Design and implementation of the AGTS Probabilistic Tagger

Hong Liang Qiao
Norwegian Computing Centre for the Humanities

Renjie Huang
Jiao Tong University

1 Introduction

The last ten years have seen the development of several probabilistic taggers, such as the PARTS program (Church 1988), the De Rose tagger (De Rose 1991), the Brill tagger (Brill and Marcus 1992) and the Xerox tagger (Cutting et al 1993). The AGTS (The Automatic Grammatical Tagging System) Tagger was a key project funded by China Social Science Academy. It was undertaken at the laboratory for Computational Linguistics of the Institute for Natural Language Processing of Jiao Tong University in Shangha, from 1987 to 1990. The basic techniques of the AGTS tagger are very similar to the CLAWS tagger in terms of the principles of Constituent Likelihood Grammar (Atwell 1987). One the of purposes of the AGTS project is to tag the JDEST (Jiao Da English for Science and Technology) Corpus, which consists of one million words. The JDEST Corpus was built up by Jiao Tong University in Shanghai in 1985. The JDEST Corpus collected English texts covering ten subject areas in science and technology: physics, nuclear energy, metallurgy, computer science, aeronautics, mechanics, electrical engineering, chemical engineering, architectural engineering and shipbuilding. The texts were randomly selected from theses, textbooks, academic works, popular science and science digests, published in the United Kingdom, the United States and other countries. The corpus consists of 2,000 units of about 500 words each. The size of the JDEST Corpus is over 1,000,000 words. The sizes of the units and the entire corpus are very similar to those of the Brown Corpus of American Written English. The AGTS has experienced changes of its programming languages from BASIC to SNOBOL, and then to Turbo C.

2 The Structure of the System

The general structure of the AGTS System is made up of two parts, namely the data and the programs. The data are in fact the corpus, tags, a machine dictionary, rules and a probabilistic matrix database. The programs control the processes step by step. They retrieve, recognise, compare and judge the data and output from raw texts to tagging results. The following figure depicts the general structure of the AGTS System:

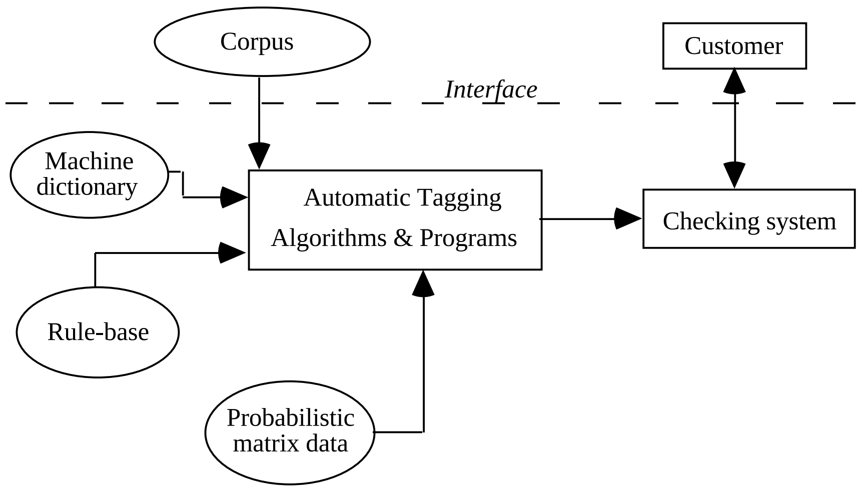


Figure 1: General structure of AGTS system

3 The AGTS Tagset

The tags in the AGTS Tagset are basically similar to the tags used for the LOB Corpus. The reason for adopting LOB tags is that it makes it easier to compare the results of the research with the results of the LOB Corpus tagging project. It is not only beneficial to evaluate the results and efficiency of the AGTS System later on, but it is also helpful for comparative research in contrasting corpus linguistic studies between

English for science and technology purposes and English for more general purposes. In the LOB Corpus there are 130 tags, compared to 131 in the AGTS system. Most of the tags in the two tagsets are the same. However, some changes were made; for instance, the Lancaster tag ‘XNOT’ was changed to ‘NT’ in the AGTS tagset. In treating idioms or special phrases, the Lancaster method uses one of the major tags as labels to all tags and also attaches serial numbers to the tags. The AGTS System keeps the original word class of each tag in the idiom. For example, the determiner phrase *a few* is tagged by CLAWS ‘AP21 AP22’, and ‘AT AP’ by AGTS. In ‘AP21 AP22’, ‘AP’ means post–determiner. ‘21’ and ‘22’ respectively indicate the first element of the two and the second element of the two. ‘AT AP’ indicates a singular article followed by a post–determiner.

Each tag in the AGTS tagset consists of one to five characters and/or numbers, eg ‘.’ for a full stop, CD for cardinal numbers except ‘one’, JJT for a superlative adjective, NN$ for a possessive plural noun, and PP3AS for a third person plural pronoun. The 128 tags in the AGTS tagset are divided into three categories: 1) part–of–speech tags, 2) punctuation tags, and 3) special tags.

3.1 Part of Speech Tags

Part of speech tags comprise the main share of the tags, with a total of 114, which occupies about 87 per cent of the total. These tags are represented by two to five characters and/or numbers. Compared to traditional categories, they are divided into more detailed sub–categories. With personal pronouns, for example, the tags are divided into ten different kinds:

PP1A (*I*), PP1AS (*we*), PP1O (*me*), PP1OS (*us*);
PP2 (*you*);
PP3 (*it*), PP3A (*he, she*), PP3AS (*they*), PP3O (*him, her*), PP3OS (*them*).

For verbs, not only are the tags divided into BE(be), HV(have), MD (modals) and VB(verbs), etc, but also suffixes are attached, such as Z in VBZ for the third person singular present, D in VBD for the simple past, N in VBN for the past participle, and G in VBG for the present participle.

Tags in certain groups of categories can be recognised by either prefixes or suffixes. In the tags listed above, all personal pronouns take the prefix ‘PP’, the suffix ‘S’ means plural, ‘Z’ stands for third person singular and present tense, and ‘A’ indicates subject case and ‘O’ object case.

3.2 Punctuation Tags

We generally use the punctuation marks themselves as punctuation tags, eg ‘.’ (not including the quotation marks) for full stop, ‘;’ for semi-colon, ‘!’ for exclamation mark, *’’ for opening quotation mark and **’’ for closing quotation mark. A single asterisk indicates opening quotation mark, while a double asterisk indicates closing. There are 13 punctuation tags, which usually consist of one to three characters.

3.3 Special Tags

There are only four special tags, which consist of two to three characters. `&FO` means ‘formula(e)’, `&FW` ‘foreign word(s)’, `ZZ` English letters and ‘---’ is a sentence separating tag, which is a tag that is not included in the LOB tagset.

Tags are normally attached to individual words, rather than to a pattern or an idiomatic phrase consisting of more than one word. On the whole, this resembles the practice of the CLAWS tagger:

in order to `IN NN TO`

In this way the word *order* will not be treated as a verb, which is one of its possible tags (*order* can be a noun or a verb according to different situations). *In* and *to* will also not be confused by taking adverb and preposition, respectively, as other possibilities. These idiomatic phrases are stored in a special dictionary to avoid unnecessary ambiguity.

4 Grammatical Data for Tagging

A correct part of speech (POS) tag assignment to each word in a running text requires several types of grammatical analysis. In AGTS, a machine dictionary, a second-order Markov grammar rule file for disambiguation and probabilistic matrix data have been designed to cope with different tasks in tagging.

4.1 Machine Dictionary

The machine dictionary includes a special wordlist, a prefix list and a suffix list. The most frequent 12,000 English words in the JDEST Corpus are taken as entries in the machine dictionary. These words cover more than 97 per cent of the corpus.

4.2 Suffix List

The grammatical identity of a word is closely related to its suffix. Therefore, suffixes can be useful in judging the part of speech of a word. For example, words with the suffix – *tion* are most probably nouns, with rare exceptions such as – *mention*, which can also be a verb. Suffixes like – *est* can very often be superlative adjectives, eg *best*, *fastest* and *earliest*, of course, with exceptions like *rest*, *interest* and *request*. Words with the suffix – *able* are very often adjectives. If a word ends with – *ly*, the word is most likely an adverb. Altogether, there are some 600 suffixes in the suffix list. Besides normal suffixes like – *ment*, – *wide* and – *or*, which are supposed to be traditional suffixes described in morphology, the suffix list also includes strings which can function as word terminations, although, strictly speaking, they are not suffixes according to word-formation theory. In other words, suffixes in the suffix list as defined here are different from the suffixes in the sense of conventional morphology. As a matter of fact, they are treated as word-ending strings. POS tags are then attached to the suffixes in the suffix list according to a study of the most likely word-classes that belong to them, eg:

rm (NN, VB) – *storm*, *term* and *form*

For instance, *rm* is linguistically not usually seen as a suffix. However, computationally it is helpful to suggest words as singular noun, or present and plural (ie not singular) verb.

4.3 Special Wordlist

If the tag(s) belonging to a word's suffix are exactly the same as the tag(s) of a certain word in the dictionary with exactly the same suffix and the same tag(s), then the word is deleted from the dictionary, so that the words left in the dictionary form a special wordlist.

For instance, here is a suffix list and a dictionary:

Suffix List	Dictionary

Suffix	Tags	*No.*	*Tokens*	*Tags*
–ed	VBD VBN	(1)	*bed*	NN
		(2)	*indeed*	RB
		(3)	*led*	VBD VBN (deleted)
		(4)	*need*	VB NN MD
		(5)	*seed*	NN VB
		(6)	*worked*	VBD VBN (deleted)

Figure 2: Suffix list and a dictionary

Only *led* and *worked* meet the criterion for a past tense verb, so, they are deleted, even though the *ed* in *led* is NOT a genuine suffix. The special wordlist contains about 4,000 types, with a maximum of five possible different tags for an individual word.

There are three types of errors that qualify a word as an entry in the special wordlist – erroneous (totally mismatch), superfluous (containing extra tags) and omitted tags (not containing the required tags). Below is an example:

words	suffixes	tags assigned by suffixes	actual possible tags	error types
mention	*-tion*	NN	VB NN	omitted
explosive	*-sive*	JJ	JJ NN	omitted
converter	*-ter*	NN JJ	NN	superfluous
describe	*-be*	NN VB	VB	superfluous
friendly	*-edly*	RB	JJ	erroneous
flee	*-ee*	NN	VB	erroneous

Figure 3: Types of errors in a dictionary for establishing a special wordlist

If a word ending in *-tion* has both VB and NN as its part-of-speech tags in the dictionary, then NN is left out, leaving 'the tags assigned by suffixes' not indentical with 'the actual possible tags'. In this case, that word becomes an entry in the special dictionary. The same happens to the superfluous (contrary to the omitted type) and erroneous tags.

Most function words, eg *would* and *could*, cannot be tagged according to the suffix list. Therefore they are included in the special wordlist.

4.4 Prefix List

The prefix list contains a list of prefixes for the purpose of a second special wordlist look-up. The prefix list is constructed on the basis of morphology. For example, if the word *Sino-Anglo* cannot be located in the special dictionary, while the word *Anglo* is present, *Anglo* can then be matched, if the prefix *Sino* is removed. Therefore the search is successful. The prefixes mentioned here, like those suffixes mentioned in 4.3, may not be regular morphological prefixes.

4.5 Context Frame Grammar Rules for Tag Disambiguation

The Context Frame Grammar Rules (CFGR) are designed according to first- and second-order models Markov Models. For example:

```
1. A  +  B  +  ?  +  C  +  D  -->   Y
2. A  +  B  +  ?  +  C  +  D  -->  ~Y
```

Rule 1 means that if an undetermined word (in the form of a question mark in the above examples), ie a word with multiple possible tags, is between tag A and tag B on the left and tag C and tag D on the right, and Y is one of the possible tags, then Y is selected. Rule 2 means that, under the same conditions as in Rule 1, tag Y should be excluded or deleted from the possible tags. We call the rules of type 1 positive rules and those of type 2 'negative rules'. Of the four known items A, B, C and D, up to three can be omitted, so that six types can be derived:

Rules	Explanations

1. First Order Pre-Position Rule eg `AT- + ? -> ~VB-`	After an article of any kind, all verb tags are deleted.
2. Second Order Pre-Position Rule eg `CD + JJ + ? -> NNS`	After a cardinal number except *1* (whose tag is `CD1`) and an adjective, the tag to be selected is NNS (plural noun), if it is one of the possible tags.
3. First Order Post-Position Rule eg `? + VB -> MD`	If the tag `MD` is among the possible tags and is directly before a verb in its original form, then `MD` is selected.
4. Second Order Post-Position Rule eg ? (*such*) + (*a*) `+ NN -> AP`	Before *a* and a singular noun, if the word is *such*, then the tag `AP` (post-determiner) from the possible tags is selected. The lower case words in round brackets are literal words.
5. First Order Mid-Position Rule eg `IN + ? + . --> NN-/VBG`	If a tag is between a preposition and a full stop, the tag can either be a noun or a present participle of a verb.
6. Second Order Mid-Position Rule eg (*as*) + ? + (*as*) + (*possible*) - >RB/JJ	If a tag is located between *as* and *as possible*, it can either be a general adverb or an adjective.

4.6 Probabilistic Matrix Data

By deriving statistical data from the tagged LOB Corpus, we constructed a probabilistic matrix to account for the probability of a given tag appearing after another. Supposing that there are two adjacent tags A and B in the corpus, we can calculate the probabilistic value of tag B after tag A. The formula is as follows:

Formula 1: Probabilistic value of the adjacent tag

$$P = \frac{nAB}{NA} \times 1000\_$$

NAB = number of occurrences of tag B immediately after tag A in the whole corpus.
NA = number of occurrences of tag A in the whole corpus.
P = the probabilistic value, set to _.

According to the matrix, the P values of NN, JJ, RB, QL and JJT preceded by AT (the singular article, eg *a*, *an* and *every*) are as follows:

Table 1: Probabilistic values of NN, JJ, RB, QL and JJT preceded by AT

NN	(singular noun)	526.4_
JJ	(adjective)	369.9_
RB	(general adverb)	19.9_
QL	(degree adverb)	15.6_
JJT	(superlative adjective)	9.7_

Note that LOB is a collection of British English texts of different types. The matrix retrieved from LOB may therefore not be one hundred per cent appropriate when applied to the tagging of JDEST Corpus, which is a corpus of English texts purely for science and technology purposes.

5 The Design and Implementation of the AGTS System

Linguistic principles and computational algorithms for the system design and implementation are divided into input text preparation, tag assignment, rule matching and probabilistic finalising of the tags.

5.1 Input Text Preparation

The input text files for AGTS has a file name with .lib as its extension. Each text in a file should have a text serial number at the beginning. The following is a text in T1.lib of the JDEST Corpus:

```
T0001TP3A1972TE
Formulation of the Logic Design Problem.
The general procedure followed in any design
task starts by defining a behavior and other
specifications or characteristics of the system
to be designed. The latter are given as a
set of limiting values for the parameters
defining the system’s performance. For example,
if the task is to design a binary adder
(binary addition represents the behavior in this
case), the limiting factors could be: speed
less than 100 nanoseconds, operand length 32
bits, cost less than X dollars. With this
information the designer proposes an organization
of the system consistent with the given speci-
fications. He may, for example, propose in this
case of an adder design (discussed in detail
in Chapter 3), a serial version composed of
a 32 bit register, a full-adder, and a carry
flip-flop.
```

The first line of a text is the text serial number and the second line is the title of the text. However, not all texts have a title in the second line. The rest is the body of the text. A new paragraph always starts with two spaces at the beginning of a line. There is no blank line between texts and paragraphs.

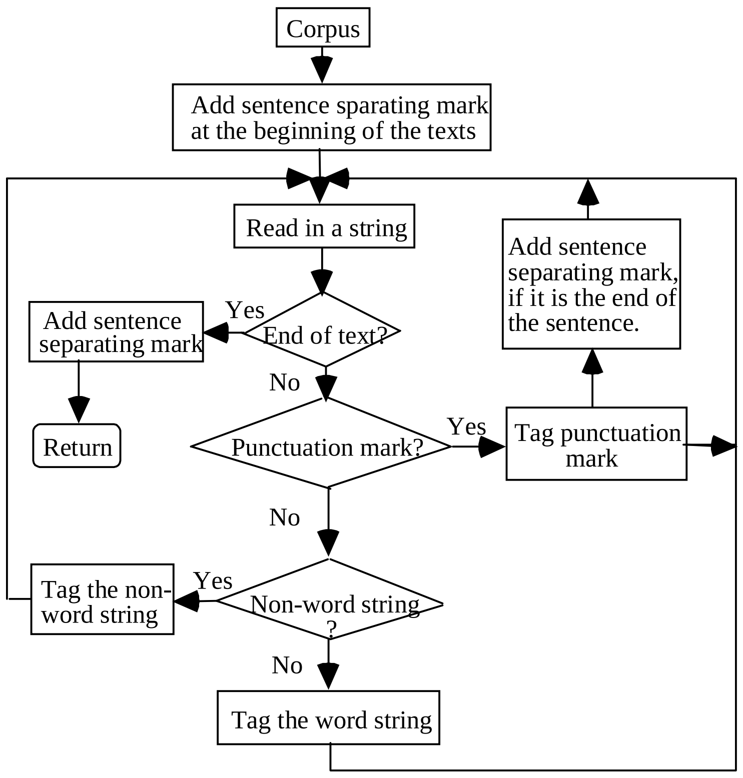
5.2 Assignment of Possible Tags by Checking Wordlists

In phase 1, authentic texts (corpus texts and other input) are first segmented into sentences and then into words. Possible tags are assigned to words or other items according to the special wordlist, prefix list, suffix list and special processing, like capitalised words processing and

S processing. In this phase, no contextual information is involved.

Figure 4: Assignment of possible tags by checking wordlists

This flow chart above should be interpreted as follows:
AGTS starts to read from the beginning of a text and inserts a sentence



separating mark (indicating the beginning of a sentence and a text).
Next, the program reads a string and tests whether this is the end of the text. If it is, another sentence separating mark is attached at the end of the text, which acts as a text and sentence ending mark, and

phase 1 finishes. 'Return' in the flowchart means an end of the current loop and the beginning of the next, if there is one.

However, if the string does not indicate the end of the text, the program will check whether the string is a punctuation mark. If the string is a punctuation mark, the program uses a special subroutine to check if it is a sentence ending mark. If the string is the end of a sentence, then a sentence separating mark will be inserted and the program will read in a new string. If the string is not the end of a sentence, the program will attach a punctuation tag to it and read the next string.

If the string is not a punctuation mark, the program checks if it is a non-word string. If it is a non-word string, it is tagged with a relevant tag; else it must be a legal word and the system tags it with possible tag(s), before reading in the next word.

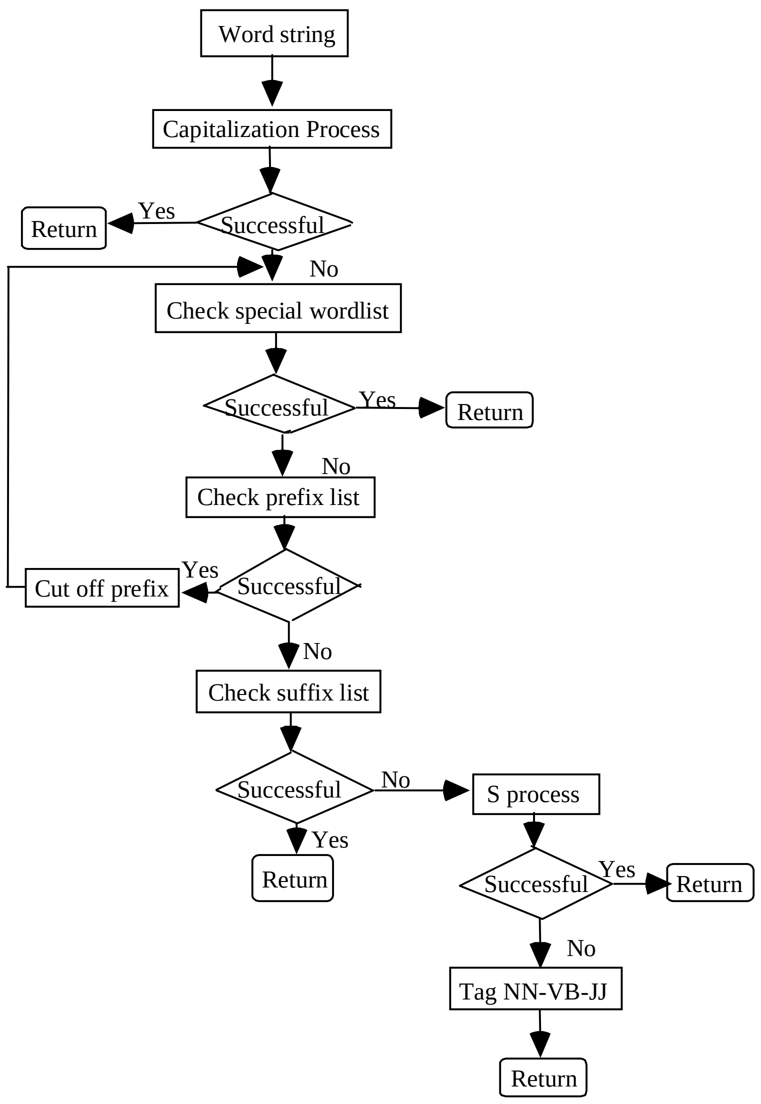
The AGTS System attaches a proper tag to every English word, punctuation mark and other strings. Since many English words play different grammatical roles in different contexts, it is possible that a word will take more than one tag. These are called 'possible tags'. For instance, *work* has two possible tags (NN and VB); *present* has three possible tags (JJ, VB and NN); *past* has four – (JJ, NN, IN and VBN); *round* has five – (JJ, NN, VB, RB and IN). At this stage, we supply words with all possible tags through checking against wordlists, without consideration of or interference from the context. This phase is the assignment of possible tags. The suffix list, prefix list, special list and pattern list provide the check data.

Now consider Figure 5. First, if the string is capitalised, then the program will process it with a special processing program. Capitalised words are checked to see if they are proper nouns or ordinary nouns, as in a title or the beginning of a text, so that they can be further processed in the tagging:

1) Capitalised words such as *Australia* and *Chinese* can be found in the special wordlist and are then tagged according to that list.
2) Capitalised words at the beginning of the sentence are recognised according to sentence separating marks and then converted to lower-case to be processed.

Figure 5: Tagging the word string

3) Capitalised words in titles, such as *Logic*, *Design*, and *System* in *Formulation of the Logic Design System* will treated as ordinary nouns.



4) Fully capitalised words, such as *FORTRAN* and *CPU* will be tagged as proper nouns.

The program then checks the special wordlist. If it finds an exact match, it directly assigns possible tag(s) to it; otherwise, it checks against the prefix list. The prefix is removed and then the word is tagged according to the special wordlist again. If the tagging fails again, tags are assigned according to the suffix list. Sometimes, when both VBZ and NNS occur as tags of a word, it is processed it with the S sub-program to disambiguate the two. If the tagger fails in all the matchings, the program tags the word with a default tag (NN-JJ-VB). This default tag is an artificial one, which consists of three ambiguous tags. The default tag is created on the basis of the three top frequency tags in the LOB Tagged Corpus.

Certain tags occur rarely and are labelled with rarity symbols, of which there are two kinds. One is '@', which shows that the frequency of the tag is less than ten per cent, and the other is '%', marking a frequency of occurrence of less than one per cent. For example, *pressure* is tagged with NN and VB@, which indicates that the possibility of VB is less than ten per cent, and there is a more than 90 per cent chance that the word is NN. Another example is *even*, which is tagged with JJ, RB and VB%. Though *even* can be a verb, it functions as a verb in less than one per cent of all cases.

The use of the suffix list of the AGTS System follows a length-priority principle. Consider the following three suffix items:

-*le*	NN
-*ble*	NN VB
-*able*	JJ

When the program is running, it looks for the four-letter suffix -*able* first. If it matches the grammatical information of the word in the dictionary, then JJ is assigned. If it fails, then the three-letter suffix -*ble* is compared. NN and VB are assigned, if they match the word's grammatical information, and so on.

5.3 Disambiguation by CFGR Rules

After the first two steps, possible tags are attached to the words, among which only one third have two or more than two tags. That is to say, 66 per cent of the words in the running text are labelled with an unambiguous single tag. The Context Frame Grammatical Rules (CFG) are supposed to disambiguate those still carrying multiple tags. The CFG rules are based on two principles, which restrict and affect each other:

1) The CFG rules should create as many rules as possible, so that the rules can cover a higher percentage in tag processing. In other words, the rules are expected to do most of the work in tag disambiguation.
2) When a new rule is created, it is expected to provide a good control of leakage of the rules (ie the exceptions, which cannot be covered or captured by general rules). Only when the rule meets these requirements can it be collected in the CFG rule-base.

For example, consider a noun phrase with the following two rules:

```
noun phrase: a book      a = AT       book = NN VB
rules: a) AT- + ? --> NN              b) AT- + ? --> ~VB-
```

Rule a) means that, after an article, the tag will be a noun if NN (noun) is one of the possible tags, while rule b) means that, after an article, verbs of any kind should be deleted from the tags that are attached to that word. When we delete `VB`, the only tag left is `NN`. The results of the two rules are the same, but when we look at another example with two rules, things could be different:

```
noun phrase: a made story          made = VBD VBN JJ
rules: a) AT- + ? --> NN           b) AT- + ? --> ~VB-
```

Rule a) will not fire, because it cannot find an NN in the tags. Fortunately, the rule will not make any mistakes in this case. Rule b) seems to be safer by deleting `VBD` and `VBN` and leaving `JJ` as the single correct tag. If rule b) is processed first, it will be more appropriate and efficient, because when rule b) is checked and `JJ` is assigned as the single tag, there is no need to go through other CFG rules. Therefore, this saves computation load and time. In this particular case, negative rules are more efficient than positive rules. Let us examine another example:

```
noun phrase: a Chinese text          Chinese = JNP NN
```

rules: a) `AT- + ? --> NN` b) `AT- + ? + NN- --> JNP`

Rule a) considers only one tag before the target tag, and the rule's contextual environment is too small to make a better judgement than b), which considers one tag on each side of the target tag, and is, comparatively speaking, more powerful.

Three main methods are applied to control leakage from the rules (ie the exceptions):

1) If more known items are added to the rules, more contextual information can be applied to the rules. This can make the rules more accurate and reduce leakage significantly. This indicates that second order-rules are more powerful than first-order ones.
2) While tagging, the program will check rules by priority. If those rules are able to solve the problem, other similar rules will not be called, as in the case of 'a made story'. The rules are not listed in alphabetical order or by the length of the rules. Priority is given to rules with a longer context. Second-order rules are applied before first-order rules. Among the first order rules, mid-position rules are executed before pre-position and post-position rules. The longer the context, the more the known items, and the higher the reliability in tag selection. This principle enables the program to deal with exceptional cases.
3) Individual tag exceptions or pattern exceptions (eg a phrase containing several tags) to the rules will be stored in a special wordlist and each word will be assigned a single tag to avoid further processing by CFG rules.

The judgement made by the rules may or may not be effective. The solution can also be single or multiple tags. What should be emphasised is that a chosen tag must be a possible and existing tag to the word, and there can be more than one. If the rules cannot choose any tag(s) for a word, and even if all other known items are exactly the same as in the rules, the rule will not function. For example, if the pre-position rule '`AT- + ? --> NN`' is applied to a phrase like *a linguistic seminar*, in which the word *linguistic* is only labelled with a '`JJ`' (adjective) tag, the rule cannot find an '`NN`' to match. Therefore it fails to work. In other words, the rule was programmed to choose an '`NN`' from possible tags, but unfortunately '`NN`' is not available.

If a rule selects or deletes certain tag(s), leaving a single tag for a word, this indicates the best performance of the rule. If there is more than one tag left, the rule will be re-used until no further tags can be deleted for the word.

Figure 6 shows the results from phase 1, which assigns all possible tags by checking several kinds of wordlists. The system now reads in a sentence and then checks if it is at the end of the text. If the word is at the end of the text, the program stops; otherwise it checks if the current word is marked with a unique tag. If the word is marked with a unique tag, the system checks if it is the end of the current sentence. If it is, the system reads in the next sentence. If not, it reads in the next word.

If the word is tagged with more than one tag, the pattern list is checked first. If it can be found in the pattern list, then the tags will be assigned to the pattern. The program will now check if it is the end of the sentence. If yes, it will read in another sentence. If not, it will read in the next word.

If the word cannot be found in the pattern list, then the second order rules will be checked. If this fails, the program will check mid-position rules, pre-position rules, post-position rules and special rules (a small number of rare rules that are not found in other rules) until the rules are exhausted. If one of them matches, then are disambiguated the tags and update the tagging results are updated. It should always make sure whether the string is at the end of the sentence, no matter whether the disambiguation is successful or not.

Through the tag selection by rules, the proportion of words with unique tags rises to around 89 per cent, an increase of 23 per cent from the figure before CFG rule selection. Checking reveals that errors are no more than 0.3 per cent by rule disambiguation.

The following flowchart is a summary of the CFG rule selection process:

5.4 The Probabilistic Selection

We have found differences in the frequency of the application of the CFG rules. With further observation, somewhat unexpectedly, we noticed that first order rules are of high frequency: 80 per cent of the tag disambiguation judgements were made by these rules. The most frequently used rules have the least contextual information. This phenomenon

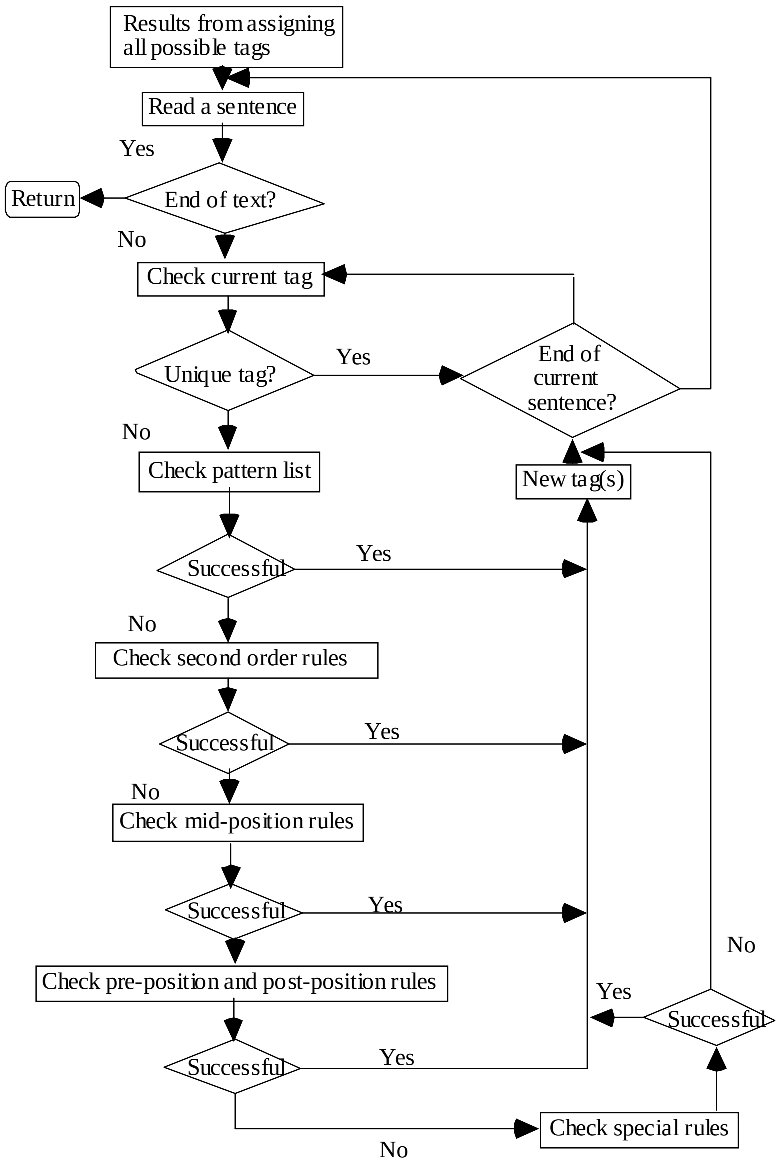


Figure 6: CFGR rule selection

indicates that the information of an adjacent tag actually plays the most important role of all. This proves that the Constituent-Likelihood Grammar based on the simple Markov first-order model is extremely powerful in assigning part-of-speech tags.

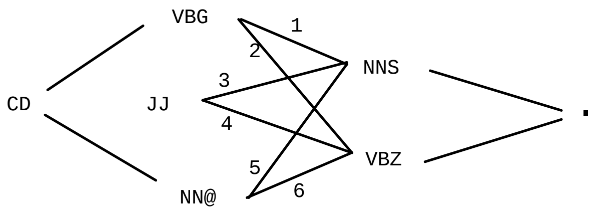
The probabilistic matrix reflects the statistical relationship of two adjacent tags. Therefore, the matrix can be successfully applied only when adjacent words have only one tag assigned to them. However, in practical processing, ambiguous or multiple tags to words do occur. The number of words with ambiguous tags in a sequence is normally no more than five. When they appear, the method to deal with them is as follows: first, all different possible combinations are listed; then the probabilistic values are retrieved from the matrix data and the multiplication values of the different combinations can be calculated. The tags with the highest multiplication value will be selected. For example, the vertical display of the sentence *There are four cutting tools* is as follows:

---	---
There	EX
are	BE
four	CD
cutting	VBG JJ NN@
tools	NNS VBZ
.	.

In this sentence, the words *cutting* and *tools* failed to obtain a single tag after the rules were exhausted. The word *four* before the ambiguous words and the full stop after them has a single unambiguous tag. In this case, the system first had to find the paths or possible combinations of the four tags. They are:
Figure 7: Paths of tags

That is to say, the tags can be arranged into six different paths according the following formula:

Formula 2: Tag transition value



$$Val = p_1 \times p_2 \times ... \times p_n$$

p_1, p_2 and p_n are the probabilistic values of up to n pairs of adjacent tags. According to Formula 2, we can calculate the probabilistic value of the six paths:

1. Val	(CD-VBG-NNS-.) =	1.1 x	32.2 x 133.7 =	4735.7
2. Val	(CD-VBG-VBZ-.) =	1.1 x	0.4 x 30.2 =	13.3
3. Val	(CD-JJ-NNS-.) =	41.8 x	175.4 x 133.7 =	879655.1
4. Val	(CD-JJ-VBZ-.) =	41.8 x	0.3 x 30.2	378.7
5. Val	(CD-NN@-NNS-.) =	26.3 x	(13.1 x 0.5) x 133.7 =	23031.8
6. Val	(CD-NN@-VBZ-.) =	26.3 x	(13.1 x 0.5) x 30.2 =	5202.4

Val =	probabilistic value of path.
@ =	symbol for rarity, where extra 0.5 the value in the matrix must be multiplied to reduce the tag's chance of being selected.
% =	symbol for rarity, where extra 0.125 the value in the matrix must be multiplied to reduce the tag's chance of being selected.

The highest value is 879655.1, so the combination of group three is selected. The tag for *cutting* is JJ and for *tools* NNS. We then come to a finalised sequence of tags as follows:

--- ---

There	EX
are	BER
four	CD
cutting	JJ
tools	NN
.	.

All other combinations, with much smaller chances of success, are abandoned. The tagger can provide a list of combinations with descending probability values to show the likelihood of each of the combinations.

6 A Tagging Experiment

The 100 sentences collected for the test are from 20 different citations. They cover five different genres, ie news reportage (category a), biography (category g), governmental documents (category h), scientific writing (category j) and novels (category k). Altogether 2,202 strings are found in the text, including 2,006 words and 196 punctuation marks. The number of errors found are listed below:

Table 2: The index of the types of tagging errors

Wrong tag	Correct tag	Count	Index
" "	NNS	1	h08
-	NN	4	g07 g12 g12 g15
AP	PN	1	a15
AP	QL	1	g12
AT AP	AP21 AP22	4 (=2*2)	a04 j19
BEDS	BEDZ	16	a05 a07 a10 a12 g01 g02 g07 h17 k02 k07 k08 k09 k09 k13 k14 k16

CD	OD	1	g19
CS	IN	1	h12
CS	WP	1	j12
DT	CS	1	h17
IN	TO	1	h09
IN NN	NNU21 NNU22	6 (=3*2)	a13 a14 j20
IN NN TO	TO31 TO32 TO33	3 (=1*3)	j06
JJ	IN	1	a07
JJ	NN	1	g07
JJ	RB	1	k01
JJ	VBG	1	a04
JJ	VBD	1	a06
JJ	VBN	1	j15
JJT	NN	2	a16 a17
NN	AP	4	a13 j15 j16 j20
NN	JJ	1	a20
NN	NP	6	a07 a15 a15 h05 j12 k03
NN	NR	1	h07
NN	RB	1	h04
NN	VB	1	h09
NN	&FW	3	g06 g16 g16
NN	RP	1	g07
NNP	NR	3	g01 k11 k15
NNS	NP	1	a10
NNS	UH	1	k10
NNS	VBZ	2	g09 g14

NT	XNOT	8	a08 a19 j08 j15 k14 k15 k19 k20
PN	CD1	1	a13
RB	ABN	1	k15
RB	AP	1	h12
RB	ATI	1	h12
RB	IN	1	k11
RB	NN	1	a13
RB	QL	2	g16 g19
RB	RP	1	j18
RI	IN	1	g09
VB	JJ	1	j01
VBD	VBN	2	a03 k07
VBG	JJ	4	g12 h16 j17 k12
VBG	NN	3	h08 h01 k03
VBN	VB	1	a05
VBN	VBD	3	j18 k03 k18
VBZ	NNS	1	j01
WP	WDT	1	g13
Total	50 type pairs	108 tags	101 citations

According to the CLAWS standard, the tagging success rate is 95.09 per cent, with an error rate of 4.91 per cent (108/2202). Some of the errors are caused by incompatibility of the tagsets between the two corpora, eg NT (AGTS) vs XNOT (CLAWS), NN (AGTS) vs &FW (CLAWS), BEDS (AGTS) vs BEDZ (CLAWS), and idiomatic patterns such as AT AP (AGTS) vs AP21 AP22 (CLAWS), IN NN (AGTS) vs NNU21 NNU22 (CLAWS) and IN NN TO (AGTS) vs TO31 TO32 TO33 (CLAWS). Without these errors, the success rate can be as high as 96.91 per cent (68/2202). These errors can easily be avoided by adding information to the current versions of special wordlists and/or pattern lists in the AGTS System and changing those part-of-speech tags to conform with the LPC or LOB tagset.

Table 3: The distribution of tagging errors

No. \ Text	a	g	h	j	k
01	0	2	1	2	1
02	0	1	0	0	1
03	1	0	0	0	3
04	3	0	1	0	0
05	2	0	1	0	0
06	1	1	0	3	0
07	3	4	1	0	2
08	1	0	2	1	1
09	0	2	2	0	2
10	2	0	0	0	1
11	0	0	0	0	2
12	1	4	3	2	1
13	5	1	0	0	1
14	2	1	0	0	2
15	3	1	0	3	3
16	1	3	1	1	1
17	1	0	2	1	0
18	0	0	0	2	1
19	1	2	0	2	1
20	1	0	0	3	1

```
Total =108   28 tags 22 tags 14 tags 20 tags 24 tags
tags
```

There are 39 fully correct sentences (sentences with the value 0 in the above figure) vs 61 erroneous sentences. However, if the six types of errors resulting from the incompatibility of the two tagsets are not counted, 15 sentences can be added to the number of the fully correct sentences, making the total number of the correct sentences 54 out of 100.

In any case, for the parsing, there is a need for manual post-editing of the tagged sentences as an output from the tagging by the AGTS System, because of the tagging errors and some typographical errors due to the slight difference between the tagsets of the LPC and the JDEST Corpora.

7 Conclusions

The tagging success rate of the AGTS System is above 95 per cent, according to the LOB tagset, subject to the difference of subjects and types of texts processed. Its average correction rate is over 96 per cent, according to its own tagset. This result is very similar to the results of the Lancaster CLAWS tagger and other probabilistic taggers around the world. To a certain extent, this has provided evidence of the feasibility of the Markov model for statistics-based tagging programs and also laid a foundation for further statistical parsing, although statistical parsing approaches are different from tagging.

Acknowledgment

We would like to thank Prof. Roland Sussex of the University of Queensland for his insightful comments to the paper.

References

Aarts, Jan and Wilhem Meijs (eds). 1990. *Theory and Practice in Corpus Linguistics*. Amsterdam: Rodopi.

Atwell, Eric. 1987. Constituent-likelihood grammar. In R. Garside, G. Leech, and G. Sampson (eds). *The Computational Analysis of English*, 57–65. London: Longman.

Brill, Eric and Mitch Marcus. 1992. Tagging an unfamiliar text with minimal human supervision. In *Proceedings of the International Workshop on Fundamental Research for the Future Generation of Natural Language Processing*, 112–120. Manchester: Centre for Computational Linguistics, UMIST.

Church, Ken. 1988. A stochastic parts program and noun phrase parser for unrestricted text. In *Proceedings of the 2nd Conference on Applied Natural Language Processing*. Association for Computational Linguistics, 136–143.

Cutting, Doug, Kupiec, James, Pedersen, Jan and Peter Sibun. 1993. *A practical part-of-speech tagger*. Palo Alto: Xerox Research Centre.

De Rose, Steve. 1991. An analysis of probabilistic grammatical tagging methods. In S. Johansson and A-B. Stenström (eds) *English computer corpora*, 9–13. Berlin: Mouton de Gruyter.

Garside, Roger. 1987. The CLAWS word-tagging system. In R. Garside, G. Leech, Geoffrey and G. Sampson (eds). *The Computational Analysis of English*, 30–41. London: Longman.

Garside, Roger, Leech, Geoffrey and Geoffrey Sampson (eds). 1987. *The Computational Analysis of English*. London: Longman.

Johansson, Stig and Anna-Brita Stenström, (eds). 1991. *English computer corpora: Selected papers and research guide*. Berlin and New York: Mouton de Gruyter.

Qiao, Hong Liang. 1997. T-tag oriented parsing. PhD. Thesis. Brisbane: University of Queensland.

Sampson, Geoffrey. 1983. Probabilistic models of analysis. In R. Garside, G. Leech and G. Sampson (eds). *The Computational Analysis of English*, 16–29. London: Longman.

Souter, Clive and Eric Atwell (eds). 1993. *Corpus-based Computational Linguistics*. Amsterdam: Rodopi.

Svartvik, Jan (ed). 1992. *Directions in corpus linguistics*. Berlin: Mouton de Gruyter.