Measuring syntactic difficulty
(draft September 1995)
Richard Hudson

1. Overview

Some sentences are harder to process than others, and in some cases this is clearly because of their syntax rather than because of their meaning. The contribution of syntactic structure is illustrated by the following pairs of sentences which have the same meaning and contain virtually the same words, but have different syntax; in each case the first member of the pair is easier to process than the second.

(1) aIt is remarkable that both of the Siamese twins survived the operation.
bThat both of the Siamese twins survived the operation is remarkable.

(2) aHe gave away all the records that he had collected when he was younger.
bHe gave all the records that he had collected when he was younger away.

(3) aBooks are expensive that students buy who lecturers advise.
b#Books that students who lecturers advise buy are expensive.

The first pair are from Frazier and Rayner (1988), and the others also illustrate well-known processing problems. In 0b the heavy processing load caused by the long subject can be fixed by extraposition; in 0b the long wait for away, the second complement, can be fixed by moving the object across it; and in 0 the centre-embedding can be fixed once again by using extraposition. (I follow Gibson 1991 in prefixing `#' to sentences which are simply impossible to process.) In such examples it is clear, and uncontroversial, that the differences in processing difficulty are entirely due to the syntactic structures concerned.

What is still a matter of dispute is the explanation for these differences. Whatever the explanation turns out to be, it will have to be based on general theories for two different areas of human language: a theory of grammar, and a theory of parsing. Various alternatives have been considered in both areas, but most researchers have assumed some version of Chomskyan phrase-structure. It is this assumption that has driven the distinction between `top-down' and `bottom-up' parsing which has underlain much of the debate about parsing, and which obliges any parser to construct non-terminal nodes. It is very natural that psycholinguists have assumed phrase structure in view of its dominant position in linguistic theory, but this position is due at least as much to social pressures as to academic debate.

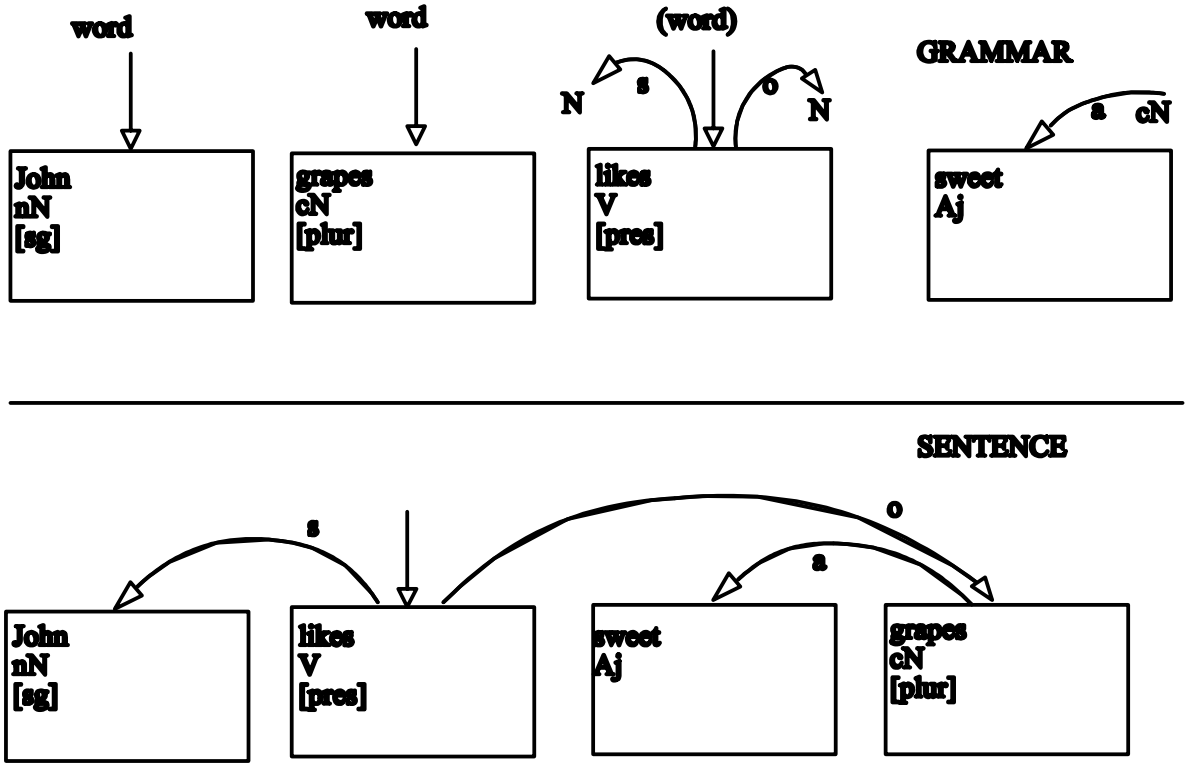
One alternative which is already available to psycholinguists is categorial grammar (e.g. Pickering & Barry 1991, Steedman ...); one of the distinctive characteristics of at least some versions of this theory is that phrase-structure plays a rather small part (and of course phrase-structure rules play no part at all). The purpose of the present paper is to introduce an even more radical alternative in which phrase structure (as such) plays no part at all: dependency theory, and in particular the version of dependency theory called Word Grammar (**WG** - see Hudson 1984, 1990, 1992, 1993a, b, c, 1994, 1995a, 1996, Fraser & Hudson 1992). The

next section will explain the WG view of sentence-structure, and the following section will introduce a theory of parsing which yields such structures. In spirit this will turn out to be similar to some parsing theories based on phrase structure, but the differences are still significant. Section 4 discusses the first of two complementary measures of syntactic difficulty, based on `dependency distance' - the number of words that separate a word from the word on which it depends - and applies it to some well-known processing difficulties. Section 5 introduces the second measure of difficulty, based on `dependency density' - the number of incomplete dependencies outstanding after each word, which is similar in spirit to the various node-counting measures based on phrase structure. However, when we apply these measures to the main problem in this area, the difficulty of centre-embedded structures, they both fail. Section 6 will suggest why these are unprocessable, building on the details of the parser suggested earlier. Section 7 looks at some head-final structures from other languages - Dutch, German and Japanese. All three languages seem to be affected by density in the same way as English. The appendix will suggest a new source of psycholinguistic data - the statistics of naturally-occurring texts.

2. Word Grammar and sentence structure

The characteristic of WG which is most relevant to this discussion is that (with the important exceptions of coordination, compounding and clitics) the grammar refers exclusively to single words. Sentence structure is therefore defined entirely in terms of pairwise relations between individual words. These relations are dependencies, in which one word is the dependent and the other the `**parent**'[1], but since some words have more than one dependent, each with very different characteristics, the dependencies themselves are classified in terms of more or less traditional `function' labels such as `subject', `object' and `adjunct' (abbreviated in obvious ways to `s', `o' and `a'). The task of parsing a sentence therefore reduces to the task of finding each word's parent, labelling the dependency and classifying the words. Fig. 1 illustrates with a simple example.

In all my earlier work I have used the term `head' for the superordinate word, but this has led to a great deal of confusion because, for all its apparent similarity to the heads of phrase structure, it defines a different word. Take a phrase such as in large cities. What is the head of large cities? In phrase structure the answer is cities, but in dependency-based theories such as WG, it is in (though strictly speaking in is the head of just the single word cities). In any case many people find the metaphore of `head' unhelpful in thinking of dependencies, whereas children obviously depend on their parents (in much the same way as a subordinate word depends on a superordinate one). The analogy is much better than the one between mothers and phrase nodes, with daughters as their parts - in real life, a daughter is not part of her mother, though as a child she does depend on her.



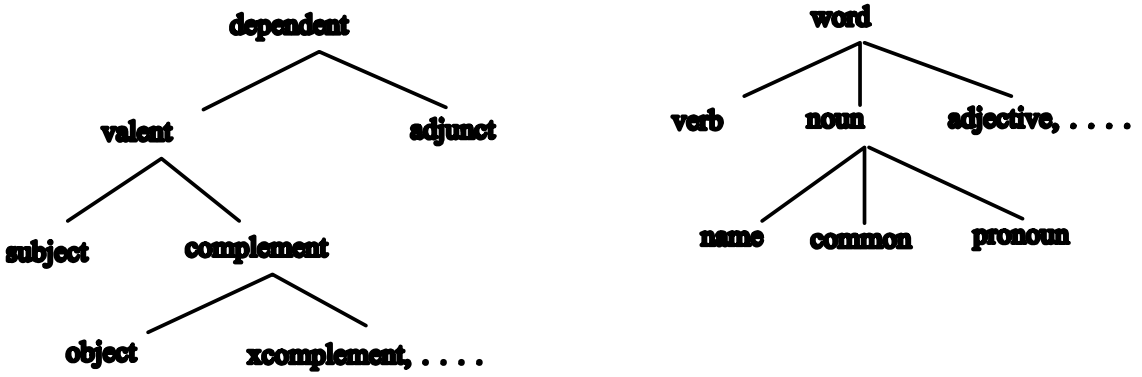
1
Fig. 1

In this example, the underlying `grammar' is shown above the dotted line and the structure for John likes sweet grapes is shown below it. The arrows show dependencies, with the arrow-head pointing towards the dependent (i.e. towards the word that has the named function). As can be seen, the sentence structure is an ordered list of the entries for the individual words, exactly as shown in the grammar except that the dependencies that are dangling loose in the grammar-entries are tied to other words in the sentence. Each word and each dependency must be compatible with the information in every relevant grammar-entry.

In some cases all or most of this information is contained in the entry for the head. For example, the subject of likes must be a noun and must precede it because of the requirements imposed by likes, and similarly for its object, but the entries for John and grapes say nothing either about the position of the their parent, or about its word-class. In their case, virtually all the restrictions on the dependency flow from the verb entry. The same is true for their semantic relationships, though the simplified entries in Fig. 1 do not try to show these. The fact that John is the `liker' and the grapes are the `likee' comes entirely from the verb's entry. In contrast, the adjective sweet is an adjunct of grapes entirely by virtue of information contained in its own entry (which requires its head to be a following common noun). There is no `slot' for adjuncts in the dependency structure for grapes, so there is also no upper limit to the number of adjuncts that are permitted. The position of sweet in relation to grapes is also fixed by sweet, rather than by grapes: in English, modifying adjectives precede the modified noun whereas other modifiers (preposition phrases,

relative clauses etc) follow it.

The example was chosen to illustrate the contrast between `valents' (subjects and complements, all of which are selected and constrained by the parent and which are typically bare nouns) and adjuncts (where information is contained in the dependent's entry). Since the flow of information is crucial to this distinction we might well expect it to be relevant to the way in which parsing is carried out; and we shall indeed see below that it is quite crucial to the measurement of complexity, and also in the explanation of centre-embedding problems. In terms of standard phrase structure analyses complements and subjects do not form a natural class in contrast with adjuncts; rather, the natural grouping would contrast complements, as sisters of the head-word, with subjects and adjuncts, as non-sisters. Since tree geometry plays no part in the completely flat structures generated by a dependency grammar, grammatical functions are distinguished by label; and labels allow a different kind of hierarchical organisation in terms of inheritance hierarchies such as those shown in Fig. 2.



2
Fig. 2

Inheritance plays an important part in WG because it allows full entries for individual words to be derived without being listed. Redundancy can be minimized - though it is of course a matter of psycholinguistic debate how far it actually is minimized by humans - by locating generalisations at the appropriate level in one of these hierarchies. The diagrams in Fig. 1 are not the `official' notation, which consists of verbal propositions (`facts') to which default logic may apply. For example, the entry for John presents the following facts:

(4) aJohn is a name (`nN').
bJohn has a parent.
cThe parent of John is a word.

The second and third of these facts are inherited from much more general facts:

(5) aA name is a noun.
bA noun is a word.
cA word has a parent.
dThe parent of a word is a word.

Similarly for the word-order information in the entries for likes and sweet, which are derived from more general facts:

(6) aA word follows its parent.
bA word's subject precedes it.
cAn adjective precedes its parent if this is a common noun.

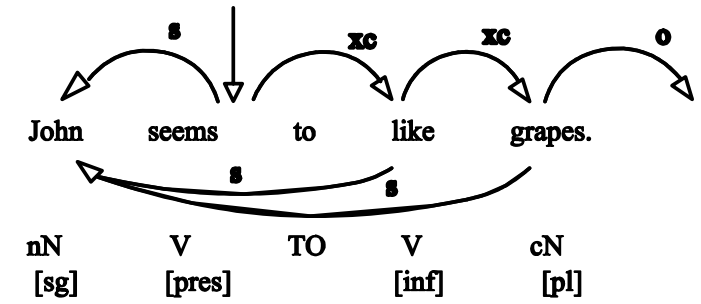
The details of these derivations are not relevant here, but it is important to bear in mind that the full word-entries which a parser can retrieve are derived rather than simply found ready-made in the lexicon. For present purposes the diagram notation will be most helpful, though we can now dispense with the little boxes.

Another relevant characteristic of WG structures is that they are completely monostratal. There is a single syntactic structure which is consistent (no contradictions), complete (includes every word) and minimal (no empty nodes). However, more complicated examples naturally need more complicated structures, and all the extra complexity is packed into the dependency structures. Every sentence has a simple `surface structure' of dependencies which are equivalent to a phrase structure: every word has a single head, and no dependencies tangle with each other. For convenience this can be drawn above the words (`on the surface', if the words correspond to `ground level'), and, as we shall see, it is this that controls word order and guides the parser. But (unlike other versions of dependency grammar) WG allows extra dependencies to be added to the surface structure in constructions like raising, relativization, extraction and extraposition. These `extra' dependencies are typically the most relevant to recovering semantic structure, but they are built more or less mechanically onto the surface structure on the basis of information supplied in the word entries.

For example, take the entry for a raising verb like SEEM. This says that SEEM takes an xcomplement (a term borrowed from LFG, for a subject-sharing complement), and the entry for xcomplements tells us that they add an `extra' dependency:

(7) aSEEM has an xcomplement.
bA verb's xcomplement's subject is also the verb's subject.

These entries conspire to produce an entry for seems which gives it an xcomplement link in the surface structure, but a subject link from its own subject to its xcomplement among the extra dependencies. If the xcomplement is TO, the same applies as TO is also a subject-raiser - hence structures like the one in Fig. 3.

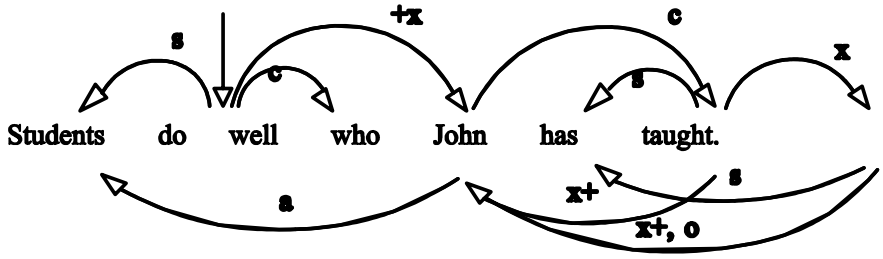


3

Fig. 3

For good measure, Fig. 4 gives a taste of more complex structures that can be generated for a sentence which involves extraposition, extraction and raising. (The `extraposee' relation is labelled `+x' and the extractee `x+', to show their respective relations to their parent: `parent + x' or `x + parent'.)

(8) Students do well who John has taught.



4

Fig. 4

Interestingly, all the extra dependencies which make some sentence structures look so complicated seem to add little if anything to the difficulty of parsing. We shall see below why this is so.

3. Word Grammar and parsing

Suppose a parser could reconstruct the surface structure for a sentence such as 0. How difficult would it be to work out the extra dependencies? For raising there would be no difficulty at all, because the extra dependency follows automatically from the entry for the raising verb, as we saw for seems in the previous example. The extra dependencies associated with extraction are almost as easy, because the extractee relationship is passed recursively down the surface dependency chain until it can be `cashed' as a meaning-bearing dependency - in this case, as object of taught. At each word in this chain there is a simple choice between passing the extractee down the chain and using it as a `proper' dependent (e.g. object).

The extraposee dependency could in principle be much harder to handle, because this supports an extra adjunct dependency whose parent is some (otherwise undefined) dependent of the extraposee's parent; so the parser has to cope with an

input such as .. do .. who , where the antecedent of who is some dependent of do. This could involve an extensive search, but in practice extraposition (i.e. extraposition of a noun's postmodifier) is generally limited to cases where the parent is the nearest noun dependent of the verb. A reasonable search strategy for finding the parent of a word which is also the extraposee of W would therefore be to search for the most recent dependent of W which is a noun.

Moreover, in all constructions the extra dependencies are very local: given the dependency relation between word X and word Y, we add an extra dependency between X and some other word which is also more or less directly dependent on Y. As analysed in WG, long-distance effects are due to recursive application of the relevant rule. This combination of predictability and locality means that the extra dependencies in a WG sentence come more or less for free, and are to that extent irrelevant to parsing difficulties.

The aim of a WG parser is therefore to reconstruct a surface structure, which means finding one head for each word (except the sentence's main verb). However, another requirement is that the parser should work incrementally, taking the words in the order of utterance and pushing the analysis of each word as far as possible before accepting the next word. We shall consider a WG parser which has these characteristics below, but it may be helpful to locate the parser in relation to the more familiar phrase-structure based parsers.

The typology of `top-down' versus `bottom-up' means very little in dependency parsing, as Fraser shows in a thorough survey of known dependency-based parsing systems (1993). A dependency parser is `top-down' in the sense that each word may set up expectations which later words have to satisfy, but it is `bottom-up' in the sense of being driven entirely by the incoming words. It is interesting to find that even if human parsing is based on phrase structure it cannot work in either a pure `top-down' or a pure `bottom-up' direction (Gibson 1991, Abney & Johnson 1991). The favoured strategy is a `left-corner' direction, which means a bottom-up approach to the left-most constituent of a phrase, followed by a top-down approach to the rest of the phrase. This is rather similar to a dependency-based parsing strategy in its move from bottom-up observation to top-down prediction, but it still faces some of the weaknesses of phrase-structure based parsing which I list below:

a. The information about non-terminal nodes takes up much of the parser's memory, and, since most such nodes are simply copies of the head-word's features, there is vast redundancy.

b. If phrases are generated as wholes, either by phrase-structure rules or by more general principles plus projection from the lexicon, then optional complements are a problem because it is unclear when the phrase can be closed. If it is closed too soon, backtracking will be needed in order to reopen it; but if it is left open indefinitely, just in case there is more to come, the burden on working memory is intolerable. For example, having heard the partial sentence I'll help you, we have the choice between closing the VP (and the whole sentence) and leaving it open. If it continues with to wash the car, the first choice was wrong, but if the next words are which will be a change the second was wrong.

c. Adjuncts are a perennial source of difficulty and uncertainty. If they are Chomsky-adjoined to the tree, then a following adjunct requires a certain amount of back-tracking in order to graft an extra node into the middle of the existing tree; but attaching them directly to the existing nodes conflicts with what most phrase-structure grammarians believe. The uncertainty about adjuncts has allowed psycholinguists to pick and choose to suit their theories; a particularly blatant case which has been criticized is Frazier's (1979) assumption that PPs Chomsky-adjoin to NPs and attach directly to VPs (Abney 1989).

d. There is a vast, perhaps infinitely vast, range of alternative phrase-structure analyses of a given sentence, assuming different grammars, so the psycholinguist is very much at the mercy of theoretical fashions among linguists. This is especially true when empty categories and transformations are allowed, but even surface phrase-structure grammars which allow unary branching offer many possible structures. Such differences threaten to undermine any attempt to explain why some sentences are harder to process than others if the explanation is to involve counting structural nodes. In contrast, a completely surface dependency-based approach offers very few possible analyses compatible with the obvious facts. For example, it is hard to imagine any alternative to the analysis of John likes sweet grapes in Fig. 1 (though many different notations are of course possible, as are differences in the classification of the words and dependencies).

Some of these problems can be avoided without abandoning phrase-structure entirely, though the solutions tend to involve moves towards dependency-based parsing, which leaves phrase structure as an irrelevant inconvenience. An interesting example is Abney's (1989) proposal of a `licensing-structure' parser, described as follows:

> In licensing-structure parsing, .. node expansions are collections of smaller pieces of structure, viz. binary relations between head and sister - a sister appearing in the expansion of XP iff it is assigned a licensing relation by the head of XP.

This is a very good description of a head-based dependency parser, which generates pairwise dependencies between words, except that the dependency parser does without the XP node.

However there is a conflict between the demands of a head-based parser and those of incremental parsing. Suppose we are parsing the input large brown cows .... If we have to wait for a phrase-head before assigning any structure, we can do nothing with the first two words apart from putting them onto the `pending' stack. The situation gets worse as the number of premodifiers increases; and at worst we could have a stack containing numerous unrelated words, each occupying a separate cell, even when the input is quite easy (if boring) to process: long boring uninspired badly-prepared barely audible dogmatic conference presentations, and their ilk. For incremental parsing it is essential to be able to handle sequences like this word by word, which means being able to integrate them even before the phrase-head (i.e. the

words' parent) reaches the parser. In principle two adjectives could each have a separate parent; for example, in that well-known transformational grammar textbook, the adjective well-known depends on textbook while transformational depends on grammar. However, in practice it is more likely that they share their parent, so it seems safe to assume that when we hear or read large brown, we guess that their parents are the same. (Such a guess would presumably be in the spirit of left-corner parsing: you hear an adjective, so you assume it starts an NP which could contain any number of adjectives; therefore when the next word turns out to be an adjective, you guess that it fits into the structure of this same NP.) The WG parser exploits this intuition by allowing expected parents to be merged even while they are still only expected. As we shall see, this makes a considerable difference to the measure of complexity if we assume that expected heads (or dependents) are a burden on working memory; but we shall also blame it for the difficulty of centre-embedded examples.

The WG parser reflects the following assumptions about the **surface** dependency structure (though the total structure may be much richer):

a. Dependencies can `nest' but must not tangle, so a word's dependents are never separated from it by its parent. The parser reflects this restriction by searching among the preceding words for dependents of the current word before it seeks a parent.

b. Every word except the main verb has a parent, which may have other dependents. The parent plus all its dependents (plus their dependents, recursively) constitute a `phrase' which is held together internally by the various dependencies, but which is represented externally (in relation to the rest of the sentence) by the shared parent, which we can call its `**ancestor**'. The parser recognises `phrases' defined in this way among the words processed so far, assigning each such phrase to a separate stack-cell, but the question of when to `close' a phrase does not arise because phrases are not generated as such. For example, John likes is stored in a single cell, but it is still open to receive grapes (plus the latter's dependent sweet) when the latter arrives. At any point a new word may be added (as a dependent) to an existing phrase.

c. Since each word has only one parent, there is only one word in each stacked phrase that is available as a potential dependent for following words: the phrase's ancestor, the one word in the phrase that still has no parent. The parser labels the ancestor of a phrase so that it is easily recognised, and when looking for potential dependents among the preceding words, it considers only phrase-ancestors, ignoring all other words.

d. Because dependencies cannot tangle, some words in a stacked phrase may be made `invisible' as potential parents for the current word by some intervening dependency. For example, when John likes is on the stack, and grapes seeks a parent, John is not available because a dependency from John to grapes would have to cross a dependency from likes to any parent it may have. (The vertical arrow stands for this potential dependency, and counts as an infinitely long dependency that must not tangle with any other in the surface structure.) The only words in a stacked

phrase that are available as potential parents for later words are those on its `right edge' - i.e. on the chain of dependencies from the phrase's last word to its ancestor. The parser therefore restricts its search for parents to this chain of words, starting with the most recent word and working back `up' the chain.

e. The valency information about a word includes all the other words that it needs: its dependents and also its parent. After the current word has been processed some of these needs may not have been satisfied. If the grammar says that they should have been satisfied by now (because the dependent or parent concerned has to precede the word), the analysis fails, but if it allows them to be satisfied later the parser stores them as outstanding `needs' which it will try to use later words to satisy. The needs are cancelled when satisfied, and the list of needs should be empty by the end of the sentence. The needs can be stored in the same format as they will have when satisfied. Assuming that words are numbered W1, W2, ... a complete analysis of our example sentence John likes sweet grapes contains the following propositions:

(9) a The parent of W1 is W2.
b The parent of W2 is Ø. (i.e. W2 has no parent.)
c The parent of W3 is W4.
d The parent of W4 is W2.

While waiting for a complete analysis, the word-numbers can be replaced by a hypothetical index (using letters instead of numbers) such as Wa. Thus at the point where John has arrived but likes has not, we have the proposition:

(10) The parent of W1 is Wa.

f. As I explained earlier, we probably guess that two attributive adjectives like large brown share the same parent; but more generally, the same is probably true of nearly all words that still need a parent. For example, given the input Actually in Japan they, it is absolutely certain that the words actually, in and they share the same parent. This may be a fact of English grammar rather than a general characteristic of all languages, but the way English works guarantees that no back-tracking will be needed if the parser anticipates a shared parent for these three words. The parser therefore makes this guess by assigning the same index to the hypothetical words which will eventually serve as parent for all three. In the analysis of a sentence which starts Actually in Japan they, the parser therefore generates the following propositions:

(11) a The parent of W1 is Wa.
b The parent of W2 is Wa.
c The parent of W3 is W2.
d The parent of W4 is Wa.

The hypothetical shared parent is enough to bind these words into a single phrase, which can occupy a single cell in the stack.

This may reduce the memory load, compared with carrying forward three unrelated `phrases', but it certainly reduces the work that needs to be done when the verb eventually arrives (as W5, let us suppose). In this case the parser starts by checking the needs-list, and finds Wa, whose specifications precisely fit the current word; so all that needs to be done is to identify Wa with W5. We can summarise the output of these four words as follows:

(actually (in Japan) they: + Wa)

In this summary, the phrases are bounded by brackets, the colon follows the words already processed, and the needs list is shown after `+'.

As we saw, a similar guess for large brown is more risky because these words could in fact modify different nouns as in large brown-paper bags, but the risk is worth taking. It is likely to reduce work later, and if the guess is wrong the back-tracking which is needed involves no more than transferring a dependency from one word to another. Furthermore, the risk-taking may go further if the grammar permits. We know for sure that when an adjective precedes its parent, the latter must be a noun, so meeting the adjective is tantamount to meeting the noun itself. Now suppose we have the sequence Tomorrow large brown at the start of a sentence. We can be absolutely sure that the noun which is guaranteed by the two adjectives will also share the same parent as tomorrow, so even at this stage we can predict a dependency between two words which are still hypothetical: the parent of tomorrow and the parent of large and brown:

(11) a The parent of W1 is Wa.
b The parent of W2 is Wb.
c The parent of W3 is Wb.
d The parent of Wb is Wa.

The stack holds the following:

(tomorrow (large brown: + Wb) Wa)

Such intelligent guessing by the parser can be built into the parsing procedure itself, but ultimately it rests on the grammar of English.

A reasonable question at this point would be: given that this parser is aiming to group words together into phrases, in what sense is it **not** constructing a phrase structure? The answer is that it would be very misleading to describe the output of this parser as a phrase structure, because it has none of the characteristics usually associated with this term: the only relations that are explicit in the analysis are pairwise word-word dependencies. Phrases are not shown as such, less still are they labelled with class-names such as `NP'. The only role that `phrases' play is in the parsing, where a phrase is a group of words which offers a single ancestor word as a potential dependent for later words, but these phrases do not have to satisfy any grammatical constraints on phrase-structure other than those which apply to word-word dependencies. Since phrases are implicit in any dependency structure, the

parser cannot help generating implicit phrases, but this is very different from generating a phrase structure.

Given these general characteristics of the parser, then, the parser operates as follows, where I assume two storage spaces: the `stack', and the `current buffer'. For all I know these are actually a single data-structure, but it is helpful to describe the operations as though structures were stored in the stack, with the buffer as a kind of workbench on which bits taken from the stack may be glued onto the current word. The overall structure of the operations is simple: the parser looks for the current word's dependents first, and its parents only after the supply of dependents is exhausted. For example, in processing grapes after John likes sweet green, it builds the buffer as follows:

grapes
green grapes
sweet green grapes
John likes sweet green grapes

In more detail, the following operations take place:

1. Read the current word, C into the `current buffer' and inherit its permanent properties from the grammar, including its valency `needs' (valents and parent). Give each needed word an abstract index (e.g. Wx). Order these valents around C according to the available information about word order. Where order is free (as for the anchor of a noun), order the needed word both before and after and coindex the two occurrences:.

Stack = (W1 ...: Wf .. Wa)
Buffer = Wp_i (Wx .. C: .. Wy) Wp_i

2. Consider the stack for potential dependents of C. Check the first word (Wf) in the current `needs' list of anticipated words.

a. Suppose Wf follows a phrase bracket (i.e. Wf is the anticipated parent of the last phrase processed), with Wn as the ancestor of that phrase:

Stack = (W1 .. (.. Wn ..:) Wf .. Wa)

a1.Check whether Wf's characteristics (word-class, etc.) match C's, and if so:
a2. identify Wf as C and remove Wf from the stack's needs list,
a3.move the whole phrase dependent on Wf from the stack into the buffer, and record that C is the parent of Wn, the phrase's ancestor.
a4a.Suppose further that Wn matches a valent Wy of C:
Buffer = Wp_i (Wx ..Wy C: .. Wz) Wp_i
then record that Wn is that valent and remove Wy from C's needs list:
Buffer = Wp_i (Wx .. (.. Wn ..) C: ..Wz) Wp_i
a4b.otherwise record that Wn is an adjunct of C and move the phrase into the buffer after any valents:
Buffer = Wp_i (Wx ..Wy (.. Wn ..) C: ..Wz) Wp_i

b. Repeat a until it fails; then go to 3.

3. Consider the stack for a parent of C.

c. Suppose Wf does not follow a phrase bracket. In this case Wf is an anticipated dependent, i.e. a valent of some word Wp.

Stack = (W1 .. Wp .. : Wf .. Wa)

Test whether C can be this dependent:

c1. Does C match Wf?
c2. Is C's list of needed preceding dependents empty?
c3. Can C's parent precede it?
c4. Does Wp match C's potential parent?

Buffer = Wp_i (Wx .. C: .. Wy) [x a number]

If all answers are positive, C can be the dependent in question, so record that it is, remove Wf from the list of needs in the stack, and move the entire buffer back onto the stack, thereby terminating the cycle for word C. Otherwise try d.

Stack = (W1 .. Wp .. (Wx .. C: .. Wy) .. Wa))

d. Try C as an adjunct of some word on the stack, starting with the immediately preceding word and working up the exposed `right edge' of the last phrase. Assume the following stack and buffer:

Stack = (W1 .. Wn: .. Wa)
Buffer = Wp (Wx .. C: .. Wy) [x a number]

d1. Compare Wp, the permitted parent of C, with Wn. If they match, and C is a semantically possible adjunct of Wn, record that C is an adjunct of Wn and move the entire buffer into the stack, terminating the cycle for C. Otherwise go to d2.

Stack = (W1 .. Wn (Wx .. C: .. Wy))

d2. Repeat d1 with Wn replaced by Wn's surface parent, and so on up the dependency chain. If there is another phrase on the stack, try this as well. If this still fails, go to 4.

4. If the parser has not been able to find a parent for C among the words so far parsed, prepare for a following parent which C may share with preceding words:

e1. Remove the first Wp if it is coindexed with a later one, leaving just the one at the end of the buffer.
e2. Suppose the first anticipated word in the stack, Wf, follows a bracket, showing

that Wf is the anticipated parent of some earlier word Wn.

Stack = (W1 .. (.. Wn ..:) Wf) .. Wa)

If Wf matches Wp (the anticipated parent of C), and Wp could also be the parent of Wn, merge Wf and Wp. (I.e. guess that C is part of the same phrase as Wn.) Move the entire buffer into the stack, terminating the cycle for C:

Stack = (W1 .. (.. Wn .. (Wx .. C:) Wf) .. Wy)

e3.If e2 fails, try to look a further step ahead by considering the needed head of Wp - i.e. the grandparent of C, Wg. If Wf matches Wg, merge them and proceed as above:

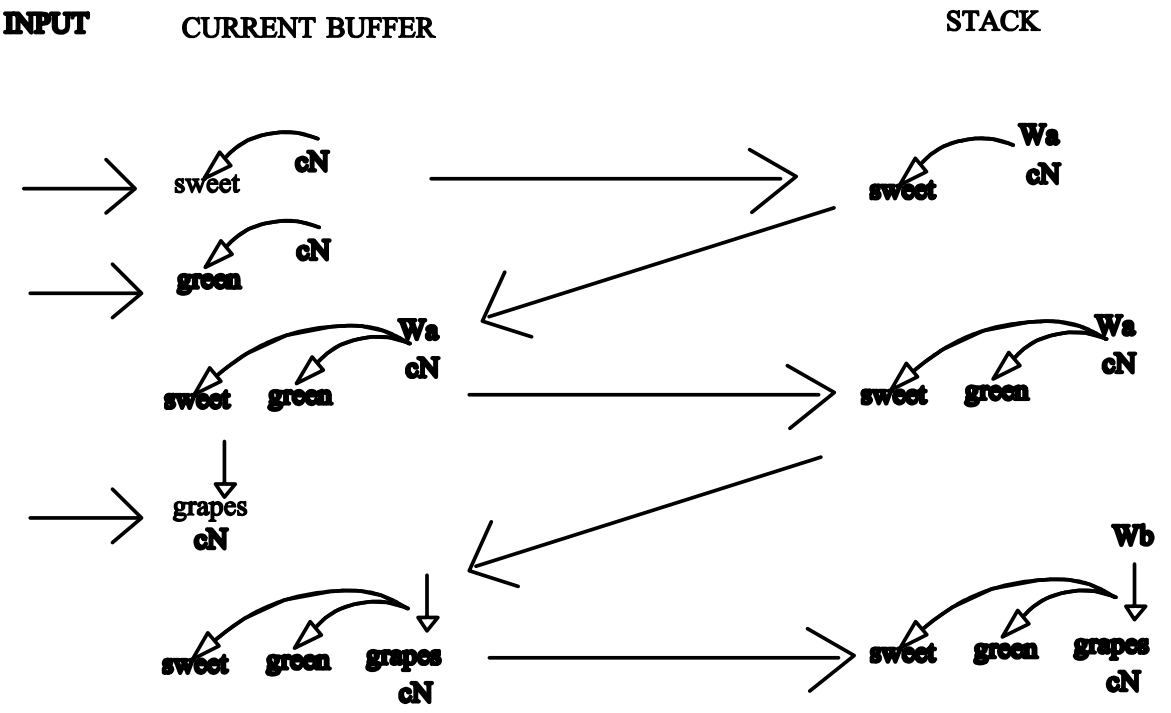
Stack = (W1 .. (.. Wn .. (.. (Wx .. C:) Wp) Wf) .. Wy)

e3If Wp cannot be merged with an already anticipated parent, move the entire buffer into a new cell on top of the stack.

Stack = ((Wx .. C: .. Wy) Wp)
(W1 .. Wn: .. Wz)

5. Repeat with the next word as C.

The movement between stack and current buffer is illustrated in the diagram in Fig. 5, which shows how the input sweet green grapes would be treated.



5
Fig. 5

This parser[2] does not take account of coordination, nor (more seriously) does it provide any mechanism for handling ambiguous input. At present it will produce at most one analysis per input string, and a genuine working model would have to consider serious issues such as the choice between parallel and serial treatments of alternative analyses. However we shall see that some of the measures of difficulty which we shall discuss below are relevant to this choice.

4. Dependency distance

We now turn to the sources of difficulty in using this parser, and more specifically difficulties which arise because of the specifically syntactic structure of the input. We ignore difficulties due to ambiguities, to purely semantic complexity (e.g. an excess of negatives), or to pragmatic problems; all these things are clearly important, but independent of syntactic difficulty. Dependency structures suggest three variables which might be relevant to how hard a sentence is to process:

a. The number of extra dependencies due to `complicated' constructions such as raising, extraction and extraposition. As I have already explained, these generally follow more or less automatically from the word entries plus the surface

[2]The parser as outlined here is intended as a model of the human parser, but there are also a number of computer implementations of more or less similar models, all of more or less `Mickey-Mouse' proportions. The survey in Fraser 1993 includes all WG parsers as well as other dependency-based systems; my own contribution is reported in Hudson 1989.

dependencies, so they need not cause difficulty; and my impression is that in practice they are irrelevant to processing difficulty. Indeed, one of these constructions (extraposition) is used precisely in order to make processing easier, as we shall see shortly.

b. The distance between words and their parents, measured in terms of intervening words. This is the topic of the present section.

c. The `density' of surface dependencies at each point in the sentence, which we consider in the next section.

One reason why `dependency distance' matters is that working memory decays. A realistic model of a human parser (unlike the one outlined above) would not assume that the whole of a sentence is preserved, being shunted between the stack and the current buffer, right to the end. We know for sure that this is not so - that the shelf-life of a word in memory must be quite short. Incremental parsing extracts meaning (and the intended message) as quickly as possible, after which the words themselves are of no value and can be abandoned. However `forgetting' is clearly not a clean deletion operation, but a gradual deterioration; so words get harder and harder to retrieve, without necessarily being lost altogether. It is the words on the stack that are deteriorating in this way, so being on the stack does not guarantee equal accessibility.

Bearing this inevitable decay in mind, let us consider the two possible relations between a word and its parent: either the word comes first, or the parent does. Let us call the dependent word D and its parent P. Suppose D arrives before P. By the time P comes, the parser will have stored D as the ancestor of the phrase on the top of the stack, and in this phrase it is only D that is available as a dependent of P. For example, in the input Sweet grapes without seeds are, the word grapes is the only candidate for the subject of are because all the other words already have parents. This means that the parser can in principle go straight to this word, or at least that it can instantly eliminate all the alternatives - but only if the memory of grapes is still strong and clear. The longer the subject phrase, the more effort it takes. Consider, for example, the unextraposed version of our very first example:

(14) That both of the Siamese twins survived the operation is remarkable.

I assume that the subject phrase is the clause that both of the Siamese twins survived the operation, and that its ancestor is that. That is the subject of is, but by the time is arrives, that has been held on the stack for eight words-worth of time - several seconds, which is a long time in short-term memory.

The parser has three ways of coping with such a sentence:

a. It can leave that to deteriorate in the hope of being able to muddle through on the basis of semantics and pragmatics when the main verb arrives; this is risky, and semantic and pragmatic guesswork may demand as much effort as it would take to preserve that in memory.

b. It can leave that to deteriorate but revive it once it is actually needed. Presumably it takes more effort to find a weak memory than a strong one, so this strategy takes more effort than would be needed if the subject had been processed recently.

c. It can keep on boosting the memory of that precisely because it is still not syntactically complete. When needed, that will therefore be as strong as if it had been freshly processed, but once again the boosting process must cost some effort.

Whichever of these strategies the parser adopts, the same conclusion emerges: the further D is from P, the harder the dependency between them is to process.

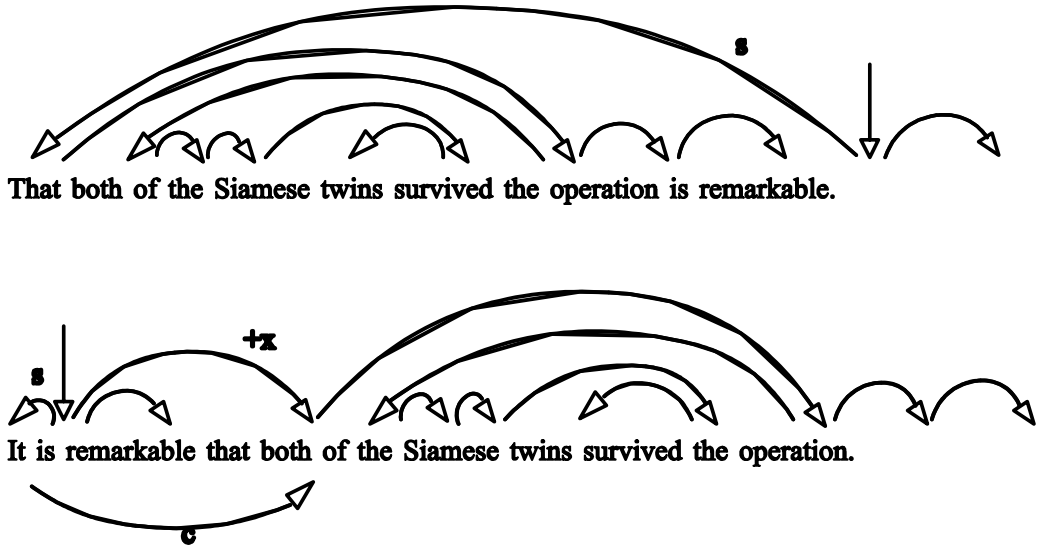
Professional writers, including journalists, often seem surprisingly unaware of the difficulties posed by long subjects with early ancestors; the following is a typical hard-to-read sentence (Guardian 6th September 1995; the comma after batons may be a sop to the reader's difficulties):

(15)A British Aerospace sales executive disciplined by the company for taking part in a demonstration of illegal electric batons, is hosting overseas delegations at the UK defence equipment exhibition at Aldershot this week.

However, such problems seem to have had an impact on the development of grammatical systems. On the one hand, various `escape routes' are available for those who would otherwise be in danger of overloading their hearers' and readers' memories. One such is extraposition, mentioned earlier as a source of complexity (because of the extra dependency) which justifies itself by reducing difficulty. Consider for example our first pair of sentences:

(12) a. That both of the Siamese twins survived the operation is remarkable.
bIt is remarkable that both of the Siamese twins survived the operation.

I assume that the subject phrase in the first sentence is that both of the Siamese twins survived the operation, whose ancestor is that. The structure is shown in Fig. 6, contrasted with the second sentence. If we measure distance in terms of the number of intervening words, the distance from this word to its parent (the main verb is) is 8. In the second sentence, extraposition has replaced this long phrase by it and relocated it at the end of the sentence. The distance from it to is is zero, but that still needs to be linked into the dependency structure. The relevant link in the surface structure, once again labelled `+x' for `extraposee', is between that and is, which spans a distance of one. The result is clearly easier to process, and the explanation presumably lies in the dramatic overall reduction of dependency distance.



6
Fig. 6

Turning now to the reverse situation in which D follows P, the same considerations apply but a second reason also favours short dependency-links. Suppose we are parsing the following input: John likes grapes without. The current word is without, and the problem is to find its parent. The stack contains John likes grapes, but the strategy for finding a parent is to try the last word in the stacked phrase, then its parent and so on recursively. In this case the search is easy, because the last word is grapes, which is an excellent parent for without. This strategy guarantees that the first candidate considered will be the most recent one, which is equivalent to Frazier's (1979) principle of Late Closure: attach the current word as low as possible in the most recently processed phrase. The further back the parent is, the more alternative candidates have to be eliminated.

At the same time, though, the memory degradation discussed above also applies, so the further back the parent is, the more degraded it is. In this case the parser cannot anticipate later problems by boosting the parent, because potential parents for adjuncts cannot be recognised as such until the adjunct word is being processed. For example, in I know you like smoking cigars, which. The words know, like, smoking and cigars are all available as potential parents for later words, so when which arrives it could in principle take any of them; and indeed all possibilities are easy to construct. The parent of which is shown in brackets.

(13) a I know you like smoking cigars, which are bad for you. [cigars]
b I know you like smoking cigars, which is bad for you. [smoking]
c I know you like smoking cigars, which surprises me. [like]
d I know you like smoking cigars, which may surprise you. [know]

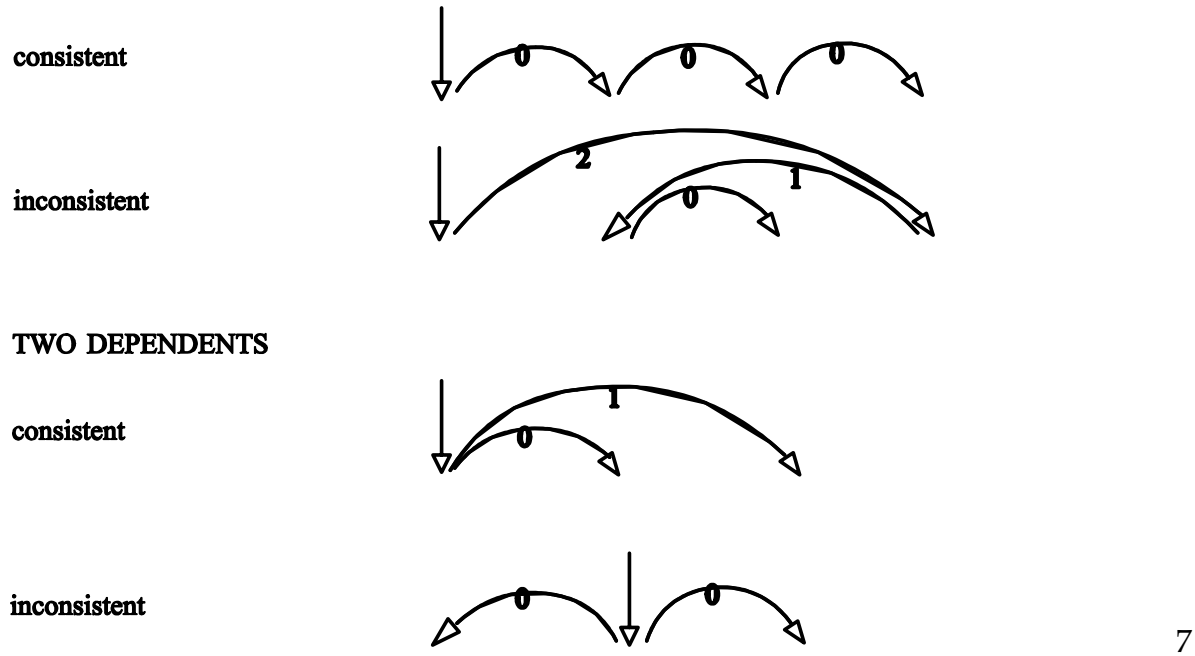
In processing the last sentence, the reader presumably finds the actual parent for which only after considering and eliminating three other candidates, which takes time and effort, in addition to the effort of retrieving aging words from working memory. Once again, then, minimum distances, alias Late Closure, helps the reader

or hearer.

Another way in which languages help their users is by adjusting their word-order rules to allow dependency distances to be short (and we shall see in the appendix that at least in English distances do tend to be very short indeed; in fact most words are immediately next to their parents). Typologists have long recognised a strong tendency for languages to order words consistently in relation to their heads even though head-dependent relations occur in all sorts of different constructions - relations between verbs and their objects, between prepositions and their `objects', between verbs and their subjects, between auxiliary verbs and their xcomplements, between subordinating conjunctions and the subordinate clause-ancestor, and so on and on (Tesnière 1959/1966, Venneman 1974, 1984). Some languages tend to put all words before their heads, while others tend to put them all after their heads. Japanese is a clear example of the former type, and Welsh of the latter type. Unfortunately English is something of a mixture, though the dominant pattern is head-initial (as we shall see in the appendix).

Enough languages belong more or less clearly to one type or the other to make it clear that the tendency to consistency is a reality, so there must be some kind of pressure at work which leads to such consistency; but there is at least one construction which appears to be insulated against these pressures, namely the combination of a noun and a modifying adjective. According to Dryer (1988, 1992), there is no statistical correlation between a language's preferred overall head position and whether adjectives precede or follow the noun that they modify. (Notice that even English illustrates this exceptional behaviour: in spite of the overall head-initial tendency, attributive adjectives are head final, as in big book, not *book big.) This finding undermines any attempt to explain the tendency purely in terms of simplicity and consistency within the grammar. After all, if words follow their heads in every single construction, then a single rule will cover the whole language: A word follows its head. Why, then, should adjectives be immune?

One possible answer is that the tendency to consistency is due to the need to minimize dependency distance when parsing by keeping phrase-heads as close as possible - but that exceptions can also be explained by the same need. The crucial variable is how many dependents a single word has. Suppose, first, that every word has just one dependent. In a consistent language every word must be immediately next to its parent, giving an average distance of zero, but in a mixed language some words will inevitably be separated from their parent by their own dependent (and its dependents). Fig. 7 illustrates these two abstract cases (with numbers showing dependency distance rather than dependency type). Consistency is a clear advantage in such cases, and if words never had more than one dependent we might expect all languages to have evolved to a state of total consistency. However, the same pressure to minimize dependency distance produces the opposite effect when words have two dependents, because in that case inconsistency is an advantage. With its dependents on opposite sides, a word can be immediately next to both, whereas in a consistent language one dependent must inevitably separate the other from their shared parent. Again Fig. 7 illustrates the point. Assuming that every language has words which may have more than one dependent, every language is subject to competing pressures towards consistency and away from it.



7

Fig. 7

However, words are not equally likely to have two dependents. Some words cannot possibly have more than one dependent (e.g. the English preposition OF), while others are very likely to have two. One type of word which typically does have several dependents is the class of verbs, which often have both an object and a subject. Interesting, head-initial languages tend strongly to be SVO rather than VSO: only about 10% of the world's languages are VSO, compared with about 40% that are SVO, the remainder being almost all SOV and head-final (Tomlin 1986). The other obvious exception involves nouns, which often have more than one modifier. (In the last paragraph there were two such examples: average distance of zero and competing pressures towards consistency and away from it.) Here too it would be advantageous to locate dependents on opposite sides of the shared parent, and this pressure may well be sufficient to explain why the position of attributive adjectives is statistically independent of the overall head-position classification of a language, as noted above.

It is interesting to contrast this account with another attempt to explain word-order rules in terms of processing strategies. Hawkins (1990, 1993) argues that the most important goal for a parser is to be able to construct phrase nodes as early as possible, so word-order rules conspire to locate the clues for recognising a phrase as early in the phrase as possible. In the model proposed here, recognising phrases is not one of the parser's ultimate goals, but Hawkins is right (in my opinion) to suggest that it is helpful to recognise phrases as early as possible. As I explained in the discussion of sweet green grapes, the sooner we recognise that the two adjectives share the same parent, the better. If they share the same parent, they belong to the same phrase, so I agree with Hawkins that an adjective can be taken as a clue to a `noun phrase'. Where I disagree with Hawkins is over the implications of this

observation for our thinking about heads. He uses it as evidence that heads are irrelevant in parsing, and more generally in grammatical theory. My conclusion is the opposite: the only reason for recognising that sweet belongs to a noun phrase is that this phrase has a noun as its head, so the parent of sweet is going to be a noun. By anticipating this noun before it arrives we can then guess that the same word will serve as the parent for green, which reduces two elements of uncertainty to one. In other words, the reason for using the adjectives as clues for the noun phrase is precisely that they are **not** its head, but that some other word **is**; far from undermining the status of heads (and parents), such examples actually reinforce it.

In conclusion, I have argued that the distance between a word and its parent is relevant to the difficulty of processing a sentence. If this is correct, then it is evidence that syntactic structure consists of word-word dependencies, and not just of mother-daughter relations as in pure phrase structure. If all word-word relations are mediated by phrase nodes, it is hard to see why parsing should be affected by the distance between the heads of the phrases concerned - why for example a long subject should be so much harder to process than a short one.

5. Dependency density

The second measure of difficulty is the number of `open' dependencies at any given point of time. A dependency is open from the moment when it is first recognised as a `need' to the point where the need is satisfied. For example, in John likes sweet grapes one dependency is opened at John, namely the dependency from John to its parent; at likes this particular dependency is closed because John's parent is likes, but another is opened, between likes and its object. This dependency remains open at sweet, but this word adds another dependency, its need for a parent. Both of these dependencies are closed at grapes, and as with any grammatical sentence this one ends without any open dependencies.

It seems reasonable to assume that dependencies place a greater load on working memory when they are still open than once they are closed, because a closed dependency has achieved its main function of guiding the hearer to a semantic structure. Once the semantic structure is in place, and especially when it has been pragmatically exploited, the syntactic dependency can be allowed to disappear from working memory, but till then it must be maintained. Moreover, the WG parser gives each anticipated word almost the same status as a word that has already been processed, so it presumably occupies almost as much memory; so even after hearing the one word John, the hearer's memory already contains entries not only for John but also for this word's anticipated parent. While the parser is working on the pragmatic reference of John it can already start collecting clues about the parent. Suppose the next word was recently: John recently. The parser merges anticipated parents whenever possible (as with sweet green), so it identifies the parents of John and recently as the same word, still named only by an abstract index which we can label `Wa'. The hearer knows quite a lot about Wa even before this word arrives:

(14) a Wa is the parent of the noun John.
b Wa is also the parent of the adverb recently.

cWa must be a tensed verb, because this is the only word class that can be parent to both a noun and an adverb. (I.e. recently has eliminated the possibility of John depending on possessive 's).

d Because of the meaning of recently, Wa must be past tense.

eWa must be the subject of Wa; the other possibility, that it is extractee (e.g. John I saw in the library) is eliminated because extracted nouns cannot be separated from the subject by an adverb (*John recently I saw in the library).

All this information is based on the two dependencies that are still open after the word recently has been processed. The information about Wa that has already been stored takes up memory space, and the processes which have produced the information must also contribute some load. Of course every bit of work which is done in anticipation reduces the load when Wa finally arrives, but the fact remains that each of these open dependencies adds a processing load during the time that it is open.

The simplest way to measure the load due to open dependencies is to weight each open dependency equally, and to sum the open dependencies after each word: one after John, two after recently, one after saw and zero after Mary, for example (in John recently saw Mary). We can call the score after a word W_i the `**dependency density**' at W_i. This is the dependency equivalent of the familiar node-counting systems in phrase structure, whereby difficulty is measured in terms of the number of nodes dominating each word. (Gibson 1991 contains a useful survey of such systems.) However one important difference between this measure and measures based on phrase structure is that our measure automatically treats adjuncts that follow their parents differently from all other dependents. For example, in John saw Mary recently, the valency of saw allows us to anticipate Mary, but not recently. Perhaps the most important difference between adjuncts and complements is that whereas complements are listed in the valency entry for the parent, adjuncts are not. Subjects are like complements and indeed in LFG they are included under the general category `complement'; this is why we use the cover term `valent', in contrast with adjuncts. Following valents are anticipated, but adjuncts are not, so only the former contribute to dependency density. This contrast applies of course only to dependents that follow their parent; those that precede it all set up an expectation for the parent, regardless of whether they are valents or adjuncts (e.g. John and recently contribute equally to the density after recently). If we simply count the structures of completed sentences, following adjuncts look just like all other dependents, but dependency density distinguishes them. Fig. 8 shows densities for John likes green grapes without seeds, with a dotted arrow for without to show that it does not count.

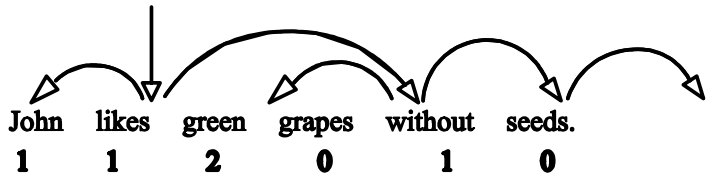


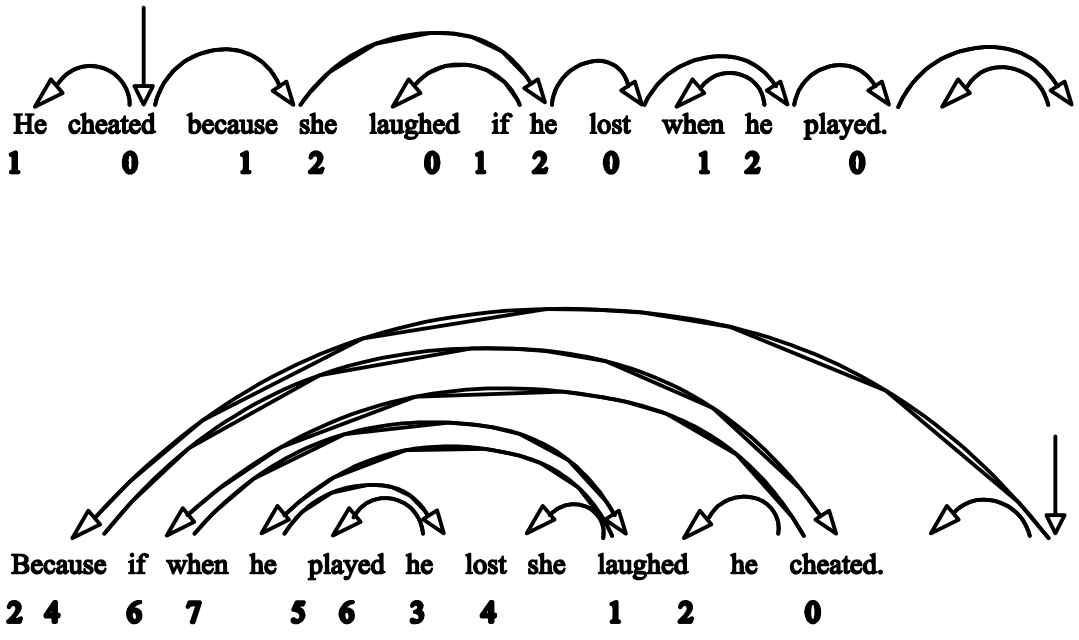
Fig. 8

This distinction between following adjuncts and other dependents is also the main reason for distinguishing density and distance, which are otherwise hard to separate conceptually. Unlike density, distance is as relevant to final adjuncts as to any other dependents - indeed, attachment of adjunct PPs is one of the main activities controlled by distance, as we saw earlier. In other respects the two measures are closely connected because distance measures the number of intervening words, and most words which separate a word from its parent add to the density of dependencies between them. The link is not perfect - apart from following adjuncts, we must recognise coordinated structures as contributing to distance but probably not to density - but uncomfortably close.

Density and distance must contribute separately to processing difficulty because quite short sentences can be hard to process when density builds up. This is easy to see in the contrast between two positions for adverbial clauses: before or after the superordinate verb. Consider the following series of sentences, which all contain exactly the same words and express exactly the same meaning:

(15) a He$_1$ cheated$_0$ because$_1$ she$_2$ laughed$_0$ if$_1$ he$_2$ lost$_0$ when$_1$ he$_2$ played$_0$.
b Because$_2$ she$_3$ laughed$_1$ if$_2$ he$_3$ lost$_1$ when$_2$ he$_3$ played$_1$, he$_2$ cheated$_0$.
c?# Because$_2$ if$_4$ he$_5$ lost$_3$ when$_4$ he$_5$ played$_3$ she$_4$ laughed$_1$, he$_2$ cheated$_0$.
d #Because$_2$ if$_4$ when$_6$ he$_7$ played$_4$ he$_5$ lost$_3$ she$_4$ laughed$_1$, he$_2$ cheated$_0$.

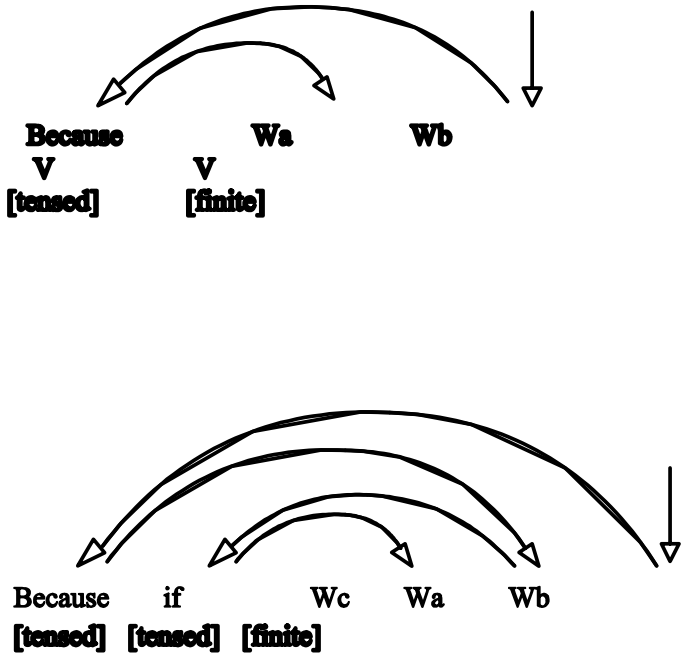
The density figures are shown after each word. The basis for calculating them is shown in Fig. 9, which contains the first and last sentences. Dependencies to later adjuncts are shown as dotted arcs to remind us that they do not contribute to density.



9
Fig. 9

In sentence (a) all the adjuncts because, if and when follow their parents, and the other sentences are formed by preposing these adjuncts one at a time. Each preposing makes the sentences harder to process, and the last one is virtually impossible to process without pencil and paper. This is easily explained in terms of the load created by open dependencies, with maxima rising from 2 to 3 to 5 to 7. The difficulty rises quite steadily in proportion to the number of open dependencies - a characteristic which will be important when we contrast such examples with centre-embedded constructions. We can also contrast density with distance by noting the very short distances involved in the last sentence - even the eight words between because and its parent are well within the normal bounds of processability, so the distance in itself cannot explain why this sentence is so hard.

No doubt this rather crude way of quantifying dependency density can be improved. Instead of scoring each open density equally, for example, we could take account of mergers among the anticipated words. In a sentence starting with Sweet green, for example, we are assuming that the parser merges the anticipated parents, so the load is correspondingly less than in, say, Yesterday green. Similarly for words like because, which need verbs as both parent and complement; in a sequence of such words such as because if it is possible to merge the complement of one word with the parent of the next one. (In other words, the if clause has to be subordinate to the because clause.) The state of the stack after hearing these two words is shown in Fig. 10.



10
Fig. 10

The question is whether we should score the density after if as four, in recognition of the open dependencies, or as three (because only three words are anticipated).

Furthermore we might consider weighting open dependencies according to the amount of information that they provide (Hudson 1993b). One difficulty with this idea is that it is unclear whether the load is increased by facts or by uncertainties. For instance, consider the loads after recently and John. Recently gives a great deal of information both about its parent and about its relation to the parent - it must be a time adjunct and the parent must be past tense. In contrast John gives very little information as it could be subject, extractee or `complement' of possessive 's (John came; John I can't stand or John's father came). Their effects on parsing are clearly different, but which is easier? Without an answer to this fundamental question it is premature to try and develop scoring systems (such as the one in Hudson 1993b).

6. Dependency decisions: centre-embedding

One of the most interesting processing questions for syntactic theory is why `centre-embedded' structures are impossible to process. Take a classic example like (from Gibson 1991: 52).

(16) #Men women dogs bite love like fish.

The structure for this sentence is shown in Fig. 11, with bite and love as adjuncts of women and men. (This is the normal WG analysis of bare relative clauses; see Hudson 1990 for further details.)

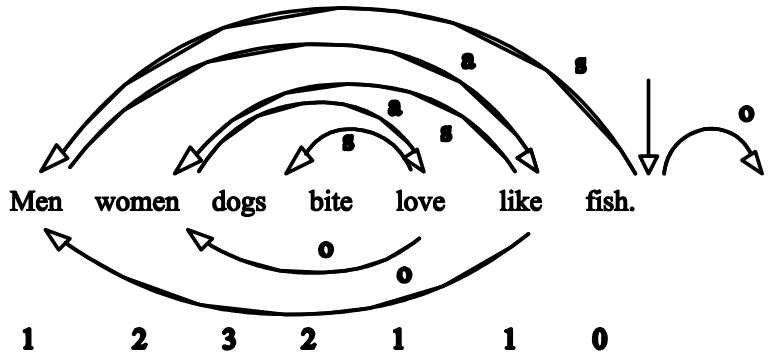


Fig. 11

One thing which is certain is that these difficulties cannot be explained in terms of either dependency distance or dependency density. The distances here are quite trivial: the longest dependency crosses only four words; and the maximum density is 3, which is well within the range of other sentences which are easy to process such as .

(17) Students$_1$ with$_2$ difficult$_3$ problems$_1$ need$_1$ time$_0$.

In this section I shall suggest that the problems of centre-embedding have nothing directly to do with memory overload, but are due to misplaced `intelligence' in the parser somewhat similar to `garden-pathing'.

Let us first consider some differences between the effects of centre-embedding and genuine overload problems which can be explained in terms of dependency distance and density.

a. A centre-embedded sentence is impossible to process even with practice on the same sentence, and even after a pencil-and-paper analysis. The best most of us can do with sentences like 0 is to memorise them as strings of words. As evidence that at least my mind is incapable of processing centre-embedded sentences, I can admit that in preparing a handout on this topic I found I had used the following example:

(22) #Books that students that lecturers recommend buy are often expensive.

It was only on drawing the structure diagram that I realised it was nonsense - the correct structure treats students as the object of recommend, whereas the pseudo-structure I had in mind must have justified recommend by its (non-) relation to books. In contrast, difficulties due to distance or density usually reduce with practice.

b. The problem with centre-embedded sentences does not increase gradually with structural complexity. We saw a gradual deterioration in the sentences in 0 as we preposed more and more adverbial clauses, but there is no such gradual change in centre-embedding. Sentence 0a is unproblematic, with a single relative clause, but one more relative clause inside this makes it completely impossible.

(18) a Men women love like fish.
b #Men women dogs bite love like fish.

c. The difficulty of centre-embedded sentences does not become apparent until after the first verb, by which time the number of open dependencies has already started to drop. When presented with sentences like 0b, people tend to comment on the second or third verb, rather than on the third noun as one would expect if this was overloaded; they say things like "What do you mean, `love'?" or "What's `love' doing there?" By the time love arrives, bite has already closed at least one open dependency (by providing a parent for dogs), so if overload had been the problem it should have led to a crash before bite.

d. Centre-embedding problems seem to be restricted to relative clauses, and seem to be caused by a relative clause modifying the subject of a relative clause. For example, Men women love like fish will not accept a relative clauses modifying women. This limit cannot be due to quantitative structural overload. For one thing, other kinds of subordinate clause have higher limits (which can presumably be explained in terms of dependency density). Take 0, for instance:

(19) aThe idea [that [whether or not it rains] doesn't matter] is nonsense.
b Then [if [when you get there] you find she's out] try next door.
c #Men [women [dogs bite] love] like fish.

Both the first two examples contain a subordinate clause at the start of a larger subordinate clause of the same type. (The subordinate clauses are bracketed.) To show the contrast with relative clauses I have added our main example bracketed in the same way. Furthermore, relative clauses are fine within other kinds of subordinate clause:

(20) aThe idea [that the books [our students read] should be censored] is absurd.
b[If [when the books [you ordered] arrive] you don't want them], let me know.

The problem, then, seems to be specific to relative clauses.

e. Furthermore, the problem is related to the number of nouns rather than to the number of relative clauses. The classic example crashes with two relative clauses, but it is easy to save it by removing the third noun without also removing the third relative clause. It is easier to see this if we supply relative pronouns:

(21) a #Men [who women [who dogs bite] love] like fish.
b Men [who women [who live here] love] like fish.
c Men [who women [looking for partners] love] like fish.

I find the second and third of these examples much easier to process than the first, and I guess this is because the lower relative clause has no separate subject. But in each case the overall structure in terms of clauses is the same.
f. Another way to improve a centre-embedded structure is to use free relative clauses. These are introduced by words like what and whoever which double up as both relative pronoun and antecedent. Being pronouns they are also nouns, so the number of nouns is not affected, but replacing any one of the first two nouns in a three-noun sequence improves processing considerably.

(22) a#The money [that the books [that John bought] cost] surprised him.
b [What the books [that John bought] cost] surprised him.
c The money [that [what John bought] cost] surprised him.
d [What [what John bought] cost] surprised him.

It seems clear, then, that centre-embedding problems are not caused by the number of subordinate clauses nested in one another, but have something to do with the peculiarities of relative clauses. In particular we must focus on how our parser handles the start of a relative clause. The gist of my explanation is that the parser merges the anticipated verb of the first relative clause with that of the next one, so that when the latter arrives it is not expected and cannot be accommodated.

We start with the simplest input, three bare nouns: Men women dogs. How does the parser handle these? All that emerges from the grammar is an entry for

each word which says that it needs a parent; crucially, of course, there is no mention of any adjuncts, and in particular no mention of relative clauses. However on the basis of the information available in the grammar the parser can guess after reading Men that the parent will be the main verb (though as noted earlier it could be possessive 's; I ignore this possibility as irrelevant here). The same guess may apply to women - after all, Men could be an extractee of this verb and women its subject as in Men women love. Suppose it does; the parser now has one cell of the stack containing the following string, in which both nouns depend on Wa:

Stack = (men women: Wa)

Then comes dogs. As a well-informed parser, ours knows that two initial noun dependents is the limit for one verb, but there is no other way to attach dogs to the words already in the stack, so it creates a new cell.

Stack = (dogs: Wb)
(men women: Wa)

The next word is bites, which combines easily with dogs (as subject) but also with women (as parent), thereby integrating the two cells:

Stack = (men women (dogs bite): Wa)

but the resulting string only has a single anticipated word, Wa to accommodate the two verbs which remain, love and like. This is why hearers are so puzzled by the number of verbs, rather than by the nouns.

Now consider a very different pattern of relative clauses, where the relative clauses are signalled by pronouns. (What follows will apply equally to that, regardless of whether this is classified as a `complementiser' or as a pronoun.) Take the sequence Men who women who dogs bite. The essential fact about who is that it takes a preceding noun as its parent and a following tensed verb as its complement. Therefore after men who we have:

Stack = (men (who: Wb) Wa)

(Wa is the anticipated parent of men, i.e. the sentence's main verb.) As a tensed verb, Wb accepts the next word, women, as its subject, thereby integrating women into the stack phrase:

Stack = (men (who women: Wb) Wa)

The crucial step is the next one. The second who also needs a tensed verb Wc as parent, and merging Wc with Wb is so tempting that the parser succombs.

Stack = (men (who women (who: Wc)= Wb) Wa)

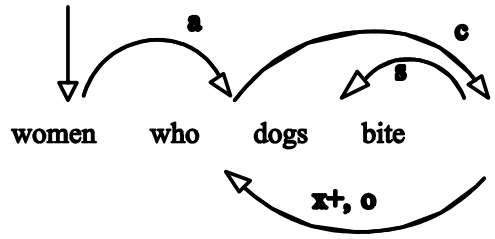
What makes it possible is the way in which dependencies are assigned to the surface

structure. The general principle is very simple (Hudson 1994): every word (except the phrase ancestor) must have just one parent in the surface structure, which links it to the phrase ancestor by a tangle-free chain of dependencies (i.e. a chain which never crosses any other word's chain). The sentence's surface structure consists of all the chains from its constituent words. This means that if a word has more than one parent, either of them could in principle be in the surface structure; for example, in a subject-to-object raising example like I expect it to rain, there are two words which share it as dependent: it is object of expect and subject of to (rain). Either of these dependencies could satisfy the need for a connected surface structure. In the case of relative clauses, we shall see that this flexibility has dire consequences when combined with the parser's determination to merge expected words.

Returning to the analysis of Men who women who, the second who contracts three distinct sets of dependencies, and the question the parser has to solve is which to put into the surface structure. It is an adjunct of women, the parent of Wc (the following tensed verb) and also the extractee of Wc. All this information is easily available to the parser from the entry for who:

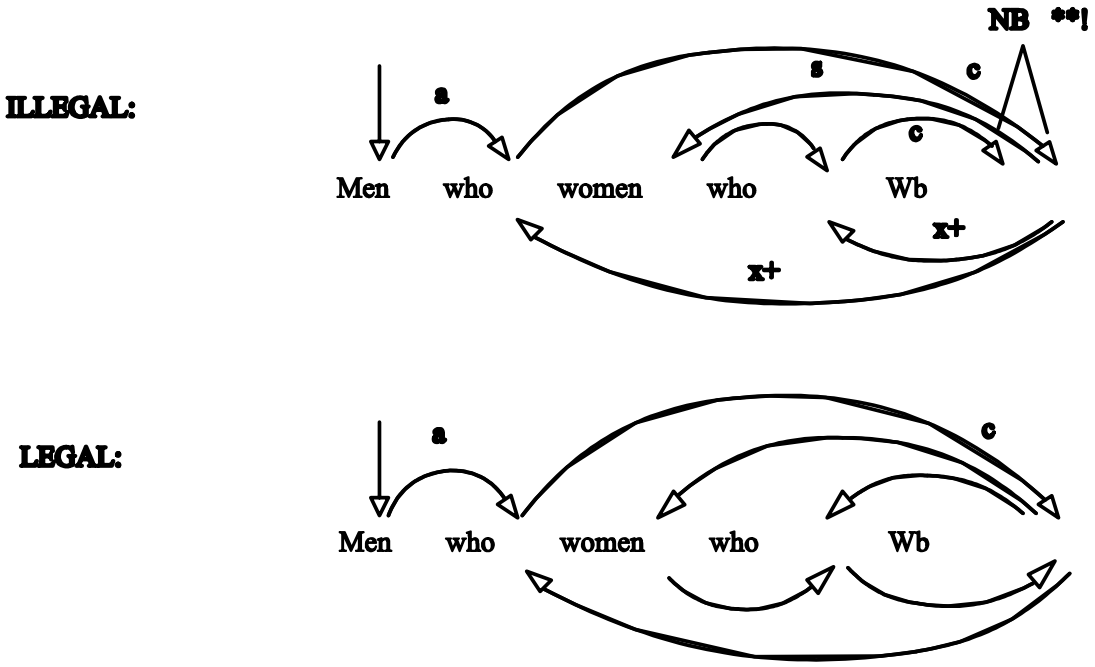
(23) a who has a complement.
b The complement of who is a tensed verb.
c who is the extractee of its complement.

In a straightforward unembedded example the surface structure would show who as dependent of women and the tensed verb as dependent of who, with the extractee relation relegated to the extra dependencies. This is shown in Fig. 12 for Women who dogs bite ....



12
Fig. 12

If the surface structure had to be like this the question of merging Wb and Wc would not arise because the merged verb would then have two surface parents: the first who and the second one. But there is another option: to promote the extractee relation to the surface and relegate the other two to the extras. These two possibilities are shown in Fig. 13.



13
Fig. 13

Therefore the parser guesses (wrongly) that Wb can merge with Wc, and moves on to the next word with the following stack containing only one anticipated verb (Wb) shared by two different relative clauses:

(men (who women (who: Wb)) Wa)

The rest of the disaster follows in an obvious way. Wb is identified with bite, so after Men who women who dogs bite only Wa is left, and is used up by love, leaving no role for the last verb like.

In both the unprocessable examples, the problem lay in a wrong guess made by the parser, whose result was that the parser expected only one verb where two were actually needed. We now consider the third type of relative clause, free relatives, to see why it is immune to this problem. We have to change to a different example:

(24) What what John bought cost surprised him.

This example escapes the problem because the entry for what leaves less flexibility to be misused by the parser. Whereas who takes a preceding noun as its parent, what is like any other noun in its parent needs; so if it is at the start of a sentence, it naturally needs a following parent. After the first what the parser needs Wa (the main verb) as its parent and Wb, a tensed verb, as its complement verb.

((what: Wa) Wb)

The second what has similar needs, but must take Wb as its parent in order to satisfy

the latter's need for a subject. The grammar makes no provision for what using the same word as its parent and its complement, so this merger of Wb with what's parent rules out any merger of Wb with Wc, the needed complement of the second what. The parser emerges from this stage in fine form, with three needed verbs:

((what (what: Wc) Wb) Wc)

Everything is straightforward from here on: bought is Wc, cost Wb and surprised Wa:

((what (what John bought:) Wa) Wb)
((what (what John bought) cost:) Wa)
((what (what John bought) cost) surprised him:)

We have explained so far why ordinary relative clauses cannot be `centre-embedded' and why free relatives can. However we can push the explanation a little further. I have the impression that some ordinary relatives do in fact allow centre-embedding. Take the following examples:

(25) a All I want is a room somewhere.
b Everything I bought was wrong.
c People I know are mostly liberals.

What the examples have in common is a relative clause whose antecedent, exceptionally, leads one to expect a relative clause. This expectation is very strong after all, which is almost like what in its need for a tensed complement; a participle, for example, will not do (*all bought, *all lying on the table). After everything the expectation is quite strong, though a wider range of modifiers is possible, and similarly for people (and other very general nouns such as things). If these observations are true then they raise interesting theoretical problems for the sharp distinction that we all assume, and which has played an important part in this discussion, between adjuncts (which are not anticipated by their parents) and valents (which are).

However for present purposes they raise the possibility that some centre-embedding may be better than others, because antecedents like these are somewhere between what and men in the amount of guidance they give to the parser. This prediction may be confirmed by the following examples, which seem a lot easier to parse than the earlier ones.

(26) a All [the man [who came] did] was to sit around talking.
bEverything [everyone [I know] tells me] confirms my impression.
cThings [people [I talk to]] tell me suggest there's a crisis brewing.

If these are easier, the reason may be that the first noun guides the parser to expect a relative clause, so the second noun can immediately be accepted as the relative clause's subject and the rest of the analysis goes smoothly.

To summarise the discussion of centre-embedded sentences, I have suggested

that these are difficult because of systematic parsing mistakes rather than because of memory overload. The mistakes arise from overapplication of a principle which in other cases is very sensible: try to merge the anticipated parents of different input words. This principle can only apply when the mergers are allowed by the general grammar, and in particular by the word-order principles; but when structures are as rich as they are in relative clauses the general word-order principles allow slightly too many options. The parser works very smoothly most of the time; in fact, to the extent that the same constraints apply to the producing system as to the parser, we could go so far as to say that the parser can never be confronted with centre-embedded sentences, so it works well **all** the time. The analysis offered has explained all the characteristics of centre-embedding structures except the very first one: the fact that such structures are impossible to process even after practice. This fact suggests a remarkable lack of flexibility in our processing system: the only way in which we can build a structure is by using the normal parser. Even theoretical linguists cannot work it out with pencil and paper, memorise it and retrieve it fully formed. Nor can we interfere in this normal processing by preventing our parser from going down the garden path. Why? It seems unlikely that linguistic analysis will be as much help with this question as it has been with the previous ones.

7. Dependency direction: head-final structures

We have seen that the direction of dependencies is important for processing in that adjuncts which follow their parent do not contribute to dependency density, whereas those that precede the parent do. The discussions of parsing difficulty (especially Frazier & Rayner 1988) have raised more general questions: are all languages processed in the same way, and is there any general difference in ease of parsing between head-initial and head-final languages? Most discussions of parsing assume a single universal parser, equally applicable to all languages, but this need not be so - and indeed common sense would suggest, for example, that a parser suited to a language where word order carries a lot of information would be unsuitable for one where the same information is carried by case-markings. On the question of ease, common sense suggests that head-initial structures are easier because the head provides the framework which integrates all the dependents into a single phrase; but common sense may be heavily influenced in this case by the fact that English itself is head-initial. The fact is that languages of the world are divided roughly equally between the two types, so the apparent advantages of head-initial structures must be balanced by advantages of head-final structures.

In this section I shall consider a very small amount of data from head-final languages which suggests that Dutch and German speakers use the same kind of parser as English speakers, but that Japanese speakers may be different.

German relative clauses seem to suffer the same centre-embedding problems as the English ones that are introduced by relative pronouns. Take the following series (from Gibson 1991) for example.

(27) a Der Mann schwamm.
`The man swam.'
b Der Mann, [den die Frau sah], schwamm.

`The man [who the woman saw] swam.

c #Der Mann, [den die Frau, [die der Hund biß]] sah schwamm.

#`The man [who the woman [who the dog bit] saw] swam.

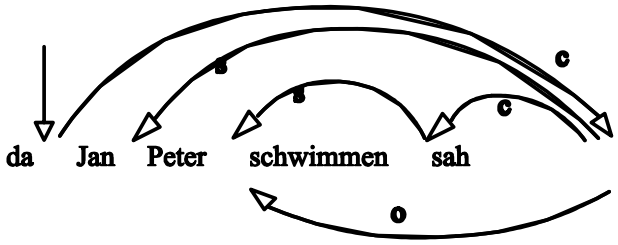
As with the English examples there is a qualitative difference between the example with just one relative clause and the one with two. It seems reasonable to assume that the same explanation applies here, in spite of the fact that all subordinate clauses in German are verb-final. However it is important to remember that German is only partly head-final (just as English is only partly head-initial); in particular, clause-subordinators such as relative pronouns are all head-initial (i.e. they all stand at the start of the subordinate clause), just like their counterparts in English. The flow of information to the parser would be so different if relative pronouns were clause-final that I doubt whether they could match the centre-embedding problems of English.

The effects of dependency density also seem to be similar in German and Dutch in spite of the different structures involved. One of the most interesting (and best known) facts about these languages is that although subordinate clauses have to be verb-final in both languages, they have different word-order rules for embedded subordinate clauses. Whereas (most dialects of) German avoid tangling by nesting the dependencies inside each other, Dutch allows non-nested dependencies. That is, a verb which depends on a `clause-final' verb precedes it in German, but can follow it in Dutch. (The facts are presented helpfully, with interesting analytical suggestions, in Baker 1994 and van Noord & Bouma 1995; for discussion of the WG analysis, see Hudson 1995b.) For example:

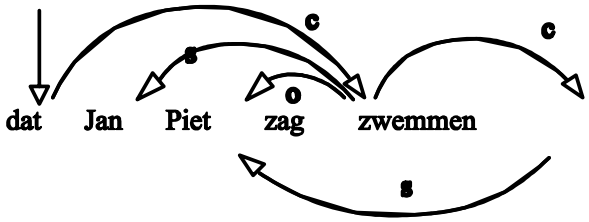
(28) German daß Jan Peter schwimmen sah.
Dutch dat Jan Piet zag zwemmen.
`that Jan saw Peter swim'

The main problem is to explain the Dutch `cross-serial dependencies' which involve tangling in a conventional dependency analysis - for example, the link between zwemmen and its subject Piet crosses the one from zag to its subject, Jan. A WG analysis is possible if we assume that Piet depends on zag as well as on zwemmen. On this assumption either of the dependencies may be in the surface structure, and tangling can be avoided. Fig. 14 shows dependency structures for these two sentences.

GERMAN



DUTCH



14
Fig. 14

In these sentences the word for `Peter' functions both as object of `see' and as subject of `swim', and it is fairly uncontroversial (outside GB circles) to analyse it as raised out of the lower clause. This allows it to take its place in surface structure among the dependents of `see', so the order of the two verbs does not matter. However it is not only the infinitive's subject that is positioned in this way, but all its other dependents as well. For example, take: dat Jan Piet de kinderen zag helpen, `that Jan saw Piet help the children'. Here `the children, which is the object of `help', is also positioned as a dependent of `saw'. The obvious way to handle these word order facts is to introduce `universal raising' triggered by verbs such as `see' and auxiliaries (Baker 1994, Hudson 1995b). This allows the parser to attach all the initial NPs to the same anticipated verb, with a view to sorting out the extra dependencies later.

According to Gibson (1991), Dutch sentences become harder to process as the number of subordinate verbs increases. He quotes the following examples and judgements (with density numbers supplied by me):

(29) a dat$_1$ Jan$_2$ Piet$_3$ de$_5$ kinderen$_4$ zag$_1$ helpen$_1$ zwemmen$_0$.
that Jan saw Piet help the children swim
b?# dat$_1$ Jan$_2$ Piet$_3$ Marie$_4$ de$_6$ kinderen$_5$ zag$_1$ helpen$_1$ laten$_1$ zwemmen$_0$.
that Jan saw Piet help Marie make the children swim
c# dat$_1$ Jan$_2$ Piet$_3$ Marie$_4$ Karel$_5$ de$_7$ kinderen$_6$ zag$_1$ helpen$_1$ laten$_1$ leren$_1$ zwemmen$_0$.
that Jan saw Piet help Marie make Karel teach the children to swim

The steady increase in difficulty is what we should expect from difficulties due to dependency density, and indeed the densities are high enough to explain the processing difficulties. In contrast, of course, the English translations are all easy to

process because of the consistently head-initial order, so to this extent we might conclude that head-final structures are harder to process.

Fortunately there is also some experimental data which allows us to compare the difficulties of equivalent German and Dutch sentences (Bach et al 1986). In this research native speakers were presented with sentences of varying degrees of complexity and a range of judgements and questions to test how easy they were to understand. It turned out that Dutch sentences were easier than equivalent German ones when more than two non-finite verbs were stacked at the end of the sentence as in:

(30) aJeanine$_1$ heeft$_1$ de$_3$ mannen$_2$ Hans$_3$ de$_5$ paarden$_4$ helpen$_1$ leren$_1$ voeren$_0$.
bJohana$_1$ hat$_1$ den$_3$ Männern$_2$ Hans$_3$ die$_5$ Pferde$_4$ füttern$_4$ lehren$_2$ helfen$_0$.
Johana has helped the men to teach Hans to feed the horses.

The density numbers already offer an explanation for the Dutch advantage: all the numbers are the same except near the end, where the German numbers for `feed' and `teach' are higher. These figures rest on the assumption that a German parser will delay attaching Hans until its parent lehren arrives, whereas the Dutch parser attaches it directly to the first verb; and similarly for `the horses'. Both the parsers have access to the grammar for the language concerned, so they work differently because Dutch has `universal raising' and German does not. Furthermore, German has case differences which are absent in Dutch, which presumably discourages German parsers from merging the verbs anticipated by two nouns if the latter's cases would conflict. If parsing is made easier by merging of anticipated words, the Dutch parser ought to cope better than the German parser with long sentences like the examples just given.

A tiny amount of Japanese data from Gibson (1991) suggests that here too the effects of dependency density may be the same. The data[3] are as follows:

(31) a Jon-wa$_1$ Fred-ga$_2$ Biru-o$_3$ mita-to$_2$ omotteiru$_0$.
`John thinks that Fred saw Bill'
b?#Jon-wa$_1$ Mary-wa$_2$ Fred-ga$_3$ Biru-o$_4$ mita-to$_5$ sinziteiru-to$_2$ omotteiru$_0$.
`John thinks that Mary believes that Fred saw Bill'
c#Jon-wa$_1$ Mary-wa$_2$ Fred-wa$_3$ Sam-ga$_4$ Biru-o$_5$ mita-to$_6$ omotteiru-to$_4$
sinziteiru-to$_2$ omotteiru$_0$.
`John thinks that Mary believes that Fred thinks that Sam saw Bill'

The unprocessable density score of six is roughly the same as the score of seven which had the same effect in Dutch and in the English nested adverbial clauses Because if when ....This convergence across different languages is encouraging,

[3]Gibson's data contain ... Fred-ga Biru-o sukida ..., which he translates as `Fred likes Bill'. I gather this is poor Japanese, so I have replaced it with a more straightforward clause ... Fred-ga Biru-o mita ..., `Fred saw Bill'. I have also replaced all but the last -ga in each of Gibson's examples by -wa, and changed his `#' and `##' to `?#' and `#' on the advice of two Japanese speakers, Kensei Sugayama and So Hiranuma.

especially since Japanese is so consistently head-final.

8. Conclusion

We have considered three sources of parsing difficulty, two quantitative (dependency distance and dependency density) and one qualitative (bad `dependency decisions' which lead to centre-embedded failures). Dependency direction (head-initial or -final) is also highly relevant to parsing, but the difficulties that it produces can be explained in terms of the two quantitative measures, so direction in itself is not directly relevant - for example, there appears to be no need to assume different parsing strategies for head-initial and head-final languages.

References

Abney, Steven (1989 A computational model of human parsing. Journal of Psycholinguistic Research, 18:

Abney, Steven & Johnson, Mark (1991) Memory requirements and local ambiguities of parsing strategies. Journal of Psycholinguistic Research 20: 233-50.

Bach, Emmon; Brown, Colin & Marslen-Wilson, William (1986) Crossed and serial dependencies in German and Dutch: a psycholinguistic study. Language and Cognitive Processes 1: 249-62.

Baker, Kathryn (1994) An extended account of `modal flip' and partial verb phrase fronting in German. = CMU-LCL-94-4. Pittsburgh: Carnegie Mellon University, Laboratory for Computational Linguistics.

Dryer, Matthew (1988) Object-verb and adjective-noun order: dispelling a myth. Lingua 74: 185-217.

Dryer, Matthew (1992) The Greenbergian word order correlations. Language 68: 81-138.

Fraser, Norman (1993) Dependency parsing. London University PhD thesis.

Fraser, Norman & Hudson, Richard (1992) Inheritance in Word Grammar. Computational Linguistics 18: 133-59.

Frazier, Lyn (1979) On comprehending sentences: Syntactic parsing strategies. University of Connecticut PhD thesis.

Frazier, Lyn & Rayner, Keith (1988) Parameterizing the language processing system: left- vs. right-branching within and across languages. In John Hawkins (ed.) Explaining Language Universals. Oxford: Blackwell, 247-79.

Gibson, Edward (1991) A Computational Theory of Human Linguistic Processing: Memory limitations and processing breakdown. Carnegie Mellon University PhD and Center for Machine Translation Report CMU-CMT-91-125.

Hawkins, John (1990) A parsing theory of word order universals. Linguistic Inquiry 21: 223-61.

Hawkins, John (1993) Heads, parsing and word-order universals. In Greville Corbett, Norman Fraser & Scott McGlashan (eds.) Heads in Grammatical Theory. Cambridge: Cambridge University Press.

Hudson, Richard (1984) Word Grammar. Oxford: Blackwell.

Hudson, Richard (1989) Towards a computer-testable Word Grammar of English. UCL Working Papers in Linguistics 1: 321-339.

Hudson, Richard (1990) English Word Grammar. Oxford: Blackwell.

Hudson, Richard (1992) Raising in syntax, semantics and cognition. In Iggy Roca (ed.) Thematic Structure: Its role in grammar. Berlin: Mouton de Gruyter, 175-98.

Hudson, Richard (1993a) Do we have heads in our minds? In Greville Corbett, Norman Fraser & Scott McGlashan (eds.) Heads in Grammatical Theory. Cambridge: Cambridge University Press, 266-91.

Hudson, Richard (1993b) Dependency counts. In Eva Hajičová (ed.) Functional Description of Language. Prague: Faculty of Mathematics and Physics, Charles University, 85-115.

Hudson, Richard (1993a) Recent developments in dependency theory. In J. Jacobs, W. Sternefeld, T. Vennemann & A von Stechow (eds.) Syntax. An International Handbook of Contemporary Research. Berlin: Walter de Gruyter, 329-38.

Hudson, Richard (1994) Discontinuous phrases in dependency grammar. UCL Working Papers in Linguistics 6: 89-124.

Hudson, Richard (1995a) Identifying linguistic foundations for lexical research and dictionary design. In Don Walker (ed.) Automating the Lexicon. Oxford: Oxford University Press, 21-51.

Hudson, Richard (1995b) HPSG without PS? MS

Hudson, Richard (1996) Really bare Phrase structure = Dependency structure. Studies in English Usage and English Language Teaching. Kyoto: Yamaguchi, 17.

Pickering, Martin & Barry, Guy (1991) Sentence processing without empty categories. Language and Cognitive Processes 6: 229-59.

Tesnière, Lucien (1959/1966) Eléments de Syntaxe Structurale. Paris: Klincksieck.

Tomlin, Russel (1986) Basic Word Order: Functional principles. London: Croom Helm.

van Noord, Gertjan & Bouma, Gosse (1995) Dutch verb clustering without verb clusters. MS.

Vennemann, Theo (1974) Theoretical word order studies: results and problems. Papiere zur Linguistik 7: 5-25.

Vennemann, Theo (1984) Typology, universals and change of language. In Jacek Fisiak (ed.) Historical Syntax. Berlin: Mouton, 593-612.

APPENDIX

Some statistics on dependencies in texts