ED 261 395 CS 209 285

AUTHOR Dobrin, David N.
TITLE Limitations on the Use of Computers in Composition.
PUB DATE Mar 85
NOTE 35p.; Paper presented at the Annual Meeting of the Conference on College Composition and Communication (36th, Minneapolis, MN, March 21-23, 1985).
PUB TYPE Viewpoints (120) -- Speeches/Conference Papers (150)

EDRS PRICE MF01/PC02 Plus Postage.
DESCRIPTORS *Computer Assisted Instruction; *Computer Software; Elementary Secondary Education; Higher Education; *Word Processing; Writing Evaluation; Writing Improvement; *Writing Instruction
IDENTIFIERS *Software Evaluation; *Text Editing

ABSTRACT

Noting that composition teachers would like computers in order to facilitate the mechanics of writing, analyze text, and correct problems, this paper argues that classroom computer applications are limited because computers cannot analyze text the way a reader would. The paper first posits that readers look at text in terms of semantics and content while computers look at text in terms of syntax and form. It then illustrates this point by comparing spelling check programs with text analysis programs. It describes the three analysis programs--DICTION and STYLE, by Bell Labs, and the more powerful EPISTLE by IBM--and how they function, and provides output from the first two programs using the document to illustrate that the analysis is more or less useless to the writer. The paper then discusses the possibilities that text analyzers can be made better, observing that the writer must be skilled in sorting through inaccurate output in order for the output to be of any use, and that those without such skills may be damaged as much as helped by such programs. The paper concludes with the observation that programs can be made more accurate, but that a more worthy goal would be better on-line editors, better systems for communication, and better word processors. (HTH)

Limitations on the Use of Computers in Composition

David N. Dobrin

Table of Contents

List of Figures

David N. Dobrin
Massachusetts Institute of Technology
Cambridge MA 02139

Limitations on the Use of Computers in Composition

As I understand the recent literature on computers and composition, the basic question addressed is the following:

1. What applications do computers have in the composition classroom?

The literature, indeed, has agreed on an answer: Today, word processing, communications, invention, text analysis, and idea processing; tomorrow, improvements on those things and maybe a lot more. I am not happy with this answer. The list of current applications is confused, for the useful applications and the useless are lumped together indiscriminately. (Some of these applications don't work and can't work; others work, but not in any way that is useful; still others work and are terrific.) The improvements and new applications being described (see some other papers in this volume) are far less promising than is generally made out.

I think it's time we started looking critically at existing applications and at promised future applications, asking whether they are really useful and if not, why not. If we are critical, we can save a good deal of time and money today, by not spending it on useless programs, and even more tomorrow, by not spending it on development of impossible or ill-conceived applications.

The problem, of course, is how to look critically at applications. There are many out there, and each does slightly different things. If we tried to evaluate individual programs--if I tried to argue my position by looking at individual examples of each program--we'd get lost. I propose that we can begin our criticism by developing criteria for what could, in principle, be useful applications. We can, in other words, ask the following question, a question which is logically prior to the first question.

2. What applications can computers have in the composition classroom?

The first question is a question about what it would be nice to have computers do; the second is a question about what it is possible to have computers do.

Obviously the second should have been the primary question all along, since if you answer the first without answering the second, you waste time making up nice applications which won't work. The second question, however, has not been raised in the literature. I think I know why. People writing about computers and composition have assumed that computers can do anything.

This assumption is not true. Computers work in a particular way, just as automobiles do. The way they work makes them good at some things and bad at others, just as cars are good at flying along freeways and bad at flying to the moon. Just as with cars, if one wants to use computers effectively, one has to be very clear about what one wants to do with them (what one's purposes are) and very clear about what can be done with them (what their capabilities are).

What are those purposes? In general, we (teachers of composition) want computers to do two slightly different things. We would like computers to make the mechanics of writing easier, by facilitating typing and formatting and by making it easier to get access to texts. The capability that we use when we turn computers to these purposes is the ability to manipulate text. This capability is used in word processors, formatters, and communications programs. We would also like computers to make the text itself better by analyzing the text and correcting problems in it. The capability used here is the ability to respond to or analyze text. This capability is used in text analysis programs (which I will look at closely in the remainder of this paper), invention aids, and idea processors. In this paper, I want to argue that we can use the computer effectively for the first purpose, but we can't for the second, primarily because computers are good at manipulating text, but bad at analyzing it. Since these capabilities won't change, new programs that make manipulation of text even easier are promising developments; programs that try to improve the text won't work out at all.

This is a strong claim, I admit, and it's a difficult one to make for the following reason. Essentially, I am saying that it is impossible for computers to do certain things. It is, however, difficult to prove that Q is impossible, because the fact that Q hasn't been done is not sufficient proof. People who do think Q is possible will claim that progress is being made toward the goal. (They will also, in my experience, make nasty asides about how skeptics laughed at the Wright brothers.) The way to argue for this claim, therefore, is to look at the progress that made. If progress is steady, and reaching the goal merely a matter of persistence, then it is reasonable to believe that the goal can be reached. If, on the other hand, progress is unsteady because researchers are constantly coming up against problems that aren't solvable (problems, that is, that there is no reason, in principle, to think can be solved) and if, in particular, researchers in many different areas are coming up against the same kind of problem, then the burden of proof falls on the researchers, not the skeptics.

Syntax and Semantics

There is a problem with no obvious principled solution, and this problem is impeding progress. It's a very serious problem because it is built in to the way computers work; there's no way around the problem unless you change computers. The problem is this. When computers try to analyze text--respond to it is the term I will use from now on--they don't do it the way we do.

When we look at a text, we respond to it on the basis of its meaning. (Semantics and content are more technical terms for the same thing that will be used periodically from now on.) When computers look at a text, they respond to it on the basis of its shape.[1] (Syntax and form are the equivalent technical terms.) This difference exists no matter what kind of text we are talking about.

Take, for instance, the lowly letter "A." To me, the letter is a meaningful object.[2] When I type the following symbol, "A," I am typing the letter "A." When the computer responds to my typing,

---

[1] Don't worry, what I mean by "shape" will get clearer. I take the word from Jerry Fodor. See "Methodological Solipsism Considered as a Research Strategy in Cognitive Science" in John Haugeland, ed. <u>Mind Design</u> (Cambridge: M.I.T. Press, 1981).

[2] If I were Chinese, on the other hand, it would not be meaningful; it would be just a bunch of squiggles.

by displaying the letter, it only displays the symbol "A." As far as the computer is concerned, it is merely commanding the terminal to display a bit pattern that corresponds to the ASCII code 101. You can see the difference if you consider what would happen if we changed the ASCII code so that 101 corresponded to some Chinese character. When I type the letter "A," the computer would then cheerfully display squiggle, rather than "A."[3] If the computer understood "A" as a letter, it would know it had made a mistake. As it is, the computer doesn't.

The ASCII code 101 is a label--a syntactic indicator--of the letter "A." The computer never responds to letters; all it ever does is respond to syntactic indicators, to shape, not content. Whenever the computer appears to be manipulating meaningful text, all it is actually doing is simulating the manipulation by manipulating syntactic indicators of the text. In the case of single letters, there is a one-to-one correspondence between syntactic indicators and meaningful objects, so simulation is quite easy and very exact. In other cases, the simulation is not quite so accurate. Responding to the form and responding to the content occasionally produce different results.

---

[3] Those of you in the know recognize this as an arcane version of John Searle's Chinese Room argument. See "Minds, Brains, and Programs," <u>The Brain and Behavioral Sciences</u> (1981:3), pp. 417-457.

Take, for instance, the simulation done by this word-processing program. If I type the letter "A," the computer reliably simulates my action by displaying the group of squiggles, "A." Consider, however, what happens when I do something slightly more complex: I ask the computer to delete this sentence backwards. The sentence, the meaningful object, begins with the word "Consider." The computer, however, only deletes backward to the colon before the symbol "I." The computer looks for the syntactic indicators of sentence beginnings, a period or colon followed by two spaces, not for sentence beginnings. (Reasonably enough, in the case of lists.) Whenever such a sequence does not begin a sentence, or whenever a sentence beginning doesn't have that sequence (I might mistype), the computer does not simulate accurately.

If I were a computer programmer, I could improve the simulation by finding more accurate syntactic indicators. I could, for instance, label each noun, verb, pronoun, and adverb in a putative sentence (making the parts of speech of the words recognizable by their label and thus making them into syntactic objects) and have the computer look for word sequences with free nouns and verbs preceded by a period and any number of spaces. This would be difficult and the end result would be slow, but the final product would be a more accurate simulation. Not perfectly accurate, of course, but pretty good.[4] Sentences and letters are, of course, much the best targets for a simulator because we already have syntactic conventions for labeling them. Objects which are, so to speak, more "meaningful"--less determined by syntactic constraints--are much harder to simulate, and simulations, therefore go wrong more often. This is the problem mentioned above. The more meaningful a text, the more a correct response to a text requires a response to the meaning of the text and the more difficult it is to find syntactic equivalents of the text which will allow a computer to simulate the correct response. This rule has several consequences, which I present without proof. First, while any particular text may in fact receive the correct simulated response, it is not possible to build a system which can be relied on to respond correctly every time. Second, the more difficult the simulation, the less reliable it is. Third, in any application where the simulation is unreliable, the user has to identify and correct the errors of the computer. Fourth, when the user has to judge the output before making use of it, most, if not all of the utility of the program is lost.

[4] By perfect, I mean only that it would respond as a reasonably well-informed person would. People make mistakes, too, but of a different, more reasonable kind. They are mistakes in responding to the meaning of a text.

This rule depends on a notion of what "meaningful object" and "requires a response to the meaning of the text," is, and without going into a lot of technical philosophy, those notions are difficult to explain precisely. For my purposes, however, all you need is an intuitive feeling for what the ideas mean, and that I can give you with the examples in the next section. There, I'll look at several different kinds of text analyzers, ranking them according to how much they need to respond to the meaning of text and showing how unreliability goes up with the difficulty of the task.

<u>Text Analyzers: Spelling, Style, and Grammar Checkers</u>

Programs which perform text analysis have been widely touted. They are, it's said, a way of relieving both writer and teacher of unnecessary, painstaking work. They provide, moreover, a way of teaching students to perform this work, since the checkers provide instant feedback on the students' actual papers.

These claims certainly seem to have some merit in the case of spelling checkers. Spelling checkers, it appears, find misspelled words for the teacher and writer, saving both the work of proofreading. Having a checker available makes it easier for bad spellers to learn how to spell, since they provide feedback; it can even help good spellers.

Unfortunately, the appearance is rosier than the reality. Spelling checkers aren't entirely accurate. The reason for the inaccuracy is that the spelling checkers don't work the way we do. When we check spelling, we look for meaningful objects: misspelled words. Spelling checkers, on the other hand, take each discrete symbol string in a text and compare it to a list of discrete symbol strings called a wordlist. If the string is on the wordlist, it is discarded; if it is not, it is sent to output, where we call it a misspelling. The difference in the way the two works produces inaccuracies. Words witch (e.g.) have been accidentally converted into other words are not caught. Neither are misspelled words that have managed to creep onto the wordlist. Correctly spelled words, on the other hand, that are not on the list are caught. Caught, too, are non-standard, but acceptable spellings ("gage" for "gauge") and non-conventional forms of spelling (s-p-e-l-l). The upshot: the list of unrecognized words doesn't correspond to the list of misspelled words.

"So what," you might say, "the list is still useful." Not as useful as you might think. Consider, for instance, Figure 1, which contains the list of unrecognized words from the second draft of the longer version of this paper. To put it mildly, there is a lot of garbage on this list. Most, fortunately, is obvious garbage. The formatting instructions (Topmargin, Pageheading, etc.), the

```
    Bloomian
  Bottommargin
    breadbox
     Fodor's
 fullpagefigure
     grungy
   leftmargin
    LineWidth
   natatorium
   Pageheading
 PrefaceSection
       Ref
     roblem
      rote
     Searle
    tabclear
    tabdivide
     teatime
    Topmargin
     tudent
```

Figure 1: Output of Spell Routine Run on an Earlier Draft of this Paper

proper names, and the obvious gaps in the word list (natatorium), fall into that category. Some, however, is less obvious. "Roblem" and "tudent" seem to be typos; "teatime" and "breadbox" might really be misspellings. Unfortunately, they're garbage too. "Roblem" and "tudent" are produced by vagaries in the UNIX operating system, not by me; "teatime" and "breadbox" are spelled correctly. There are, in fact, no misspelled words on the list. Does that mean that there are no misspelled words in my text? By no means.

So what have I gained by using the program? I've wasted some time running the program (which is, understandably, slow); I've wasted time hunting down apparent errors; I've learned nothing knew (oops); and I still have to proofread. This is, in my experience, typical, and for that reason, I've stopped using the program. Other program designs are somewhat easier to use--we have an interactive speller, now--but the problem remains the same and so does the result. I don't use it.

Let me pause for a moment and point out the structure of my response to the program. Because the program is not accurate, I need to adjust my reactions to the inaccuracy; the first thing I must do is judge the correctness of the program's output. Usually, that's pretty easy to do; but inevitably, there are some points where my knowledge is insufficient, and I can't tell whether it's right. I then have to figure out who is right. Because I understand spelling and I understand how the computer works, the effort involved is not too great. Still, I must expend this effort before I can begin to use the program. All in all, I find that using the sorting through the garbage is not worth the effort, especially since I can't be sure that the output includes all the possible mistakes.

A normal response to this description of my problem is the following. "Sure, you don't need a spell program. But what about the people who really can't spell?" Yes, they have more inclination to use the program, and in the long run, they may use it more. But they, too, have to sort through the garbage first. And for them, the sorting is considerably more difficult. For one thing, the list is longer. For another, they know less about spelling than I do, so there are more points where they must check on the program. Remember, they are less able to tell the difference between words that just aren't on the list and words that are misspelled. So for them, the spelling checkers are much more difficult to use.

For the bad speller, it is still worthwhile to expend the effort, but for all but the dedicated bad speller there are dangers. Here's one I've seen. The bad speller gets a list of doubtful bad words and rather than figure out when the computer is wrong, he changes every entry to some other word. He excises a word from his vocabulary rather than trying to figure out whether it is misspelled.

Why, despite the problems, is it worthwhile to use the spell routines? There are three reasons:

- They are relatively accurate, so the garbage ratio is fairly low.

- They can be used to find inadvertent errors. I often accidetnally (e.g.) misspell words.

- The rules they follow are strict. When a word is misspelled, it's misspelled; that's all there is to it.

You can see, though, that if any of these conditions are not met, then it makes much less sense to do the work required to use the program.

With other text analyzers, however, none of these conditions are true. The reason they are not true is that other text analyzers are looking for features of prose that don't have such simple simple syntactic indicators, features of prose that are more heavily determined by the meaning.

To show you what I mean, let me take a look at two text analysis programs that are widely considered to be the state of the art: STYLE and DICTION, two of the Writers Workbench programs put out by Bell Labs.

STYLE describes syntactic features of your text: number of words, number of sentences, kinds of sentences, readability, etc. It does this pretty much the way I described above. The program has a list of parts of speech of each word (these are the syntactic indicators) and a list of sequences of parts of speech. It runs your text through these lists and ends up with a count of the sequences. These are then analyzed and displayed. Accuracy is not great, as you might imagine, but then again, accuracy is not really the issue with this program.[5] Consider, for instance, the output of the STYLE program run on this article up to the beginning of the previous paragraph. (Figure 2)

As you can see, the issue is that the numbers don't tell me anything useful. Given my purpose and audience, I can't tell whether a Flesch reading of 10.0 is good or bad. (It varies, by the way, by more than a grade level from draft to draft.) I can't tell whether I should increase or decrease the number of non-functional words. I can't tell whether 68% subject openers is in line or not. I'm getting syntactic information, all right, but the information is not useful unless I can relate it to the content, but relating it to the content is just what this program can't do. It couldn't relate it to the content unless there were strict rules connecting syntax to semantics (as there are with

---

[5] Lorinda Cherry estimates that the type of each sentence is identified accurately about 86.5% of the time. Her sample, however, was only 20 technical documents. Authors of programs, moreover, tend to exaggerate, so we can assume it is somewhat less. L. L. Cherry and W. Vesterman, "Writing Tools - The style and diction Programs," DICTION" in <u>The UNIX User's Guide</u>, (Murray Hill, New Jersey: Bell Laboratories, 1980), p. 10.

```
readability grades:
      (Kincaid) 9.6 (auto) 9.9 (Coleman-Liau) 10.0 (Flesch)
              10.1 (59.6) sentence info:
      no. sent 163 no. wds 3081
      av sent leng 18.9 av word leng 4.65
      no. questions 6 no. imperatives 0
      no. nonfunc wds 1705 55.3% av leng 6.12
      short sent (<14) 37% (60) long sent (>29) 11% (18)
      longest sent 84 wds at sent 1; shortest sent 4 wds at
              sent 24
sentence types:
      simple 38% (62) complex 39% (63)
      compound 9% (15) compound-complex 14% (23)
word usage:
      verb types as % of total verbs
      tobe 41% (169) aux 18% (75) inf 15% (63)
      passives as % of non-inf verbs 10% (36)
      types as % of total
      prep 9.3% (287) conj 2.7% (84) adv 6.5% (200)
      noun 25.6% (790) adj 13.5% (416) pron 8.9% (275)
      nominalizations 2 % (47)
sentence beginnings:
      subject opener: noun (52) pron (28) pos (0) adj (11)
              art (20) tot 68%
      prep 6% (10) adv 10% (17)
      verb 1% (2) sub conj 9% (15) conj 2% (3)
      expletives 3% (5)
```

Figure 2: Output of STYLE Program Run on this Paper up to the Previous Paragraph

spelling checkers). (Notice, by the way, that one is unlikely to inadvertently produce mistakes that are identifiable by the program.

The makers of this program would agree with me, but they would argue that I am demanding too much of it. The program is not designed for such sophisticated users; it's designed for somebody who might have a Flesch readability of 15.0 (the authors of the Federalist Papers, says Cherry, p. 9) or somebody whose average sentence length is 38.4. "Part of your objection, they might say, is the fact that the output looks so impenetrable. But that problem has been fixed; outputs are better and more explanatory. People are told that an average sentence length of 38.4 is way out of line, and are asked to rewrite. "Unfortunately, though, my objection remains. People whose numbers are flagged for good reasons are not likely to be helped in any fundamental way by this syntactic fix, by putting in a bunch of periods. Yet they are precisely the people whose understanding of English is bad enough that they would take the computer at its word and attempt to fix the text in the way suggested. Even for them, the syntactic information must therefore be made meaningful in terms of the content for it to do any good. They are thus caught in the same bind mentioned before. They need to be able to interpret the output, before they can use the program.

The DICTION program catches words that ought to be eliminated or ought to have another word (given by the EXPLAIN program), substituted for it. It works much as spell routines do, with different wordlists. Figure 3 shows some of the output of the DICTION program run on an earlier draft of this article up to the same point. Let me caution you that I write in a somewhat breezy, slangy way, and that the Writer's Workbench is not particularly sympathetic to that kind of style.

Like the SPELL program, DICTION produces output that is inaccurate ("ludicrous" might be a better word), so inaccurate that it is useless. Would the program be more useful to others? Perhaps, but the problem of sorting through the garbage is much more difficult. To use the program properly, the writer must be able to distinguish between correct and incorrect flags, perhaps by substituting in the word suggested by EXPLAIN.Certainly, obvious garbage, like recommending that "meaningful" be eliminated in the sample sentence, can be caught this way. But what about the rest? Almost none of the recommended substitutions have the status of absolute rules, the way recommended changes in spelling do. The correct substitutions are merely considered better by most educated writers. So, in order to distinguish between the correct and the incorrect, but possible substitutions, the writer must have a sense of what most educated writers think is correct. The writer must, in other words, be the sort of person who doesn't really need the program, except to catch inadvertent errors. If the writer isn't, each flag poses a formidable problem.

In the original documentation for DICTION, Lorinda Cherry says that in its first release, between 50 and 60% of the recommended corrections were actually (oops?) made. She thinks that that is a

```
We can, in other words, ask the following question, a question
*[ which ]* is logically *[ prior to ]* the first question.

This *[ capability]* is used in word processors, formatters, and
communications programs.

*[ Essentially,]* I am saying that it is impossible for
computers to do certain things.

It is, however, difficult to prove that Q is impossible, because
*[ the fact ]* that Q hasn't been done is not *[ sufficient ]*
proof.

        When we look at a text, we respond to it *[ on the basis of
]* its meaning.

] This difference exists no matter what *[ kind of ]* text we
are talking about.

Whenever the computer appears to be manipulating
*[ meaningful ]* text, all it is *[ actual]*ly doing is simulating
the manipulation by manipulating syntactic indicators of the text.

*[ In the case of ]* single letters, there is a one-to-one
correspondence between syntactic indicators and *[ meaningful ]*
objects, so simulation is *[ quite ]* easy and *[ very ]* exact.

First, while any particular text may *[ in fact ]* receive the
correct simulated response, it is not possible to build a system
*[ which ]* can be relied on to respond correctly every time.

Fourth, when the user has to judge the output before making use
of it, most, if not *[ all of ]* the utility of the program is
lost.

For another, they know less about spelling than I do, so there
are more points where they must *[ check on ]* the program.

The program is not designed for such *[ sophisticated ]* users ;
it's designed for somebody who might have a readability of 13.

8 *[(the authors ]* of the Federalist Papers, says Cherry) or
somebody whose average sentence length is 38.  number of
sentences 228 number of phrases found 59
```

Figure 3: Output of the DICTION Program Run on this Paper up to the Previous Paragraph

sign of the program's usefulness. On the contrary. If the printout above is any indication, far more than 50% of what it catches is garbage. These scientists at Bell Labs must not know the rules and are taking the computer's word for it. They are doing the same thing the bad speller does, abandoning their natural, correct way of speaking because they don't understand the program or are too lazy to contradict it.

You can, of course, teach people how to use these programs correctly. When you do, you have also made the program useless, since people will then have learned (pretty much) to recognize these problems in their own prose, and when only inadvertent errors remain, the garbage ratio is too high. This is not such a bad result, but it's also not a very important one. It takes time to teach people how to read this analysis of syntax, time to teach them the list of easily dispensable words. That time must be taken from other pedagogical tasks. I certainly never took this time in the past, and I don't see why the sudden availability of this program should change that decision. I should add that teaching the use of this program is not completely straightforward. Lorinda Cherry's idea of correct usage is not mine.[6]

[6] Do the uninstructed actually use these programs? In a year and a half of working in the computer rooms at M.I.T., I have never seen anyone use the STYLE or DICTION programs.

Indeed, I think both programs are something of a distraction. They give people the idea that using these programs is an efficient way of improving one's prose. It's not, particularly. One problem I have with writing this paper is that I am constantly having to run the DICTION program, since I want it to show sentences that you recognize. I am, in other words, constantly revising the sentences that DICTION is flagging. Am I making the corrections it suggests? Clearly not. I am changing other things. Am I changing those sentences more than I'm changing other sentences. I think so. When I read the sentences that have been flagged, I notice that things are wrong with them, and I change the sentences. This suggests that randomly flagging sentences will produce more and better corrections than running the DICTION program.

DICTION and STYLE are still doing very crude analyses of texts. Would a more powerful program be better? I have seen one such program, IBM's EPISTLE. EPISTLE improves on the crude analyses done by the Writer's Workbench programs by including more fine-grained lists of syntactic sequence and adding syntactic indicators of semantic content. Thus, EPISTLE will pick out witch/which spelling mistakes, because only one is syntactically possible, and it will pass "do not know which is better," because it correctly analyzes the function of which in that sentence. It will catch "My father eat turkey on Thanksgiving" and might even catch "My father eats granite on Thanksgiving." These are notable accomplishments; grammar is very difficult to analyze, because it's highly meaning-dependent. Still, even though spelling accuracy goes up, overall accuracy goes down. In the demonstration I saw[7], the makers claimed that the program caught 70% of the errors in "typical business letters." That means, of course, that the garbage ratio was 30% and that it missed 30% of the actual (oops?) errors.

Is that acceptable? Surely, only marginally. The same dynamic is at work here. The person who is good at grammar doesn't need it; the person who is bad has more garbage to pick through and doesn't have the resources to decide the questionable cases. EPISTLE does provide on-line explanations (of Grundian strictness), but if you think about it, even that can't be much help. In the questionable cases, the on-line explanation is no more helpful, nor can it be, than the grammar book's or our own, and you know perfectly well how helpful they are. (Notice, of course, that our explanations are considerably more trustworthy than the computers. With our obiter dicta, the student at least knows that the judgments are considered, and thus a student has some motivation for trying ~~to~~ understand us. With this program, the judgments are not necessarily reasonable, and so the student can have no such

[7] March 10, 1984

motivation.) The questionable cases, remember, are what matters. The obvious garbage is not useful. The obvious mistakes, the mistakes due to inadvertence, are not worth the effort of catching.

I should add that EPISTLE is not really meant for us. In its current version, the program takes enormous amounts of mainframe time just to analyze a page. The program is meant to be sold to companies who can afford it. In such organizations, if my experience is any guide, the program is likely to be imposed, rather than used. Writers will be required by harried supervisors to have a Flesch score below 9 and no more than 2 flags per page.

Can Text Analyzers Be Made Better?

When I presented this argument at the CCCC convention, there were two reactions. First, people tried to minimize the seriousness of the the inaccuracy. Teachers make mistakes too, one noted computer researcher told me. This is undeniable. But it doesn't address the difference between the kinds of the mistakes the two make. People always make reasoned mistakes; their reactions are based on some reasoned application of a rule to a text. Computer responses are not reasoned; they apply entirely different rules to a collection of symbol strings. Most of the time, the two kinds of applications correlate. Much of the time, they do not. When we try to evaluate a person's mistake, we are always responding to their understanding of the rule and the meaning. When we try to evaluate the computer's, we must first try to figure out whether there's a correlation error. Not surprisingly, it takes more knowledge and effort to do the latter, and the project is not as rewarding.

The second reaction is more important. People want a technical fix for the problem; they figure that better programs will solve it. This is a reasonable reaction, but it usually betrays a lack of understanding of the magnitude of the problem. Let me try to give you an intuitive sense of how big the problem is. Any program that gives an accurate simulation of a response to meaning must, at the very least, be able to parse sentences correctly. The only way one can do this is to extend the approach that EPISTLE takes. Consider, then, how EPISTLE might parse the following sentence.

1. The car hit the man in the street.

The problem, for the moment, is to figure out the function of "in the street," so that the computer can choose correctly between the potential response strings, "Why did it hit the man there?" and "Why did it hit that man?" There is a constraint. A programmer can just arrange a correct response to this string by fiat. What we want, however, is a way which also allows us to parse similar sentences, like the following:

2. The car hit the man in the side.

3. The car hit the man in the tree.

4. The car hit the man in the park.

5. The car hit the man in the play.

The basic way of solving the problem is to multiply the number of syntactic indicators. A computer scientist might first classify all nouns according to whether they define location in space (park, street), location in the body (side), location in activity (play), or no location whatsoever. (Most nouns fall into the latter class, e.g., "the car hit the man in the noun.") Such a classification would necessarily be incomplete and ambiguous, but it would be a start. When the computer encounters a prepositional phrase like "in" which takes a location noun as its object, it would immediately consult this list. Nouns which were not locations would generate a query. Nouns which could be locations would then generate an investigation of words in the rest of the sentence, which had previously also been classified. The program might then look for correlations between types of nouns and types of locations. "Side" would go with man; "street" with car; the rest with hitting.

This wouldn't, of course, produce accurate results, so the computer scientist would have to build a more complex system, one that also took into account the verb and indeed took into account the specific noun-verb conjunction. Such a system, if it were to be general, would be staggeringly large. Its granularity would have to be so fine that it could accurately distinguish among "The car hit," "The boy hit," The car passed," and "The boy passed," since each of those produces a quite different analysis when they are coupled with the original five prepositional phrases. Even when the computer scientist has put "car hit" into an "accident" category, "boy hit" into a "fight" category, and so on, and attempted to structure the categories so that "man" and location nouns would fit together with them, he still has problems. It looks very much as if each noun must be fit to the event separately. In most cases, one would want to classify "tree," "park," and "street," as similar location nouns and treat them together. But in this case, our common sense about where cars usually go overrides this classification and forces us to suspect that "in the tree" identifies where the man had been, not the location of the event. Similar problems occur even with a word like "side." "Middle," for instance, might usually be classified with "side," but that classification would not have enough granularity to take care of this problem. "The car hit the man in the middle" means something entirely different. Or consider what happens if "side" is appropriately modified. In a sentence like "The car hit the man in the east side of the parking lot," "in the side" should be parsed in the same way that "in the street" would be.

Notice that even if we have a system that produces the correct simulation in all these cases, that does not preclude failure. If the previous sentence were, "Two men were walking side by side, one on the sidewalk and one in the street," then the parsing of the original sentence would be incorrect.

The Overall Prospects for Successful Simulation or Successful Use of Programs That Simulate

It's easy to get lost in the details, so let me sum up. The problem with computer simulation of response to meaning is that it's inaccurate. The existence of any inaccuracy of this kind requires that the user analyze each output for correctness of simulation before analyzing the relevance of the output to the user's text. Correct analysis of the problem cases requires so much knowledge that the user is unlikely to need the analysis in the first place, especially since the amount of obviously incorrect analysis (garbage) is usually high. Those users who don't have this knowledge are arguably damaged as much as they are helped by such programs. And spending the time necessary to learn them is arguably not productive.

Though I have not shown this here, the argument applies to other kinds of programs that in fact respond to meaning: invention programs, template programs (a dutiful son of invention programs for teachers or managers who want to be sure they get it right), even idea processors.[8] Interestingly enough--and important for my argument--the details matter: Each kind of simulation of meaning fails in a different way.[9]

The obvious solution is to improve the accuracy of the programs. If we don't have to check the simulation each time, because it's perfect, then we can rely on the analysis and work with it. In the examples I've given, I've tried to show how very difficult that is even with one sentence. I've also tried to show that the methods one uses for getting the right response to one sentence don't necessarily apply to other sentences, so that any fully accurate system has to build itself up sentence by sentence. This really is the heart of the matter. Syntax (form) just doesn't have much to do with semantics (content). To ask syntax to simulate semantics is rather like asking a snail to fly.

The consequences are very simple. If people do try to work with or design applications that require syntax to simulate

[8] Yes, idea processors do respond to meaning, as I show in my "What Do Idea Processors Process?"

[9] See my "The Future of Computers in Writing and the Teaching of Writing," in Straightforward Writing, forthcoming.

semantics, they can only make that application work by accepting a certain amount of inaccuracy and by working with enormous numbers of problems on a case by case basis. Whenever the number of possible sentences is large or the amount of work involved in disentangling the meaning is great, attempts at simulation get overwhelmed. Any person who designs a simulation needs to say how he or she is going to get around this problem.

The simple way of seeing whether I am right is to read the other papers in this volume. Many of them are confronting the problem of simulating response to meaning; each of those researchers finds that the problem is a serious impediment. Each researcher has his or her own way of getting around the problem, but none of them solve the problem, and none escape the consequences I describe. I wrote this paper, by the way, well before I had read the other papers.

I did, however, have a clue about what would be in them. So far, my argument rests on the obvious difficulty of getting semantics to simulate syntax. "But," you might say, "how do I know that the difficulty is so great. Maybe computer scientists, in particular, artificial intelligence specialists can tell us something more." My argument, however, is actually a little stronger than that, because it's partly based on the record of artificial intelligence research. In AI, you see, precisely the problem we are concerned with, the problem of response to meaning has been central since the beginning. Three of the central problems in AI (language translation, speech recognition, and computer vision) require solutions to the meaning problem. None of the central problems have been solved; all provisional solutions are merely convenient ways of coping with limited problems, not principled solutions. And AI research is thirty years old. Skilled computer scientists have been looking for thirty years for the solution to the problem we are just starting to encounter, and they haven't found it. Claims, therefore, that progress is being made should be looked at with a jaundiced eye.[10]

My interpretation of AI, an interpretation which follows Dreyfus, is quite pessimistic. But even the most optimistic proponents of AI acknowledge that solving the meaning problem, getting computers to simulate responses to the meaning of texts, is probably the hardest problem facing AI, the problem that will be solved last. Patrick Winston, a noted proponent of AI, told me, when I discussed computer programs to help writing, that it would be thirty years before a solution is even approached.[11] You can

---

[10] The standard account of progress, or lack thereof toward the goal of solving the meaning problem is Hubert L. Dreyfus, What Computers Can't Do (New York: Harper & Row, rev. ed. 1979).

[11] Personal communication, March 13, 1984.

safely assume this is a minimum, not a maximum figure.

This simple fact is not a cause for despair or even alarm. It should actually be encouraging. Knowing that we can't effectively write programs that respond to text, we can direct our energies toward more fruitful, more plausible goals. We can make better on-line editors, better systems for communication, better word-processors. That is what I think we should do. It is not, however, just me who thinks so. I went around the AI department at MIT soon after I wrote this paper, and I asked them what projects we in the Writing Program should undertake. "What kinds of programs should we give students, I asked." Here are answers from two noted computer scientists, whom I will let have the last word.

Better word processors
-- Joseph Weizenbaum

Better word processors
-- Douglas Hofstadter