File No. UIUCDCS-F-85-936

85-7

Word Sense Disambiguation
Using Constraints and Supports

Jeffrey M. Becker

Department of Computer Science
University of Illinois
at Urbana-Champaign

February 1985

ISG 85-6

This work was supported in part by the National Science Foundation under grant NSF DCR 84-06801, and the Office of Naval Research under grant ONR N00014-82-K-0186.

Acknowledgements

This work was supported in part by the National Science Foundation under grant NSF DCR 84-06801, and the Office of Naval Research under grant ONR N00014-82-K-0186. Thanks go to the University of Illinois Department of Computer Science for providing the excellent facilities which made this work possible, and Professor R. S. Michalski for encouraging me to publish it.

Table of Contents

Word Sense Disambiguation

Using Constraints and Supports

Abstract

Two types of information are useful in word sense disambiguation: constraints and supports. Constraints are used to eliminate candidates from consideration, and supports are used to select the most promising candidate from those remaining. The sources of constraints and supports in parsing are discussed, along with parsing algorithms which uses this information. A program which implements some of these ideas for an operator grammar is also described.

1. Introduction

Robust natural language processing requires that systems know a great deal about the topic under discussion. This is true for any intelligent system, be it machine or human. One would not normally expect an expert in solid state electronics to be able to talk intelligently to an expert in plant pathology about plant diseases, unless he also has considerable experience in this field as well. Not only are there words unique to each field, but concepts associated with these words which must be learned. The most successful computer-based systems for natural language processing have incorporated considerable domain knowledge. Examples are Winograd's SHRDLU system which operated in a "blocks world", and DeJong's FRUMP system for paraphrasing news stories.

My goal was to develop a parser for simple operator grammars using ideas from work in spreading activation/lateral inhibition networks. Such a parser would be capable of translating unparenthesized expressions of various types into Lisp-like code, given a suitable grammar and other background knowledge. For the most part, the goal has been achieved. The parser is totally data driven so it is extremely flexible. The parser is currently being used to parse rules used by a machine learning system. It is capable of using the rules for parsing logical, arithmetic, and English-like expressions *all at the same time* without confusion. Support for bracketed expressions, which could be used for operators accepting a variable number of arguments, is not fully implemented. Also, I found it necessary to ignore concerns

held by proponents of spreading activation models of cognition about creating dynamic structures during parsing.

2. Constraints and Supports

It should come as no surprise that word sense disambiguation in natural language and other complex languages requires knowledge in addition to that available in the syntax of a language. This information is available in the form of constraints and supports. The syntax of the language provides an initial set of constraints on the surface structure of a sentence. We also know that to parse a sentence, each word must be assigned one and only one sense per use. Other constraints are available if we use, for example, knowledge about what types of objects may serve as the subject or object of a verb, or as the target of an adjective or adverb. Supports are provided mainly from the context of the linguistic act in question, but also may be provided by other real world knowledge. In a sentence where lexical ambiguity cannot be resolved by local constraints, supports can often be used to select the appropriate word sense.

Consider the following example sentences :

1. "The astronomer married a star."
2. "John shot two bucks."

In the first sentence, if we know that both the agent and object of "married" must be human beings (and in addition, should be of legal age and opposite gender) we must conclude that "star" means a famous person and not a celestial body. No additional information is needed. In the second sentence, both the verb and the object of the verb have multiple senses. If the context is "hunting" or "woods", we would conclude that John is a hunter and has fired a gun at two male deer, presumably killing them. If the context is "money" or "gambling", we would conclude that John wasted some money. Note that if we know something about the habits of the person named John, for example that John is either a gambler or a hunter, then we could choose a parse for the sentence with no other contextual information. Supports do not eliminate possibilities, but rather justify a choice of word meanings which fit the current context the best.

The often cited sentence given by Chomsky is a useful example for demonstrating the number of ways in which semantic constraints are used in parsing:

3. "Colorless green ideas sleep furiously."

Though syntactically correct, this sentence contains a large number of semantic constraint violations. For example, the adjective "green" should have a target which is a physical object. The adjective "colorless" disagrees with the adjective "green" since both have the same target. The subject of sleep normally should be an animal of some kind. The adverb "furiously" is inappropriate if one understands sleep as a time of rest.

3. Parsing with Spreading Activation Nets

Spreading-Activation Lateral-Inhibition Nets (SAN's) have been proposed as a method for combining multiple knowledge sources in a highly parallel way for performing the parsing task. It has been suggested that SAN's are useful in predicting a number of aspects of human performance in parsing [Waltz and Pollack, 1984]. For example, it has been proposed that the reason the first example above seems *funny* is that the reference to "astronomer" primes the *wrong* meaning for "star". However, building a real parser using SAN techniques is not a straightforward task.

Spreading activation/lateral inhibition cannot be used until the appropriate connecting structures are built. We can assume that a large proportion of the network exists in memory as priming relations between concepts or between sensory patterns and concepts. There would also need to be knowledge about mutually exclusive categories (alive/dead, hot/cold, etc.), constraints for fillers in syntactic structures, such as noun phrases, and constraints on fillers for subjects/objects for verbs and targets for adjectives and adverbs. Waltz and Pollack argue that for syntactic components, such as noun phrases, there must either be a supply of multiple copies of noun phrase recognizers built into the system, or a single noun phrase recognizer which gets reused. The second alternative can be rejected because it would make it impossible to, for example, set up a mutual inhibition link between S (sentence) nodes of competing parses for a sentence (see Figure 5b. [Waltz and Pollack, 84]). The first alternative is no better than what I propose below since it would require an allocation mechanism and the ability to

dynamically create activation or suppression links between node instances. I feel that a third alternative should be considered, that an instance of a noun phrase recognizer (or any other node) can be generated dynamically and used in the parsing process independently of other noun phrase recognizers.

Consider the sentence :

4. "Bill paid the bill."

If we assume that there are permanent mutual suppression links between the word senses of "bill" then it would be difficult to explain how we could understand that the first "Bill" is a person and the second "bill" is a notice of a payment owed, pretty much all at the same time. If we assume that mutual suppression links are dynamically created between some kind of instantiated forms of the word senses for "bill", for each occurrence of "bill" in the sentence, there is no problem.

I feel that the notion of dynamically creating parts of a SAN/constraint network is not only plausible but can be supported if one considers the flexibility and adaptability of the human information processing mechanism, and the requirements of the parsing/understanding task. If, for example, the brain can generate the scenes which are seen in dreams, why should it not be capable of generating a simple network of relations for parsing? The fact that humans have a memory proves that some form of knowledge structure creation occurs. The relationship between SAN's and neural "hardware" has probably been overemphasized, except perhaps where permanent memory is involved for word sense and contextual priming. Too little is known about the brain's "software" organization to make *strong* claims about what it is not capable of (e.g. whether "pointers" should be considered plausible).

4. A Parsing Program

The parser I have implemented is basically a chart parser [Winograd, 1983] which uses a spreading activation net for final disambiguation by applying contextual information when local constraints are inadequate. I wanted to be able to parse operator grammars (Lisp or Prolog-like) as well a certain natural language forms. I also wanted to be able to do so with reasonable efficiency, on available computers, which tended reduce the appeal of spreading activation. The parser activates instances of word-sense nodes and operator nodes from the symbols present in the given input string. Only nodes

containing a key for which there is a literal in the input string are instantiated. These nodes are linked up under constraints given in declarations provided by the user. The order of words is significant. Subtrees which do not account for all nodes in the input string are discarded, and if more than one root node remains, spreading activation is applied. For more general natural language parsing, it would be necessary to provide for multiple levels the kind of activation/instantiation described here.

Figure 1 shows a chart for an ambiguous sentence. The input sentence is shown at the bottom with each word linked to a chart block (the small squares) on each side. For clarity, only the parent pointers are shown in the parse trees. There are two additional pointers from each node in a parse tree, one to the chart block to the left of the leftmost child from the input sentence words, and one to the chart block to the right of the rightmost child, which are not shown. The crossed out nodes are nodes which were activated but could not be linked to a viable parse tree. Note that the "two" node was

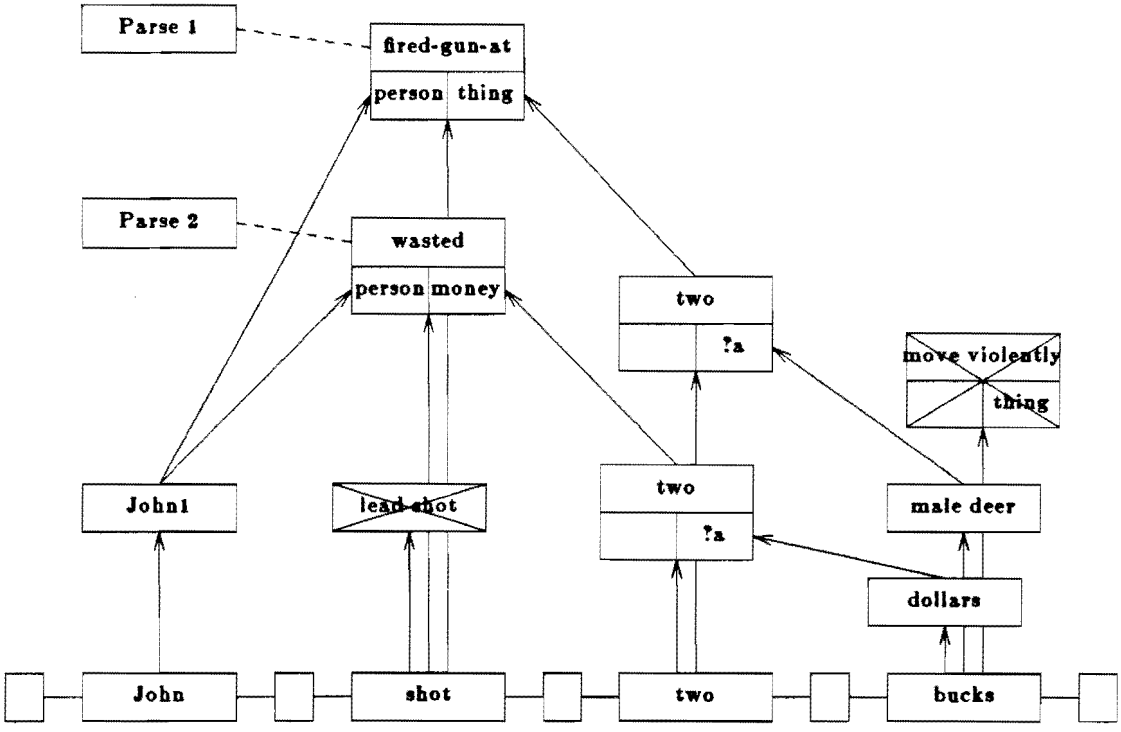

Figure 1. Chart structure for an ambiguous sentence

duplicated to accomodate multiple fillers.

Spreading activation techniques are not used in the initial phase because they are simply not needed. The combination of bottom-up instantiation of nodes, with linking under the top-down constraints present in the nodes is sufficient for dealing with all ambiguities that can be handled by local constraints. With a little imagination, one can view the chart parsing phase as a form of "predictive" spreading activation. Initial activation and elimination of candidates during linking under hard constraints accomplishes pretty much what the active part of a spreading activation net would accomplish under the same circumstances, assuming a certain degree of dynamic structure creation is allowed.

Spreading activation is used when using contextual information for final disambiguation, but it is really overkill since some simple summing of weights would be adequate for most practical problems. The parse trees are linked to form a SAN by joining parents and children nodes with mutual activation links, joining competing root nodes with mutual inhibition links, and joining competing parent nodes with mutual inhibition links. Also, context nodes are joined by mutual activation links to the appropriate parse tree nodes. Currently, activation links have a weight of 0.2 and inhibition links have a weight of -0.6. Link weights are used to scale the activation energy transmitted from a node. All node energies stay between 0 and 1. Nodes in the input sentence and context are given an initial energy level of 1.0, and all other nodes are given an initial energy of 0.5. Energy transmission to a node is scaled so that when a node energy is at 0.5 it is very easy to change it, but as it approaches 0 or 1 it becomes harder to change, thus node energies are maintained in the required range. Relaxation is terminated when either a fixed number of cycles have been run, or when the total activation energy flowing through the net drops below a predetermined threshold.

The data-base associated with the parser consists of three sections. First, there is a set of operator and word-sense node declarations. Second, there is a hierarchy of type classes. Third, there is a set of activation associations used with contexts. For each operator there is a result type and precedence, and for each argument there is a type constraint and precedence or relative precedence given. There is also a surface form specification, from which activation keys are extracted. A word-sense node is intended

for linking a literal to a particular concept. Word-sense nodes are much like operator nodes, except that the surface form usually consists of a single literal. A type is specified by a lisp atom. A type hierarchy is formed in the obvious way with 'isa properties. The type hierarchy is searched when matching slots in a node to fillers in the chart. A context, for the time being, is a list of atoms. Context associations with particular concepts are stored under the "activate" property of an atom representing a context.

The program is written in Franz Lisp [Foderaro, 1983]. I used a large macro package which I developed in part for this programming project [Becker, 1985]. The macro forms should be self-explanatory, and include iterative forms, generalized assignment, simple structure declarations, printout facilities, and miscellaneous other useful items.

5. A Sample Grammar

A brief introduction to the grammar declaration forms used in the system and a sample grammar are given below. Note that a high precedence means that the operators will bind more tightly than those with a low precedence. Also note that variables may be used in place of type names to indicate that corresponding types are required.

The forms used for declarations are:

Defining an operator:

(Defop '(operator-name (arg1)(arg2) ... : type precedence) '(surface-form))

where -

the operator-name is a lisp atom,
each arg has the form - (arg-symbol(s) : arg-type arg-precedence) where
the arg-type constrains the type of the argument and the arg-precedence
is given relative to that of the operator,
the type is an atom or variable defining the operator type,
the precedence is an integer from 0 .. 1000,
and the surface-form is a list of atoms.

Defining a concept:

(Defword 'concept-name 'type '(surface-form))

where -

the concept-name is a lisp atom,
the type is an atom defining a type for the concept,
and the surface form is a list of atoms.

Defining a type hierarchy:

(isa 'type 'supertype)

Defining a context:

(Activate 'context-name '(list of associated operators and word senses))

The following is a mixture of declarations which provide the system with knowledge for parsing expressions involving basic family relations, arithmetic expressions, astronomers getting married, paying bills, and shooting bucks.

```
; family relations

(Defop '(= (x y : ?a >) : boolean 600)          '(x is y))

(Defop '(father  (x : person >=) : person 800)    '(the father of x))
(Defop '(mother  (x : person >=) : person 800)    '(the mother of x))
(Defop '(sister  (x : person >=) : person 800)    '(the sister of x))
(Defop '(brother (x : person >=) : person 800)    '(the brother of x))

(Defword 'John1  'person  '(John))
(Defword 'Bill1  'person  '(Bill))
(Defword 'Mary1  'person  '(Mary))
(Defword 'Sam1   'person  '(Sam))
(Defword 'Ed1    'person  '(Ed))
(Defword 'Sally1 'person  '(Sally))
(Defword 'a      'article '(a))
(Defword 'the    'article '(the))


; arithmetic

(Defop '(- (x : number >)    : number 900)     '(- x))
(Defop '(* (x y : number >=) : number 800)     '(x * y))
(Defop '(/ (x y : number >=) : number 800)     '(x / y))
(Defop '(+ (x y : number >=) : number 700)     '(x + y))
(Defop '(- (x y : number >=) : number 700)     '(x - y))
```

```
(Defop '(= (x y : ?a >)      : boolean 600)    '(x = y))


; the astronomer married a star

(Defop '(married (x y : person >) : true 700) '(x married y))

(Defword 'astronomer1 'scientist '(the astronomer))
(Defword 'star-person 'famous-person '(a star))
(Defword 'star-celestial 'celestial-body '(a star))

(isa 'true 'boolean)
(isa 'false 'boolean)
(isa 'scientist 'person)
(isa 'famous-person 'person)
(isa 'person 'animate-object)
(isa 'animal 'animate-object)
(isa 'money 'inanimate-object)
(isa 'celestial-body 'inanimate-object)
(isa 'animate-object 'thing)
(isa 'inanimate-object 'thing)


; paying bills

(Defop '(paid (x : person >) (y : money >) : true 700) '(x paid y))
(Defword 'bill 'money '(the bill))

; John shot two bucks

(Defop '(move-violently (x : thing >) : true 700)  '(x bucks))
(Defop '(two (x : ?a >) : ?a 800)  '(two x))
(Defop '(fired-gun-at (x : person >)(y : thing >) : true 700) '(x shot y))
(Defop '(wasted (x : person >)(y : money >) : true 700)  '(x shot y))

(Defword 'male-deer 'animal '(bucks))
(Defword 'dollars 'money '(bucks))
(Defword 'lead-shot 'inanimate-object '(shot))

; some simple contexts

(Activate 'hunting '(fired-gun-at male-deer lead-shot woods))
(Activate 'gambling '(money wasted win casino))

(setq *context* '(hunting))
```

6. Parsing Examples

This is script of sample parses using the grammar shown above. The raw script has been edited and annotated to improve readability, and explain what is going on.

```
% lisp
Franz Lisp, Opus 38.79

-> (load 'parse)              ; load the parser
[fasl parse.o]
t
-> (load 'g2)                 ; load the grammar given above
[load g2.l]
t
-> (parse '(3 + 4 - 5 * 6))   ; first a simple arithmetic expression

; The ActiveList is a list of initially active nodes.
; Here plus, subtract, unary minus and multiplication nodes are activated.
; Node printout format is  {node-name : node-class : list-of-children}

ActiveList:
*              : operator    : nil, *, nil     ; a multiply node
-              : operator    :  -, nil         ; an unary minus node
-              : operator    : nil, -, nil     ; a subtraction node
+              : operator    : nil, +, nil     ; an addition node

; A node can become active several times during the filling in process,
; and is duplicated when multiple fillers for a slot are available.
; Every combination not prohibited by local constraints is tried.

(- (+ 3 4) (* 5 6))   ; The resulting parse in Cambridge-prefix form.


-> (parse '(- 3 * 5 / 9))

(/ (* (- 3) 5) 9)     ; Unary minus is handled ok.


-> (parse '(John is the brother of Mary))

; Only operators and word senses associated with the given input are
; considered.  Thus it is possible to mix different types of grammars
; in the data available to the parser.  Note the nodes activated -

ActiveList:
Mary1          : wordsense   : Mary
brother        : operator    : nil, brother, nil, nil
the            : wordsense   : the
```

```
=           : operator     : nil, is, nil
John1       : wordsense    : John


(= John1 (brother Mary1))   ; The parse looks something like predicate logic


; Only local constraints are needed to parse this ...

-> (parse '(the astronomer married a star))

ActiveList:
star-person     : wordsense    : nil, star          ; a star as a famous person
a               : wordsense    : a
star-celestial  : wordsense    : a, nil             ; a star as a celestial body
married         : operator     : nil, married, nil
astronomer1     : wordsense    : nil, astronomer
the             : wordsense    : the


; And the parse is ...

(married astronomer1 star-person)       ; the astronomer married a famous person


; The astronomer can get married to somebody else if he wants...

-> (parse '(the astronomer married the sister of Bill))

(married astronomer1 (sister Bill1))


; Bill pays his bills.

-> (parse '(Bill paid the bill))

(paid Bill1 bill)


; Contextual information will be needed to resolve the ambiguity here.

-> (parse '(John shot two bucks))

ActiveList:
move-violently  : operator     : nil, bucks         ; meaning 1 for bucks
male-deer       : wordsense    : bucks              ; meaning 2 for bucks
dollars         : wordsense    : bucks              ; meaning 3 for bucks
two             : operator     : two, nil
fired-gun-at    : operator     : nil, shot, nil     ; meaning 1 for shot
wasted          : operator     : nil, shot, nil     ; meaning 2 for shot
```

```
lead-shot       : wordsense   : shot          ; meaning 3 for shot
John1           : wordsense   : John
```

; Contextual disambiguation must be used, note the context.

Trying to disambiguate using SAN in context (hunting) :

; These are the possible parses arrived at. The second choice is ok
; since money is defined as an inanimate object, and a gun can be fired
; at any "thing".

```
(fired-gun-at John1 (two male-deer))
(fired-gun-at John1 (two dollars))
(wasted John1 (two dollars))
```

; These are the weights arrived at

```
0.785   (fired-gun-at John1 (two male-deer))
0.702   (fired-gun-at John1 (two dollars))
0.428   (wasted John1 (two dollars))
```

(fired-gun-at John1 (two male-deer)) ; The winner is ... a good choice.

; Now we will change the context ....

```
-> (setq *context* '(gambling))
(gambling)
```

-> (parse '(John shot two bucks))

<skip tracing stuff, same as before ....>

Trying to disambiguate using SAN in context (gambling) :

```
(fired-gun-at John1 (two male-deer))
(fired-gun-at John1 (two dollars))
(wasted John1 (two dollars))
```

```
0.633   (fired-gun-at John1 (two male-deer))
0.333   (fired-gun-at John1 (two dollars))
0.883   (wasted John1 (two dollars))
```

(wasted John1 (two dollars)) ; And again, the parse is appropriate!

; These two examples show that type correspondence can be checked for:

```
-> (parse '(John is (3 * 4)))

There is no parse for
(John is (3 * 4))


-> (parse '(3 * 4 = 2 + 5))

(= (* 3 4) (+ 2 5))

-> Goodbye
```

7. Summary

The use of constraints and supports in word sense disambiguation has been discussed and illustrated. Information in the form of constraints is important for reducing the number of alternatives which are considered during the parsing process. It has been suggested that the requirement that all linking structures be static in a knowledge-based parser (such as a spreading activation - lateral inhibition net) is too severe, and that a certain degree of dynamic organization should be considered allowable. A program which uses constraint and support information for word sense disambiguation during parsing has been described. The program's performance has been illustrated with a sample grammar and sample parsing results. The program has proven flexible, powerful, and efficient enough to be put to practical use for parsing rules containing arithmetic and logical expression used as input to a machine learning program.

References

Becker, Jeffrey M., "Macros and Utilities for FRANZ LISP", ISG Report, Department of Computer Science, University of Illinois, Urbana, Illinois, in preparation.

Foderaro, John K., Sklower, Keith L., and Layer, Kevin, *The FRANZ LISP Manual*, University of California, Berkeley, California, June, 1983.

Waltz, David L., and Pollack, Jordan B., "Massively Parallel Parsing: A Strongly Interactive Model of Natural Language Interpretation", Coordinated Science Lab, University of Illinois, Urbana, Illinois, 1984, in preparation.

Winograd, Terry, *Language as a Cognitive Process, Syntax*, Addison-Wesley Publishing Company, 1983.

BIBLIOGRAPHIC DATA SHEET	1. Report No. UIUCDCS-F-85-936	2.	3. Recipient's Accession No.
4. Title and Subtitle Word Sense Disambiguation Using Constraints and Supports			5. Report Date February 1985 6.
7. Author(s) Jeff Becker			8. Performing Organization Rept. No.
9. Performing Organization Name and Address Department of Computer Science University of Illinois Urbana, IL 61801			10. Project/Task/Work Unit No. 11. Contract/Grant No. NSF DCR 84-06801 ONR N00014-82-K-0186
12. Sponsoring Organization Name and Address National Science Foundation, Washington, DC Office of Naval Research, Arlington, VA			13. Type of Report & Period Covered 14.

15. Supplementary Notes

16. Abstracts

Two types of information are useful in word sense disambiguation: constraints and support. Constraints are used to eliminate candidates from consideration, and supports are used to select the most promising candidates from those remaining. The sources of constraints and supports in parsing are discussed, along with parsing algorithms which used this information. A program which implements some of these ideas for an operator grammar is described.

17. Key Words and Document Analysis. 17a. Descriptors

Natural Language Processing
Parsing
Lexical Ambiguity

17b. Identifiers/Open-Ended Terms

Chart Parser
Spreading Activation/Lateral Inhibition Network

17c. COSATI Field/Group

18. Availability Statement	19. Security Class (This Report) UNCLASSIFIED	21. No. of Pages 19
	20. Security Class (This Page) UNCLASSIFIED	22. Price