the Spy window showing the 'tree', in this case an empty head node with four lexical items.

If we wished to edit the tags we could do this with the 'unlock' command (F9). This allows the individual annotator sole control of the sentence and allows him/her to make changes.

In this case we wish to edit the underlying words. We cannot do this directly (this is protected by ICECUP), but you can do it if you 'edit as text'. There is a short cut in the system menu. Click down with the mouse in the top left of the window bar and select '[Unlock and] Convert to Text'.

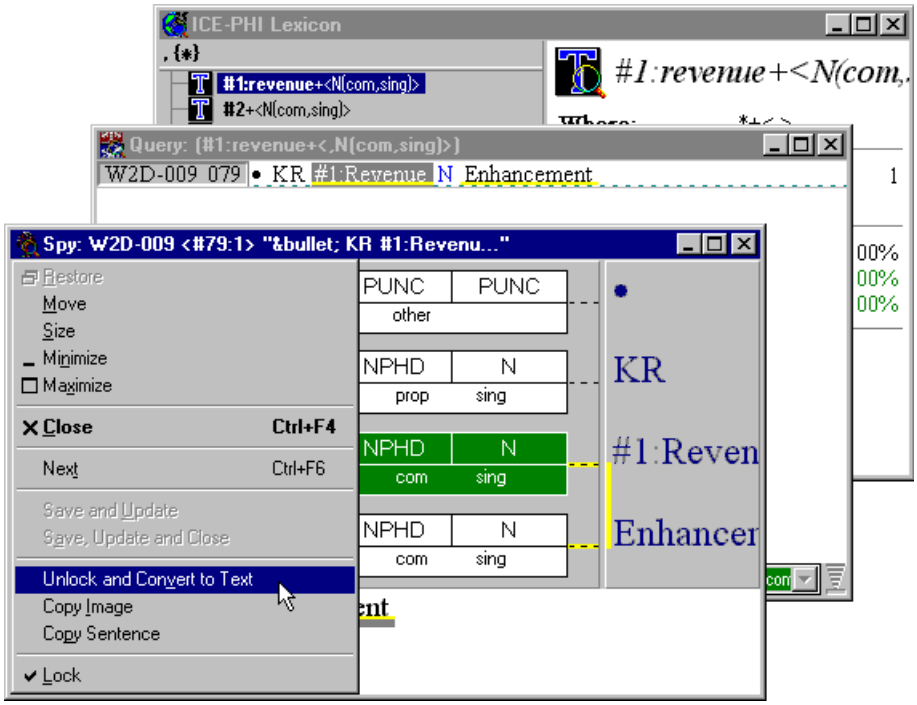

Figure 5: From lexicon to editing the sentence

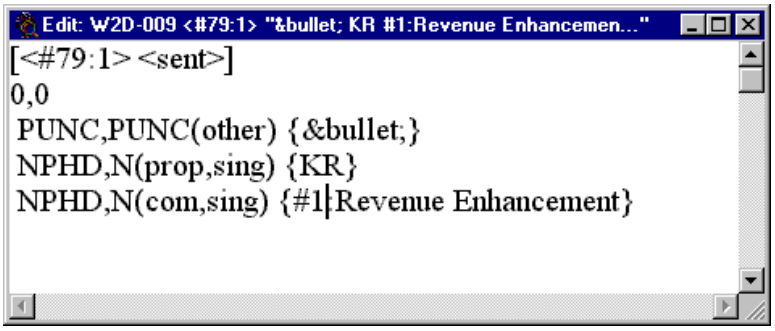

Figure 6: Editing a tree as text.

Alternatively, the 'Convert to Text' command appears under the File menu that appears when you unlock the window.

Editing the text (Figure 6) allows you to directly edit the text version of the representation. Here you need to put spaces in on either side of the colon (above). You can switch back by hitting 'Convert to Tree', which reads the text back in and converts it to the tagged representation.

Note: If you carry out a 'Save and Update' command the corrected file is saved and the query

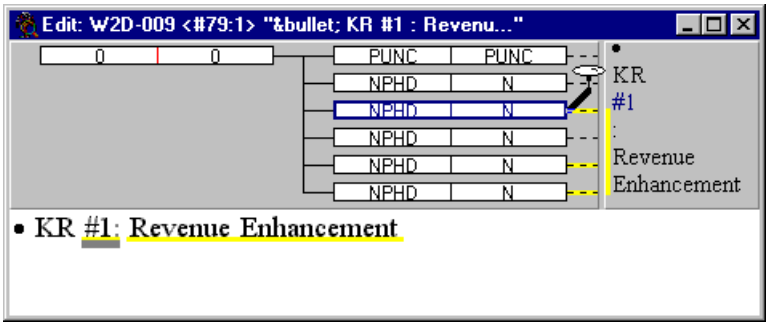

Figure 7: Editing the extent of compounds.

results window will be updated. Note that as a known punctuation symbol, the colon is separated out logically in the sentence fragment, but presented visually adjacent to the previous word in the text browser. (Alternatively, 'Save, Update and Close' will close the edited file.)

In this case we wish to split the compound tags. We can do this using the '**Edit Ditto Tag**' command. If you click on the right hand side of the node you can do this with the mouse.

- If you drag the pin and 'rubber band' to another node you extend the compound (including, potentially, a discontinuous compound) to that node.
- If you drag the pin and 'rubber band' to empty space you break the link to the compound.

Alternatively you can edit sequences with the 'dialog window' for Edit Ditto Tag. (NB. This works in a similar way to Move Node.)

To edit the content of the tag, hit F2. This brings up the Edit Node window (Figure 8). This 'floats' over the screen and you can select any term from the window.

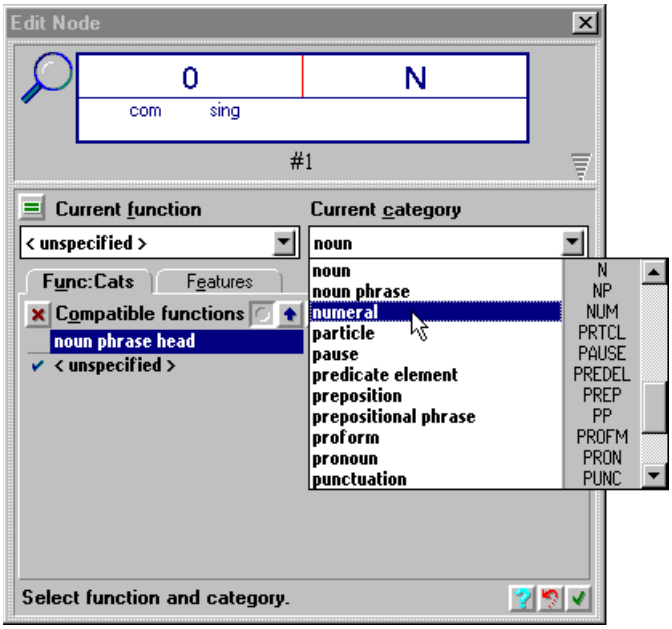

Figure 8: The Edit Node window, editing text.

When you have finished editing you can save the tree.

As we commented, the formatting problem identified may occur elsewhere in the text. To check, you may need to edit other words. Returning to the Query window, click down with the right mouse button in the line to bring up the pop-up menu in Figure 9.

Browsing the context of the query opens the original text in the window. We can view tags inline (Figure 10) with the 'View | Syntax' menu options and buttons. You can then select the other entries, open them and edit them.

Sources of information on the tagging scheme include: ICE-GB itself, Greenbaum's full ICE Tagging manual, the tagging manual built into online Help, and the "red book", Nelson *et al* 2002.

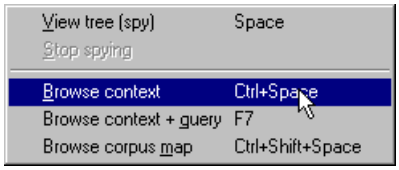

Figure 9: browse popup menu

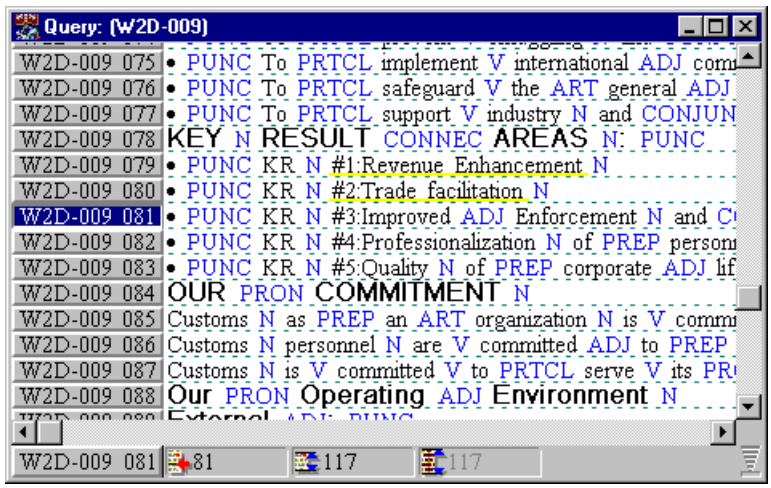

Figure 10: Browsing the text.

Extensions to ease tagging

We have extended ICECUP's editing facilities. I have added the option to pull down a list of possible complete tags (for a given word, taken from the tagger database) with a right mouse click. At the moment this consists of the ten best tags for the individual item but also includes the 'ditto.dat' file. It does not tell you which tags are 'better' in the context but orders them in terms of frequency.

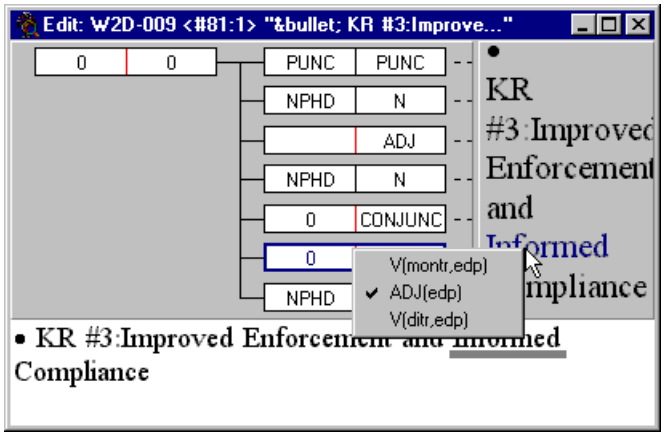

Figure 11: Top 10 Tags – selecting from the tagger lexicon tags.

Ideally this option might be available from the browser window without going into the 'tree window'. Shortcuts of this kind would be particularly valuable for rapid editing.

5. Some technical points about ICECUP editing

Changes made to the corpus with ICECUP do not modify the underlying corpus files. These are effectively read-only, permitting users to 'roll back' to the previously built version of the corpus annotation in that sentence. New changes are placed in 'correction' files, one for each amended sentence, which contain the correction(s).

ICECUP uses some large 'hash table' indexes (stored on the disk) to search the corpus efficiently. The most important are a lexicon and a node index. When you perform a single 'word+tag' search with 'Text Fragment' or a 'Node' search ICECUP looks up and calculates an exhaustive list of all

the cases in all matching sentences. When you do a structured FTF search, ICECUP first computes the set of candidates and then applies the matching process to sift through them.

However if these hash tables were recomputed every time changes were made to the corpus, the program would grind to a halt! (This is because the hash tables store **every combination** of every entry, either directly or through a second order index, see Wallis and Nelson 2000 for the details.)

Instead, the search process is modified so that a list of amended sentences is maintained. These are then searched separately. What this means is that when you search the corpus, ICECUP will exhaustively apply the query to the latest version. But the lexicon, which is based on a 'word+tag' index, will become progressively out of date as words and tags are altered As a rule of thumb, a periodic 'rebuild' is usually necessary every fortnight or so once correction is in full swing.

References

Nelson, G., Wallis, S. & Aarts, B. (2002). *Exploring Natural Language: Working with the British Component of the International Corpus of English*. Benjamins.

Wallis, S.A. & Nelson, G. (2000). Exploiting Fuzzy Tree Fragment Queries in the Investigation of Parsed Corpora. *Literary and Linguistic Computing* **15**.3, 339-361.