A Trainable Bracketer for Noun Modifiers

Ken Barker

School of Information and Technology Engineering
University of Ottawa
Ottawa, Canada K1N 6N5
kbarker@site.uottawa.ca

Abstract. Noun phrases carry much of the information in a text. Systems that attempt to acquire knowledge from text must first decompose complex noun phrases to get access to that information. In the case of noun compounds, this decomposition usually means bracketing the modifiers into nested modifier-head pairs. It is then possible to determine the semantic relationships among individual components of the noun phrase.

This paper describes a semi-automatic system for bracketing an unlimited number of adjectival or nominal premodifiers. Since the system is intended to start processing with no prior knowledge, it gets trained as it brackets. That is, it starts from scratch and accumulates bracketing evidence while processing a text under user supervision.

Experiments show that generalizations of the structure of complex modifier sequences allow the system to bracket previously unseen compounds correctly. Furthermore, as more compounds are bracketed, the number of bracketing decisions required of the user decreases.

1 Introduction

The HAIKU system performs semi-automatic semantic analysis on three levels: between clauses, within clauses and within noun phrases. In the absence of precoded semantics HAIKU enlists the help of a cooperating user who oversees semantic decisions. To lessen the burden on the user, the system first attempts automatic analysis by comparing input structures to similar structures in the text that have already been semantically analyzed. Since it does not have access to a large body of pre-analyzed text, HAIKU starts processing from scratch for a text and acquires the needed data incrementally.

The clause level relationship (CLR) analyzer assigns semantic relationship labels to connected clauses based on their syntactic features. CLR analysis is described by Barker & Szpakowicz [2]. Within a clause, HAIKU assigns case relationships to arguments syntactically connected to the main verb. By comparing syntactic argument markers to syntactic patterns previously encountered, the case analyzer learns to make better analyses more autonomously. Delisle *et al.* [6] and Barker *et al.* [3] describe the case analyzer and HAIKU's case system. Within a noun phrase, HAIKU assigns noun modifier relationships (NMRs) to each of the head noun's premodifiers and postmodifiers. Similar previous instances of modifier-head pairs allow the NMR analyzer to improve the accuracy of its analyses while gradually decreasing its dependence on user input.

As a prologue to NMR analysis, the system must bracket lists of premodifiers into modifier-head pairs. Like the other components of HAIKU, the bracketer is semi-automatic. It uses previous modifier-head pairs to inform bracketing decisions, relying on the user to supply decisions when previous evidence is insufficient. As more noun phrases are bracketed, this reliance on the user decreases.

The subject of this paper is the subsystem of the NMR analyzer that brackets noun premodifiers. Section 2 looks at ideas and work related to noun phrase analysis and bracketing. Section 3 describes the syntactic input to the bracketer while section 4 places the bracketer in the larger context of noun phrase analysis in HAIKU. The algorithm for bracketing general sequences of premodifiers appears in section 5. Section 6 summarizes bracketer results that have been collected during the experimental evaluation of HAIKU.

2 Background and Related Work

2.1 Noun Compounds

A head noun along with one or more noun premodifiers is often called a noun compound. There are several different types of noun compounds: those whose meaning is derivable from the individual components and those whose meaning isn't; those that are a semantic subclass of the head noun and those that aren't; etc. HAIKU deals with the semantics of a particular kind of compound, namely compounds that are *transparent* and *endocentric*.

A transparent compound is one whose meaning can be derived from the meaning of its individual elements. For example, *laser printer* is transparent (a *printer* that uses a *laser*), whereas *guinea pig* is *opaque* since there is no obvious direct relationship to *guinea* or to *pig*.

An endocentric compound is a hyponym of its head. For example, *desktop computer* is endocentric because it is a kind of *computer*. A *bird brain* is *exocentric* because it does not refer to a kind of *brain*, but rather to a kind of person (whose brain resembles that of a bird).

Since HAIKU is intended to analyze technical text, the restriction to transparent endocentric compounds should not limit the utility of the system. The experiments described in section 6 found no opaque or exocentric compounds in the texts.[1] For these texts at least, little is lost by not dealing with such compounds.

2.2 Semantic Relationships in Noun Phrases

Most of the research on semantic relationships between nouns and their modifiers deals with noun-noun compounds. Usually researchers propose finite lists of semantic relationships that a compound may express.

---

[1] In fact, many exocentric compounds pose no particular problem for HAIKU's noun modifier relationship analyzer. The main difficulty is in recovering taxonomic information (section 4.3). Options for more flexible handling of exocentric compounds are under investigation.

Levi [15] argues that semantics and word formation cause noun-noun compounds to constitute a heterogeneous class. Instead, she looks at a class of opaque modifier-noun compounds, where the modifier may be a noun or a nominal, non-predicating adjective. For this more homogeneous class Levi offers nine semantic labels. According to her theory, these labels represent underlying predicates that were deleted during compound formation. George [9] disputes the claim that Levi's non-predicating adjectives never appear in predicative position.

Warren [21] describes a multi-level system of semantic labels for noun-noun relationships. The "Major Semantic Class" level consists of fourteen general relationships. Warren [22] extends the earlier work to cover adjective premodifiers as well as nouns. The similarity of the two lists suggests that many adjectives and premodifying nouns can be handled by the same set of semantic relations.

Computer systems for assigning semantic relationships to modifier-noun compounds usually depend on hand-coded semantic knowledge. Leonard [14] describes a system that assigns relationships to noun-noun compounds based on a dictionary that includes taxonomic and meronymic (part-whole) information, information about the syntactic behaviour of nouns and information about the relationships between nouns and verbs. Finin [8] produces multiple semantic interpretations of modifier-noun compounds (where a modifier is either a noun or a nominal, non-predicating adjective). The interpretations are based on precoded semantic class information and domain-dependent frames describing the roles that can be associated with certain nouns. Ter Stal [19] describes a system that identifies concepts in text and unifies them with structures extracted from a hand-coded lexicon containing syntactic information, logical form templates and taxonomic information for each word.

In an attempt to avoid the hand-coding required in other systems, Vanderwende [20] automatically extracts semantic features of nouns from online dictionaries. Combinations of these features imply particular semantic interpretations of the relationship between two nouns in a compound.

2.3 Bracketing Noun Compounds

When a head noun has more than one premodifying noun or adjective, the sequence has internal structure and requires bracketing. For example, phrase (1) is *left-branching* and has the bracketing shown in (2); phrase (3) is *right-branching* and has the bracketing shown in (4).

(1) *laser printer manual*
(2) *((laser printer) manual)*
(3) *desktop laser printer*
(4) *(desktop (laser printer))*

Several researchers have proposed empirical solutions to the bracketing problem. Liberman & Sproat [16], Pustejovsky *et al.* [17] and Resnik [18] follow a similar approach: for a given sequence X-Y-Z, compare the number of occurrences of X-Y in isolation in a corpus with the number of occurrences of Y-Z. Lauer [12] calls this the *adjacency model* and offers a different model, the *dependency model*. The

dependency model compares the number of occurrences of X-Y to the number of occurrences of X-Z (instead of Y-Z). In Lauer's bracketer, the dependency model outperforms the adjacency model.

Ter Stal [19] confirms earlier results of Resnik [18] and Lauer & Dras [13] that between 60% and 70% of noun-noun-noun compounds in text are left-branching. He does not use bracketing in his compound analyzer, however, since his lexicon contains enough semantic information for direct identification of complex concepts.

HAIKU's bracketer differs from other approaches in three ways. First, it can be used to bracket phrases with an unlimited number of both adjective and noun premodifiers. Second, the evidence for bracketing can come from previous occurrences of a modifier-noun pair in isolation, previous occurrences of a pair *within* other compounds, as well as occurrences of a pair whose modifier and head are not adjacent in the text (see section 5). Finally, since it is interactive, the bracketer can bracket phrases even when there is no previous evidence.

3 Syntactic Input to the Bracketer

There are various ways to identify noun phrases in text. HAIKU performs semantic analysis on sentences parsed by the DIPETT parser [5]. For each noun phrase in a sentence DIPETT produces an *np* structure whose format appears in Figure 1.

The parts of the structure relevant to semantic analysis are *attributes*, *premodifiers*, the *head noun* and *np postmodifiers*. All adjectives that precede any premodifying nouns appear in the list of *attributes*. Premodifying nouns and adjectives that appear before the head but after the first premodifying noun appear in *premodifiers*. The *np postmodifiers* may include any number of postmodifying prepositional phrases as well as optional appositives or relative clauses.

The attributes and premodifiers must be bracketed before NMR assignment since some of the modifiers may modify other modifiers, not the head directly.

I said in section 2.3 that other systems that bracket premodifiers usually deal with the problem of bracketing three nouns: two premodifiers and a head (X-Y-Z). These systems compare occurrences of X-Y with occurrences of either Y-Z or X-Z (depending on the model). Since these pairs may occur infrequently in a text, it would be useful to generalize (X-Y-Z) and look for occurrences of the generalization. For example, Lauer [12] generalizes the nouns X, Y and Z to the Roget's Thesaurus categories that contain them: R_X, R_Y and R_Z. Instead of looking for other occurrences

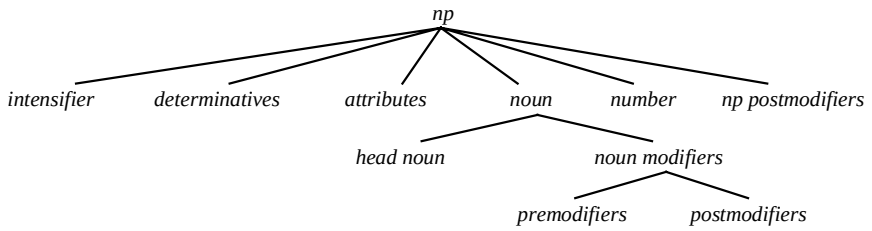

Figure 1. Syntactic structure of noun phrases

of X-Y and X-Z in the text, Lauer's bracketer looks for occurrences of U-V and U-W such that $R_U = R_X$, $R_V = R_Y$ and $R_W = R_Z$. The technique limits generalization to nouns that occur in Roget.

The problem of generalization is compounded in HAIKU since it deals with sequences of more than three words and allows adjectives as premodifiers. And since the fundamental tenet of HAIKU is to avoid precoding semantic information, a semantic generalization on the words is not readily available. What is needed is some way of increasing the number of hits when searching for previously analyzed noun phrases that can help analyze a new noun phrase.

Consider phrase (5) and a reasonable bracketing for it in (6).

(5) *dynamic high impedance vocal microphone*
(6) *(dynamic ((high impedance) (vocal microphone)))*

Each non-atomic element of each bracketed pair can be considered a *subphrase* of the original phrase. Given the bracketing in (6) the subphrases for phrase (5) are phrase (5) itself as well as the subphrases in (7). Assuming the compounds are endocentric, the subphrases will be well-formed.

(7) *high impedance vocal microphone*
high impedance
vocal microphone

Each subphrase consists of one or more modifiers and a head local to the subphrase. Local heads in (6) are *microphone* (the head of three subphrases) and *impedance*. The subphrases are generalized by reducing modifiers and modificands to their local heads. If this concept of reduction is applied to a bracketed phrase, the result is a set of reduced subbracketings of the original phrase. The reduced subbracketings of (6) appear in (8).

(8) *(dynamic microphone)*
(impedance microphone)
(high impedance)
(vocal microphone)

The reduced subbracketings together are a structural generalization of the original noun phrase. Instead of simply memorizing complete noun phrases and their analyses, the system stores the subbracketings. This allows it to analyze different noun phrases that have only subbracketings in common with previous noun phrases. Section 5 will show how the reduced subbracketings help automate the bracketing procedure.

4 Noun Modifier Relationship Analysis

4.1 Noun Modifier Relationships

Bracketing the premodifiers produces a number of modifier-head pairs, each of which will be assigned a noun modifier relationship from the list in Table 1. The set of relationships is based primarily on research on the semantics of noun compounds,

Agent (AGT)	Instrument (INST)	Property (PROP)
Beneficiary (BENF)	Located (LED)	Purpose (PURP)
Cause (CAUS)	Location (LOC)	Result (RESU)
Container (CTN)	Material (MATR)	Source (SRC)
Content (CONT)	Object (OBJ)	State (STAT)
Destination (DEST)	Possessor (POSS)	Time (TIME)
Equative (EQUA)	Product (PROD)	Topic (TOP)

Table 1: The noun modifier relationships

such as the research described in section 2.2. Barker [1] offers definitions, paraphrases and examples for each NMR.

4.2 Naming NMR Arguments

The reduced subbracketings described in section 3 are used internally to help with the analysis of new noun phrases. The NMRs, however, apply to complete modifier-head sequences, not to reduced pairs. There are two NMRs for (9), since there are two modifier-head pairs:

(9) *small (gasoline engine)*
(10) *gasoline-engine*
(11) *small-(gasoline engine)*

The NMR for (10) is Instrument, since *gasoline* is used by *gasoline engine*. Note that *gasoline* is an instrument of *gasoline engine*, not *engine* in general. The NMR for (11) is Property: *small* is a simple property of *small gasoline engine*, but not generally a property of *engine* or even of *gasoline engine*. The following structures are the semantic output for (9). When an argument name is built from more than one word, the individual words are concatenated with an underscore.

```
nmr_struct(inst,
           gasoline,
           gasoline_engine)

nmr_struct(prop,
           small,
           small_gasoline_engine)
```

The reduced pairs for (9) will be stored and used by the bracketing and NMR assignment algorithms to guide future processing:

(12) *(gasoline engine)*
(small engine)

4.3 Premodifiers as Classifiers

Warren [21] distinguishes two roles of modifiers orthogonal to the semantic relationship between premodifier and head: premodifiers as *classifiers* and premodifiers as *identifiers*. The identifying role is to point at some thing in the world

referred to by the compound. As an identifier the modifier is relevant to such tasks as anaphora resolution and concept identification.

In NMRA the classifying role of modifiers in endocentric compounds allows us to glean taxonomic information directly from the semantic structures. Whenever an NMR is assigned between a modifier and modificand, there is an implicit hierarchical or *isa* structure as well. The reduced modifier subbracketings for (13) appear in (14); the corresponding *isa* structures appear in (15).

(13) *((laser printer) stand)*

(14) *(laser printer)*
(printer stand)

(15)
```
isa(laser_printer_stand, stand)
isa(laser_printer, printer)
isa(printer_stand, stand)
```

5 Bracketing Noun Modifiers

5.1 The Algorithm

The algorithm for noun premodifier bracketing handles modifier sequences of any length by dealing with a window of three elements at a time, where an element is a word or a bracketed pair of elements.

1 Start with the rightmost three elements, X-Y-Z.

...	V	W	X	Y	Z

2a If X-Y-Z is confidently right-branching (see section 5.2), bracket it X-(YZ) and restart the algorithm with the rightmost three elements W-X-(YZ).

...	V	W	X	(YZ)

2b If X-Y-Z is confidently left-branching, move the window one element to the left and repeat the algorithm with W-X-Y. Note that X-Y-Z being confidently left-branching does not necessarily mean that it can be bracketed ...(XY)-Z, since left-branching may also be bracketed ...X)Y)-Z.

...	V	W	X	Y	Z

3 When the leftmost element in the whole sequence appears in the window, a left-branching triple U-V-W can be left-bracketed (UV)-W; restart with the three-element window expanded back to the right of the sequence.

(UV)	W	X	Y	…

5.2 Confidence in Branching Decisions

This section lists the conditions for deciding whether a given triple X-Y-Z is confidently right-branching or confidently left-branching.

The triple noun-adjective-noun is confidently right-branching since adjectives precede the nouns they modify. The exception is postpositive adjectives, which occur relatively infrequently within premodifier sequences.[2]

For any other sequence of three elements X-Y-Z, the bracketer reduces X, Y and Z to their local heads X_h, Y_h and Z_h. The sequence X-Y-Z is considered confidently right-branching if the frequency of previous occurrences of the reduced subbracketing (X_hZ_h) is greater than the frequency of previous occurrences of the reduced subbracketing (X_hY_h) times a *threshold* value.[3] If (X_hY_h) has occurred more frequently than (X_hZ_h), X-Y-Z is confidently left-branching.

In the absence of such evidence of confidence, the algorithm consults the user, in keeping with the semi-automatic approach of HAIKU. A fully automatic solution could just assume left-branching, based on results reported by Resnik [18] and Lauer & Dras [13] that left-branching is roughly twice as common as right-branching. However, these results were based on observations of noun-noun-noun triples only. For longer compounds and compounds with adjectives the ratio of left-branching to right-branching compounds might differ (see section 6.3).

5.3 What's Wrong with Adjacency?

Section 2.3 described two different models found in systems for bracketing compounds: the adjacency model and the dependency model. The algorithm presented above uses the dependency model, which compares frequencies of X-Y to X-Z when bracketing the triple X-Y-Z, ignoring previous occurrences of Y-Z.

For phrase (16) both left and right bracketings are possible. Previous occurrences of *small loan* would be evidence for right bracketing. Occurrences of *small business* would be evidence for left bracketing. Occurrences of *business loan* could be evidence for either bracketing. (17) restricts the loan; (18) restricts the business. But both refer to business loans. Therefore, occurrences of Y-Z do not count as evidence in favour of either bracketing.

(16) small business loan
(17) (small (business loan))
(18) ((small business) loan)

---

[2] Examples such as *((machine **readable**) dictionary)* must be bracketed using HAIKU's manual bracketer. An NMR can then be assigned between *machine readable* and *dictionary*, though the semantic relationship between *machine* and *readable* cannot be handled by HAIKU. Ferris [7] gives a comprehensive account of adjective syntax and semantics (including a classification of several types of postpositive adjectives) that may suggest avenues for semantic analysis of composite adjectives in HAIKU.

[3] The threshold can be set to any value by the user. A higher value means that the frequency of (X_hZ_h) must greatly exceed the frequency of (X_hY_h), or vice-versa. Section 6 examines the effect of different threshold values on the performance of the bracketer.

5.4 User Interaction

When the system cannot find sufficient evidence in favour of right-branching or left-branching, it turns to the user to supply the decision. Such decisions may be difficult and unintuitive. There are several ways to lessen the burden.

- ask only yes-no questions about right-branching — don't ask the user to supply bracketing information directly.

good: in the context of *tomato soup pot*,
does *tomato soup* make sense?

bad: does *tomato soup pot* bracket left or right?

- phrase questions in the context of three individual words by using subphrase reductions.

good: in the context of *steel soup pot*,
does *steel soup* make sense?

bad: in the context of *cotton soup pot cover holder*,
does *cotton soup pot cover* make sense?

- ask the user only about the acceptability of X-Y; do not ask the user to compare X-Y and X-Z — it is possible for *both* X-Y and X-Z to be unacceptable if X-Y-Z is in the middle of a modifier sequence, which would make the user's decision much more difficult.

good: in the context of *steel tomato soup*,
does *steel tomato* make sense?

bad: in the context of *steel tomato soup*,
which makes more sense: *steel tomato* or *steel soup*?

By using these techniques, a 'yes' answer to any question will provide confident left-branching; a 'no' answer will mean confident right-branching.

5.5 An Example

Assume that phrases (19) and (20) have already been bracketed. This section traces through the bracketing of (21).

(19) (soup bowl)
(20) (wooden (pot handle))
(21) wooden French onion soup bowl handle

Start with the rightmost three elements, soup-bowl-handle.

wooden	French	onion	soup	bowl	handle

soup-bowl-handle is confidently left-branching, since *(soup bowl)* has occurred and *(soup handle)* has not. Move the window one element to the left and restart the algorithm.

wooden	French	onion	soup	bowl	handle

Neither *(onion soup)* nor *(onion bowl)* have occurred previously and *soup* is not an adjective, so there is no confidence in right-branching or left-branching. Ask the user if *onion soup* makes sense in the context of *onion soup bowl*. The user answers 'yes', providing confidence in left-branching. Move the window one element to the left and restart the algorithm.

wooden	French	onion	soup	bowl	handle

Neither *(French onion)* nor *(French soup)* have occurred previously. Ask the user if *French onion* makes sense in the context of *French onion soup*. The user answers 'no', so the sequence is confidently right-branching. Bracket *(onion soup)* and expand the window one element to the left.

wooden	French	(onion soup)	bowl	handle

French is an adjective, so *wooden-French-(onion soup)* is confidently right-branching. Bracket *(French (onion soup))*. Since there are no more elements to the left of *wooden*, expand the window back to the right.

wooden	(French (onion soup))	bowl	handle

Neither *(wooden bowl)* nor *(wooden soup)*, which is the reduction of *wooden-(French (onion soup))*, have occurred previously. *(French (onion soup))* is obviously not an adjective, so there is no confidence in either right-branching or left-branching. Ask the user if *wooden soup* makes sense in the context of *wooden soup bowl*, which is the reduction of *wooden-(French (onion soup))-bowl*. The user answers 'no', providing confidence in right-branching. Bracket *wooden-(French (onion soup))-bowl* as *wooden-((French (onion soup)) bowl)*. Since there are no more elements to the left of *wooden*, expand the window to the right.

wooden	((French (onion soup)) bowl)	handle

(wooden bowl), which is the reduction of *wooden-((French (onion soup)) bowl)* has not occurred previously; *(wooden handle)* has occurred previously as a reduction of *(wooden (pot handle))*. *wooden-((French (onion soup)) bowl)-handle* is therefore confidently right-branching. Bracket *(((French (onion soup)) bowl) handle)*.

wooden	(((French (onion soup)) bowl) handle)

Since there are only two remaining elements in the sequence, bracketing is trivial:

(wooden (((French (onion soup)) bowl) handle))

Finally, the system stores all of the reduced subbracketings (22) for future processing. It also stores the complete bracketing (23): if the phrase is ever encountered in its entirety, the bracketer can simply look up the complete bracketing instead of going through the steps of the algorithm.

(22) *(onion soup)*
(French soup)
(soup bowl)
(bowl handle)
(wooden handle)
(23) *(wooden (((French (onion soup)) bowl) handle))*

6 Evaluation

This section gives results of using the bracketer in the context of two experiments. The *sparc* experiment applied the NMR analyzer (including the bracketer) to the first 500 non-trivial noun phrases in a computer installation guide. In this context a non-trivial noun phrase has at least one premodifier (adjective or noun) or postmodifying prepositional phrase. Bracketing the 500 noun phrases in the test produced 645 bracketed pairs. Each of the bracketed pairs was assigned a single NMR from the list in Table 1.

The second experiment was a complete analysis of a book on the mechanics of small engines, including semantic analysis by HAIKU at the three levels described in section 1. Bracketing resulted in 733 premodifier-head pairs. Barker *et al.* [4] describe the *small engines* experiment and its results.

6.1 System Performance

In both experiments, most of the modifier-head pairs occurred in noun phrases with a single premodifier and head. These simple compounds required no bracketing decisions. In the *sparc* experiment, the 645 pairs required 188 bracketing decisions. The system made 122 (65%) of these decisions correctly, with the rest made by the user as described in section 5.3. Of the 66 user decisions, 47 (71%) were required during the first half of the experiment with only 19 in the second half. The running totals of user and system bracketing decisions appear in Figure 2.

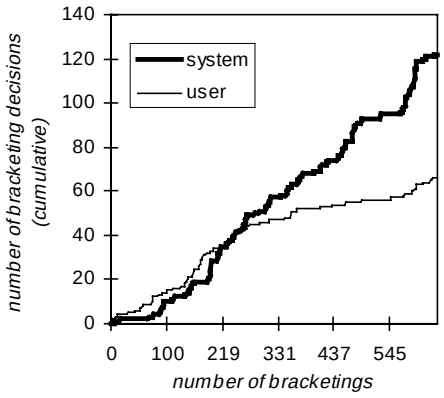

Figure 2. Bracketing decisions for *sparc*

The *small engines* experiment required 164 bracketing decisions. The system made 101 (62%) decisions correctly, with the rest made by the user. Due to the consistent terminology in the *small engines* text the cumulative number of decisions made automatically by the bracketer was always greater than the number required from the user.[4] The running totals for user and system bracketing decisions appear in Figure 3.

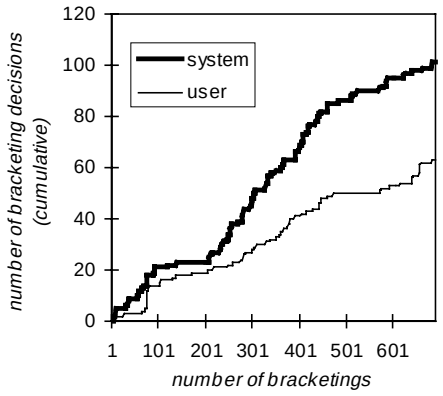

Figure 3. Bracketing decisions for *small engines*

6.2 The Effect of the Threshold

The bracketer determines whether a given triple X-Y-Z is confidently right-branching if the subbracketing (X_hZ_h) has previously occurred N times more frequently than the subbracketing (X_hY_h), where N is a threshold that can be set by the user. In the absence of sufficient evidence, the system asks the user to supply right-branching information.

If the value of the threshold is set high, the number of previous occurrences of (X_hZ_h) must greatly outweigh the number of occurrences of (X_hY_h) for the system to assume right-branching. High values of the threshold cause the system to be more conservative. Low values of the threshold (close to 1.0), make it more aggressive: the system requires less evidence to commit to a branching decision.

The *sparc* experiment was run twelve times with different threshold values. As expected, the number of system decisions, both correct and incorrect, was highest for low threshold values (see Figure 4). For higher threshold values, the number of incorrect system decisions decreased, but so did the number of correct decisions, and the user made more decisions.

For the *small engines* text, changing the threshold had no effect. This result suggests that for any triple X-Y-Z in that text, if (X_hZ_h) appears as a reduced pair, (X_hY_h) does not. In general, however, both may appear in any given text.

---

[4] The user made the decision for the very first bracketing in the text. After the second bracketing, the cumulative number of system decisions overtook the number of user decisions.

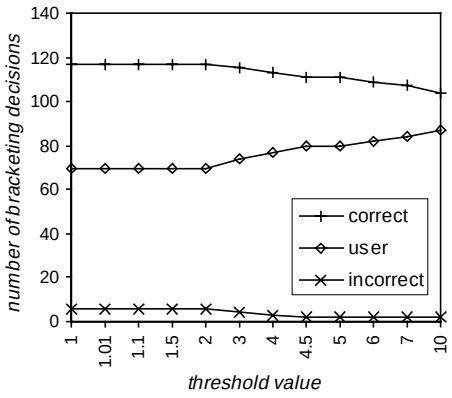

Figure 4. Effects of different threshold values on bracketing decisions for *sparc*

6.3 Branching Frequencies

Section 5.2 suggested that the bracketer could guess left-branching when there is no confidence in right-branching. Such a guess would be justified based on the predominance of left-branching compounds as reported by Resnik [18] and Lauer & Dras [13]. Results from the *small engines* experiment confirm the bias for left-branching. For the *sparc* experiment, however, the data in Table 2 show that guessing left-branching would have produced poor results.

		left-branching	*right-branching*
small engines	*noun-noun-noun*	47 (96%)	2 (4%)
	adjective-noun-noun	31 (84%)	6 (16%)
	total	78 (91%)	8 (9%)
sparc	*noun-noun-noun*	41 (55%)	33 (45%)
	adjective-noun-noun	11 (26%)	31 (74%)
	total	52 (45%)	64 (55%)

Table 2: Branching frequencies for *small engines* and *sparc*

The predominance of left-branching compounds is apparently not universal. If the system were modified to guess left in the absence of other evidence, there are texts (like the *sparc* text) for which the bracketer would perform poorly.

7 Future Work

Whenever the system brackets a list of premodifiers, it stores the reduced subbracketings to help bracket subsequent noun phrases. The system should be extended to store pairs resulting from postmodifying prepositional phrases as well.

For example, for the postmodifying prepositional phrase in (24) the system should store the pair (25).

(24) a bowl for soup
(25) (soup bowl)

Storing these extra pairs will increase the likelihood of the system finding evidence when bracketing new noun phrases.

Since HAIKU is an interactive semantic analyzer, it proceeds sentence by sentence through a text from the beginning. It cannot look ahead in the text at sentences that have not yet been analyzed. However, there may be unambiguous occurrences of modifier-head pairs in subsequent sentences that could help bracket phrases in the input sentence. The system could be modified to scan ahead in a text to look for such unambiguous pairs, although doing so may be costly.

Finally, there were two results in section 6 that need to be validated by further bracketing experiments. Left-branching triples do not always outnumber right-branching triples in a text. Counting the number of each in future experiments will determine whether the *sparc* text is unique in its predominance of right-branching triples. If left-branching predominance is confirmed, the system could be modified to guess left in the absence of other evidence.

The second result was the insensitivity of the bracketer to the value of the threshold. It is possible that a single technical text is unlikely to have both right-branching and left-branching evidence for a given triple. Future experiments may confirm this result or, if conflicting evidence is common, find a suitable threshold value for most texts.

Acknowledgments

This research is supported by the Natural Sciences and Engineering Research Council of Canada. I would like to thank Stan Szpakowicz for valuable input to the development of the ideas in this paper and to the paper itself. I am also grateful to Sylvain Delisle for comments on an earlier draft of the paper. The insightful comments of an anonymous reviewer helped shape the final version of this paper.

References

1. Barker, Ken (1997). “Noun Modifier Relationship Analysis in the TANKA System.” TR-97-02, Department of Computer Science, University of Ottawa.
2. Barker, Ken & Stan Szpakowicz (1995). “Interactive Semantic analysis of Clause-Level Relationships.” *Proceedings of the Second Conference of the Pacific Association for Computational Linguistics*, Brisbane, 22-30.
3. Barker, Ken, Terry Copeck, Sylvain Delisle & Stan Szpakowicz (1997). “Systematic Construction of a Versatile Case System.” *Journal of Natural Language Engineering* 3(4), December, 1997.
4. Barker, Ken, Sylvain Delisle & Stan Szpakowicz (1998). “Test-driving TANKA: Evaluating a Semi-Automatic System of Text Analysis for Knowledge Acquisition.” *Proceedings of the Twelfth Canadian Conference on Artificial Intelligence* (this volume), Vancouver.

5. Delisle, Sylvain (1994). “Text processing without A-Priori Domain Knowledge: Semi-Automatic Linguistic analysis for Incremental Knowledge Acquisition.” Ph.D. thesis, TR-94-02, Department of Computer Science, University of Ottawa.
6. Delisle, Sylvain, Ken Barker, Terry Copeck & Stan Szpakowicz (1996). “Interactive Semantic analysis of Technical Texts.” *Computational Intelligence* 12(2), May, 1996, 273-306.
7. Ferris, Connor (1993). *The Meaning of Syntax: A Study of Adjectives of English*. London: Longman.
8. Finin, Timothy W. (1986). “Constraining the Interpretation of Nominal Compounds in a Limited Context.” in [10: 163-173].
9. George, Steffi (1987). *On “Nominal Non-Predicating” Adjectives in English*. Frankfurt am Main: Peter Lang.
10. Grishman, Ralph & Richard Kittredge, eds. (1986). *Analyzing Language in Restricted Domains: Sublanguage Description and Processing.* Hillsdale: Lawrence Erlbaum.
11. Jensen, Karen, George E. Heidorn & Stephen D Richardson, eds. (1993). *Natural Language Processing: The PLNLP Approach*. Boston: Kluwer Academic Publishers.
12. Lauer, Mark (1995). “Corpus Statistics Meet the Noun Compound: Some Empirical Results.” *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics*. Cambridge. 47-54.
13. Lauer, Mark & Mark Dras (1994). “A Probabilistic Model of Compound Nouns.” *Proceedings of the 7th Australian Joint Conference on Artificial Intelligence*. Armidale. 474-481.
14. Leonard, Rosemary (1984). *The Interpretation of English Noun Sequences on the Computer*. Amsterdam: North-Holland.
15. Levi, Judith N. (1978). *The Syntax and Semantics of Complex Nominals*. New York: Academic Press.
16. Liberman, Mark & Richard Sproat (1992). “Stress and Structure of Modified Noun Phrases.” *Lexical Matters* (*CSLI Lecture Notes*, 24). Stanford: Center for the Study of Language and Information.
17. Pustejovsky, James, S. Bergler & P. Anick (1993). “Lexical Semantic Techniques for Corpus Analysis.” *Computational Linguistics* 19(2). 331-358.
18. Resnik, Philip Stuart (1993). “Selection and Information: A Class-Based Approach to Lexical Relationships.” Ph.D. thesis, IRCS Report 93-42, University of Pennsylvania.
19. ter Stal, Wilco (1996). “Automated Interpretation of Nominal Compounds in a Technical Domain.” Ph.D. thesis, University of Twente, The Netherlands.
20. Vanderwende, Lucy (1993). “SENS: The System for Evaluating Noun Sequences.” in [11: 161-173].
21. Warren, Beatrice (1978). *Semantic Patterns of Noun-Noun Compounds*. Göteborg: Acta Universitatis Gothoburgensis.
22. Warren, Beatrice (1984). *Classifying Adjectives*. Göteborg: Acta Universitatis Gothoburgensis.