92

Hector: Connecting Words with Definitions

Lucille Glassman, Dennis Grinberg, Cynthia Hibbard, James Meehan, Loretta Guarino Reid, and Mary-Claire van Leunen

October 20, 1992

Systems Research Center

DEC's business and technology objectives require a strong research program. The Systems Research Center (SRC) and three other research laboratories are committed to filling that need.

SRC began recruiting its first research scientists in l984—their charter, to advance the state of knowledge in all aspects of computer systems research. Our current work includes exploring high-performance personal computing, distributed computing, programming environments, system modelling techniques, specification technology, and tightly-coupled multiprocessors.

Our approach to both hardware and software research is to create and use real systems so that we can investigate their properties fully. Complex systems cannot be evaluated solely in the abstract. Based on this belief, our strategy is to demonstrate the technical and practical feasibility of our ideas by building prototypes and using them as daily tools. The experience we gain is useful in the short term in enabling us to refine our designs, and invaluable in the long term in helping us to advance the state of knowledge about those systems. Most of the major advances in information systems have come through this strategy, including time-sharing, the ArpaNet, and distributed personal computing.

SRC also performs work of a more mathematical flavor which complements our systems research. Some of this work is in established fields of theoretical computer science, such as the analysis of algorithms, computational geometry, and logics of programming. The rest of this work explores new ground motivated by problems that arise in our systems research.

DEC has a strong commitment to communicating the results and experience gained through pursuing these activities. The Company values the improved understanding that comes with exposing and testing our ideas within the research community. SRC will therefore report results in conferences, in professional journals, and in our research report series. We will seek users for our prototype systems among those with whom we have common research interests, and we will encourage collaboration with university researchers.

Robert W. Taylor, Director

Hector: Connecting Words with Definitions

Lucille Glassman, Dennis Grinberg, Cynthia Hibbard, James Meehan, Loretta Guarino Reid, and Mary-Claire van Leunen

October 20, 1992

Publication History
This paper was presented at the 8th Annual Conference of the UW Centre for the New Oxford English Dictionary and Text Research, Waterloo, Canada. October, 1992.

Affiliations
Dennis Grinberg is at the School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213-3890. Dennis worked at the Digital Equipment Corporation Systems Research Center during the summer of 1991.

©Digital Equipment Corporation 1992
This work may not be copied or reproduced in whole or in part for any commercial purpose. Permission to copy in whole or in part without payment of fee is granted for nonprofit educational and research purposes provided that all such whole or partial copies include the following: a notice that such copying is by permission of the Systems Research Center of Digital Equipment Corporation in Palo Alto, California; an acknowledgment of the authors and individual contributors to the work; and all applicable portions of the copyright notice. Copying, reproducing, or republishing for any other purpose shall require a license with payment of fee to the Systems Research Center. All rights reserved.

Abstract

Hector is a feasibility study on high-tech corpus lexicography. Oxford University Press provided the lexicographers and a corpus of 20 million words of running English text; Digital Equipment Corporation Systems Research Center provided the high tech tools to enable the lexicographers to do all of their work on-line.

The tools provide the ability to query the corpus in various ways and see the resulting matches, to write and edit dictionary entries, and to link each occurrence of a word in the corpus with its sense as displayed in the entry editor. Additional support tools give statistical information about words in the corpus, derivatives and related words, syntactic structures, collocates, and case-variants.

This report describes the tools and the status of the project as of July, 1992.

An accompanying videotape demonstrates the Hector tools. If you would like a copy, please send mail including your full postal address to *src-report@src.dec.com*.

Contents

1 Introduction

Starting in October of 1990, the Systems Research Center (SRC) of the Digital Equipment Corporation undertook with Oxford University Press a joint project called Hector. Hector was a feasibility study on high-tech corpus lexicography. The experience of the COBUILD Project at Collins and the University of Birmingham demonstrated that lexicographers need a corpus, a very large body of running text, to work on ordinary words[6, 2]. But at COBUILD the lexicographers looked at their corpus on paper and impressionistically; they spread paper concordances out on their kitchen tables, marked a few lines with colored pens, and threw them away. We wanted to offer lexicographers the opportunity to use a corpus in more rigorous and creative ways. Oxford provided the corpus and the lexicographers. We at SRC provided the high tech (and some low tech too, as you will see).

So we built an on-line corpus viewer and an on-line editor for dictionary entries. We produced indexes that would enable us to build concordances of:

words
case-variants
inflections
derivatives and related words
wordclasses
syntactic structures (including clause and sentence boundaries)
collocates

and we provided searching, sorting, and statistical information on all of them. For instance, we wanted to make it possible for the lexicographer to search for all occurrences of “cake” or “cakes” used as a finite verb in a predicate with “up.”

But we also wanted the lexicography to feed back into the corpus. We thought that a sense-tagged corpus was an interesting object in and of itself, and we wanted to let the lexicographers use one sense division in determining another. So we integrated the corpus viewer and the dictionary-entry editor; the lexicographers could link each occurrence of a word in the corpus with its sense as displayed in the entry editor, and that link could then be used in subsequent searches and sorts. Once lexicographer A had identified an occurrence of “cake” as meaning “a crusty mass,” that sense became available to lexicographer B working on “flat” in the same sentence.

We started building tools in October of 1990; the lexicographers arrived in Palo Alto (where SRC is located) and began writing definitions and linking them to the corpus in January of 1992. Now, as we write, it’s July of 1992. The pilot project will end in March of 1993. What have we learned so far?

- Natural language is hard.
- User interface design is hard.
- Speed is functionality.
- Data integrity can’t be retrofitted into a system.
- A corpus (with a set of good tools) is useful for other things besides professional lexicography.

What do we hope for by March?

- A test of the predictive value of the links between words and senses.
- An evaluation of the automatic wordclass assignments.

- 500 wonderful dictionary entries.
- A 20-million-word corpus with 300,000 words linked to those 500 dictionary entries.
- Insight into what tools the lexicographers found useful in doing their job.
- Statistical information on the distribution of words, wordclasses, and dictionary senses in the corpus.
- A contrast of the raw and the cleaned-up versions of the corpus.
- Published code for inflecting English nouns, verbs, adjectives, and adverbs.

What do we hope for in the future, beyond the end of the project?

- A solution to the copyright problems that restrict the availability of the Hector corpus.
- Another project somewhere to link all of the lexical words (but not function words, like prepositions and conjunctions and linking verbs) in a corpus to their dictionary senses.

2 Preparing the Hector Corpus

2.1 Contents of the Corpus

The Hector corpus was compiled by Oxford University Press for the Hector project and sent to us over the course of 1991.

The corpus consisted originally of 20 million words of running English text from both written and spoken sources no earlier than 1930. Its object was to sample language used in natural discourse, in a variety of social and professional contexts. It included examples of both formal and informal usage but was meant to exclude poetical or artificially self-conscious language. The written text samples included quality and popular journalism, scholarly periodicals and the journals of various professions, pop culture and hobby magazines, works of fiction, biography, autobiography, and travel. The samples of spoken language were from transcribed interviews, lectures and training sessions, radio sports commentaries, and informal meetings.

The table above shows the proportions of the main groups of texts in the corpus as it is today. Newspapers account for 60 percent of the written texts, and the composition of the corpus is heavily biased towards written text, not for any theoretical reason but because speech samples are very difficult to acquire and process. The ratio of written text to transcribed speech is approximately 32 to 1.

2.2 Cleaning Up the Corpus

There were several kinds of problems with the corpus as we received it from Oxford:

missing structure
duplication
inappropriate material
gaps
typos and misspellings
missing punctuation
inconsistent notation

Written	
newspaper journalism	59.5%
serious non-fiction	12.8%
fiction	10.9%
recreational non-fiction	5.1%
recreational magazines and periodicals	4.9%
advertising, newsletters, memos, promotional material	1.9%
serious periodicals, professional journals, news magazines	1.7%
Spoken	
all categories of speech	3.1%

The Hector Corpus as of July 1992

Throughout our work on the corpus we suffered from the lack of paper originals of the documents on which we were working, and we would recommend that any future corpus-workers be sure to obtain paper originals before they set out.

Oxford sent us the corpus essentially as one big 140-million-byte file. Our first step was to identify the document boundaries and divide the corpus into files of a convenient size, no file containing more than one document. We divided large samples into smaller chunks: for example, we divided the 7 million words of the Independent, an English daily newspaper, into 83 files.

Smaller structural elements, such as headings, salutations, addresses, datelines, bylines, and story or article boundaries, remained a problem throughout the processing; we found them as well as we could with various ad hoc techniques and marked consistently all the ones we found.

As soon as we had divided the corpus into chunks, we noticed that some of the chunks were the same as others. Most of the duplication was in the samples of journalism, where we found stories repeated—as many as sixteen times—because the samples were taken either from different editions of the same paper or from an open wire or both. Furthermore, the stories were repetitions but not identical repetitions. Sometimes the text would be altered for emphasis (the local team goes first in the local edition), sometimes for sheer journalistic exuberance—or perhaps because the paper had space to fill up. Exact duplications we could identify and delete automatically. Stories that were repeated but not exactly duplicated required ingenuity to find and judgment to eliminate; our strategy was to remove all but the longest version of closely similar stories.

We discovered inappropriate material at two different levels, whole documents and parts of documents.

We discarded some whole documents. For instance, we discarded Possession, the Booker prize winner by A. S. Byatt, because it is largely a pastiche of 19th-century prose. ("A man might die, though nothing else ailed him, only upon an extreme weariness of doing the same thing, over and over." "I can never tire of you—of this—. " "It is in the nature of the human frame to tire. Fortunately. Let us collude with necessity.") At another extreme, we discarded the Challenger Inquiry, which differed along three axes from everything else in the corpus: It was the only formal inquiry, the only sample of American speech, and the only example of technical vocabulary in its field. ("Okay, this third one, again, is—there is your plume already. This time it is lying correctly. This again is the lefthand rocket this time. The righthand rocket on the other side. The plume is coming toward you. The orbiter is here, and the external tank.")

But at a lower level, many documents that were useful overall nonetheless contained some inappropriate material: words written or spoken before 1930, passages in foreign languages, and tables. Poetry showed up in all kinds of places—it is amazing how often prose writers break into verse. Again a variety of ad hoc techniques helped us find this material. The most useful was to chop the documents up into small pieces and subject the pieces to spell-checking. Luckily, we have a good spell-checker. For instance, one piece contained this passage:

```
Right worshipfulls, This may be to acquaynt you that their is a
pore yong women in oure Towne of Aston-under-lyne infected with
a filthy deceassed called the French poxe and shee saith shee was
defiled by one Henry Heyworth a maryed man.
```

in which the spell-checker finds the following "errors":

```
Aston-under-lyne     acquaynt     oure     worshipfulls
Heyworth             deceassed    poxe     yong
Towne                maryed       shee
```

The human being following along after the spell-checker could then mark the passage to be ignored. During our first several months of work on the corpus, we simply replaced the whole of any such passage with a single notation:

```
{deadGuys}
```

but that turned out to be a mistake. Although the passage was not itself appropriate for lexical analysis, the lexicographers explained that it could help in the analysis of surrounding words. So later we would mark such passages to be ignored by the indexing programs and other tools but leave them in place for human readers looking at words before or after:

```
<ignore type=deadGuys>
Right worshipfulls, This may be to acquaynt you that their is a
pore yong women in oure Towne of Aston-under-lyne infected with
a filthy deceassed called the French poxe and shee saith shee was
defiled by one Henry Heyworth a maryed man.
</ignore>
```

(At this writing, we haven't yet gone back and restored the passages we deleted.)

The spell-checker was also helpful in finding passages in foreign languages. We were careful to mark off only whole sentences and passages in foreign languages; we did not mark individual words, which might be new assimilations.

Finding tables and verse was harder; some we were able to find automatically by looking (for instance) for short line lengths or runs of numbers, but some we just stumbled upon in looking for other things.

We found some gaps in the material, and one document simply left off mid-sentence. In an ideal world we would have restored the missing material, but we didn't.

Some of the text had been typed in, some had been scanned with an optical character recognizer, and most had been produced using electronic typesetting. All of these methods produce mistakes and so we were not surprised to find typographical errors, OCR errors, and transcription errors in the text. Because we used the spell-checker heavily to identify inappropriate material, we also found the sorts of errors that a spell-checker can find, and those we corrected with a construction called the typo sundry:

```
{typo bad="theoretcial",good="theoretical"}
```

We used the typo sundry for typos and OCR errors and misspellings and whatever without distinction; we couldn't see any point in trying to guess the origin of each error. The typo sundry keeps both the erroneous version and the corrected version, but only the corrected version gets indexed. (It may seem comical that we carefully preserved this information while discarding long sections of tables and verse and so on, but we were afraid that the fatigue of correcting errors by hand might lead us to mis-label perfectly reasonable forms as typos.)

Of course, spell-checkers can find only certain classes of spelling and typographical errors. We made no systematic attempt to find the ones that the spell-checker couldn't catch. If we chanced upon a mistake, we corrected it with the typo sundry, but that's all.

The OCRed material had much of the sentence punctuation missing or misread—commas for full stops were the commonest error, and the one that seriously reduced our chance of finding sentence boundaries. We used regular expressions to find such places and human labor to correct them.[1] Unlike our careful preservation of lexical evidence in dealing with typos and spelling errors, we silently corrected punctuation errors and left no trace behind. (The corpus would therefore be useless for a study of punctuation; too bad.)

The representation of essentially everything in the corpus except the roman alphabet varied from one document to another (and sometimes within a document as well). For instance, we found fourteen ways of representing an em-dash and rampant inconsistency in the conventions used for embedded quotes and final quotes. Again, we used a combination of regular expressions and sweated labor to make notation fairly consistent in the corpus. We started out over-enthusiastically trying to make everything perfectly consistent and eventually settled on a more manageable set of notational conventions, laid out in Appendix A.

The reduction in bulk resulting from all this cleaning-up was substantial. In round numbers, we refer to the corpus as having 20 million words. But in reality, Oxford provided us with 21.7 million at the beginning of the processing, just for safety's sake. By now, we have reduced the original 21.7 million words to 18

[1]Regular expressions let a computer user search for patterns in text; Unix systems are typically rich in regular-expression tools, other systems often lack them.

million, and still shrinking. We nonetheless call it 20 million words because we're pretty confident that any other 20-million-word corpus would yield about the same amount if processed the same way.

2.3 Why Cleanse the Corpus?

We had different motives for the different kinds of cleaning-up we did. For instance, we felt that we had no choice but to supply missing structure to make the documents tractable. On the other hand, filling gaps seemed nice but not worth the effort.

Regularizing notation seemed to us to be an engineering necessity. The problem about inconsistent notation is that unless you fix it in the source, all the downstream tools become more complicated. If you leave in fourteen kinds of dashes, the tagger, the parser, and all the indexing tools need to understand fourteen kinds of dashes.

The duplications in the corpus struck us as irritating (and sure enough, the lexicographers complained about the ones we didn't catch). More seriously, though, we were also afraid that they would skew the lexical evidence. For instance, one of the newspaper stories in the corpus used the word "perch" and the word "pope" in adjacent sentences. If that story appears only once, the most significant collocate for "perch" in the corpus is "skimmer" (a skimmer bream is another kind of fish) and the most significant collocate for "pope" is "St" (abbreviation for Saint). But if the story is repeated often enough, "perch" and "pope" will appear to be significant collocates of one another.

We agonized over removing inappropriate documents, but in the end we did decide to eliminate a few troublesome ones. Again, we were worried about skewing the lexical evidence. In the Challenger Inquiry, for instance, the word "booster" appears 464 times; in the whole rest of the corpus it appears 19 times. And we were convinced that the lexicographers would refuse any new information offered by so idiosyncratic a document. An expression that occurred only in the Challenger Inquiry might represent American speech, or technical jargon, or formal discomfort, or some combination of the three; without parallel documents along each axis, the lexicographers would have no way of guessing what was going on. The expression "O-ring," for instance, occurs 1,733 times in the Challenger Inquiry and nowhere else in the corpus. What lexicographer would feel confident, on the basis of that evidence, to make any statement whatsoever about the term?

The other kinds of cleaning-up that we did—removing inappropriate material within documents, correcting typos and misspellings, and supplying missing punctuation—took place on much shakier ground, because we were aware that we couldn't fix everything. 20 million words is just too much even for rabid enthusiasts with good machine resources and a year to spend. We guess that we may have read as much as one line in every seven or eight of the corpus; but a lot of lines remain unread.

There are two points of view from which to object to what we did. The first is that we were marking the cards; the second is that we were wasting our time.

Luckily, we're going to be able to shed some light here. By the end of the project we hope to be able to report statistical differences between the raw and cleaned-up versions of the corpus and let the world decide whether the cleansing was worth while. Being sensible people, we hope that the second point of view is correct – that extensive hand work on a corpus doesn't change it much, so there's no point in doing it. But we believe we may be the first ones in a position to test this hypothesis.

2.4 Adam: Our Wordclass Tagger

To help the lexicographers divide the words into senses, we first identified the wordclass ("part of speech") of every word in the corpus. Fortunately, wordclasses can be identified automatically with a reasonable degree of accuracy. We adapted the algorithm described by Ken Church, and used the data from the Lancaster-Oslo-

Bergen corpus (LOB), graciously provided to us in machine-readable form by Knut Hofland, to produce a wordclass tagger called Adam[1, 3].

Briefly, the LOB data tells us how often a word was assigned a particular wordclass and how often any combination of three wordclasses occurred in a row. From that, we can compute the lexical probability (the odds that word X is of wordclass A) and the contextual probability (the odds that wordclass A will be followed by wordclasses B and C). We multiply the lexical probability by the contextual probability and take for each word the combination with the greatest product.

Church tested the algorithm on a small set of texts. As in the LOB corpus, the sentence boundaries in Church's texts were already marked. He reported a good rate of accuracy.

Since the sentence boundaries in the Hector corpus were not marked in advance, we adapted the tagger to find them. In fact, although there was an LOB wordclass tag for the beginning of a sentence, Hofland didn't initially send us the data on triples where the beginning-of-sentence tag was the middle tag; it hadn't occurred to him that we were going to use this data to determine sentence boundaries automatically.

Even with the complete data, sentence boundaries remained a problem. The beginning of a sentence is invisible: there's no text to tag, not even a punctuation mark, so there's nothing that has a lexical probability for this funny kind of wordclass. We modified the algorithm to handle this situation.

Adam is fast enough to tag the entire corpus overnight by using a group of machines linked over a network.

2.5 The Houghton Mifflin Parser

Looking back on it, we wonder how we were bold enough to undertake the Hector project before Houghton Mifflin generously provided us with the use of their parser. It turned out to be the linchpin of the Hector system.

The Houghton Mifflin parser is used in their CorrecText Grammar Correction System [2]. It breaks the input text into sentences, assigns a wordclass tag to each word in the sentence, and identifies clause boundaries for each sentence, subject and predicate boundaries for each clause, and prepositional phrase boundaries.

We changed the program as little as possible, so that we could take improved versions as they became available. In addition to increasing the parser's size limits as much as possible, we made two modifications: We filter the input file before it reaches the parser, passing through only those sections containing information to be indexed. And we report the results in our preferred indexing format—every word is identified by its character position and length in the original source file.

These two modifications interact in ways that make each task harder. For the text that we pass to the parser, we must preserve information about its location in the original source file. Some of the parser's algorithms make it difficult to track the locations of the text that it is parsing. However, we managed to produce a version that generally succeeded in tracking the words; occasionally, we would change the spacing or punctuation in the corpus when we could not succeed in parsing the original version.

Why did we make these modifications? We found that unless we filtered out material that couldn't be parsed (like SGML markings and headers and addresses), the parser became very, very slow as it beat its head against insoluble problems. With the modifications, the parser was fast enough so that we could yoke a group of machines together and parse the whole corpus overnight.

[2]CorrecText is the registered trademark of the grammar-checker, which we understand is marketed with several different front ends.

2.6 Which Word Is This?

There are a number of different tools that operate on the corpus, and it was important that they be able to identify and refer to the words in a consistent manner. For instance, both Adam and the Houghton Mifflin parser were generating wordclass tags, and the indexing program needed to know when different tags applied to the same word. If one program thought of a word as a pair of offsets and another program thought of it as an offset and length, the conversion would be trivial but we'd waste time reconciling the two. If one program thought of a word as including trailing whitespace and another thought of a word as excluding it, we'd have chaos on our hands.

We decided to use the character position and length of a word in its corpus file as the standard representation of a corpus word for data produced by programs. Individual analysis tools worked with a single source file containing the information on the position, length, and corpus file of every word in the corpus; all the tools produced values in terms of this standard representation.

The indexing program combined all the data files from the different corpus files, and assigned each corpus word a unique word index. It also provided a mechanism for translating between these word indexes and the source file/character position representation.

We knew that the corpus was undergoing constant modification as we cleaned it up. Furthermore, we suspected that it would be necessary to continue to modify the corpus after the lexicographers had started sense-tagging it. So we wrote a tool that would analyze additions and deletions to the corpus and compute the difference in the word indexes that these changes implied. For instance, if we had to remove a sentence from the corpus, the word indexes of all the following words would get decremented by the number of words in the sentence.

By storing all the data, including the lexicographers' sense-tags, in terms of these word indexes, and applying this analysis tool, we can continue certain classes of improvements to the corpus without losing the sense-tag work that was done on an earlier version.

Now in reality, we have been so busy since the lexicographers arrived that the corpus might just as well have been frozen. But it is comforting to know that we can accommodate any changes we need to make—such as removing some more of those blasted duplicates when we get the time.

2.7 Which Sentence Is This?

With sentences as with words, we couldn't afford to let the different tools have different ideas about what constituted a sentence and where each sentence began and ended. Since we were getting so much more information out of the Houghton Mifflin parser than out of Adam (clause, subject, predicate, prepositional phrase) we decided to let Houghton Mifflin decide where the sentence boundaries were in the corpus. In the early months of the project, before we had the Houghton Mifflin parser, we had labored mightily to get Adam to find sentence boundaries and even wrote a stand-alone heuristic sentence chunker. We were therefore in an unusually good position to appreciate the accuracy of the Houghton Mifflin parser in recognizing sentence boundaries in the face of such unwieldy material as lists, contract language, and speech transcripts.

2.8 The "hi" Server

Mike Burrows, one of our colleagues at SRC, helped us out by writing an indexing program called the "hi" server, "hi" being short for Hector Information. One of the advantages to doing this project at SRC was the chance to tap the expertise of folks like Mike.

We had originally thought of using the Pat program from Open Text for indexing the corpus, but we stumbled over some bugs and Pat was unable to rebuild the index overnight, which was a requirement for us.[3] Because Mike was doing research in indexing, he consented to take us on as his guinea pigs, and the relationship worked well. This paper is not the right place to describe the hi server (nor could we if we wanted to). Suffice it to say that the server keeps compressed inverted indexes in memory and uses them to answer queries with blinding speed; building new indexes on the entire corpus and all the wordclass and syntax information takes only the few hours that remain in the night after Adam and the Houghton Mifflin parser have completed. It does what we need, and that's all we need to know.

2.9 Target Words

Since Hector was a pilot project, we wanted to ensure we learned as much as possible about the effectiveness of our tools. We wanted the lexicographers to use the corpus tools heavily for the duration of the project. Since the project would cover only part of the lexicon, we wanted that work done on appropriate words. If the project were covering the entire lexicon, it wouldn't have mattered what order the words were taken.

We decided to assemble a list of target words for the pilot project. Our hope is to have entries for at least those words by the end of the pilot project.

We wanted to avoid words that occurred infrequently in the corpus, since the corpus tools would provide little insight in writing dictionary entries for such words. We also wanted to avoid words that occurred very frequently, since we suspected that such words presented difficult problems in lexicography, independent of the corpus analysis, that would require too much of the lexicographers' time.

At Oxford, the headword list of the 8th edition of the Concise Oxford Dictionary was mapped onto a corpus frequency list in which words were grouped with their inflections. The headwords were then grouped into bands that reflected their frequency in the corpus. Band 7 contained words with 200-500 occurrences in the 12.8-million-word corpus that had been collected at that point, meaty but not overwhelming. We chose a target list of 570 words from that band; their actual frequencies in the 20-million-word corpus ranged from a low of 260 ("sweat") to a high of 3099 ("practise"). The target words appear in Appendix B.

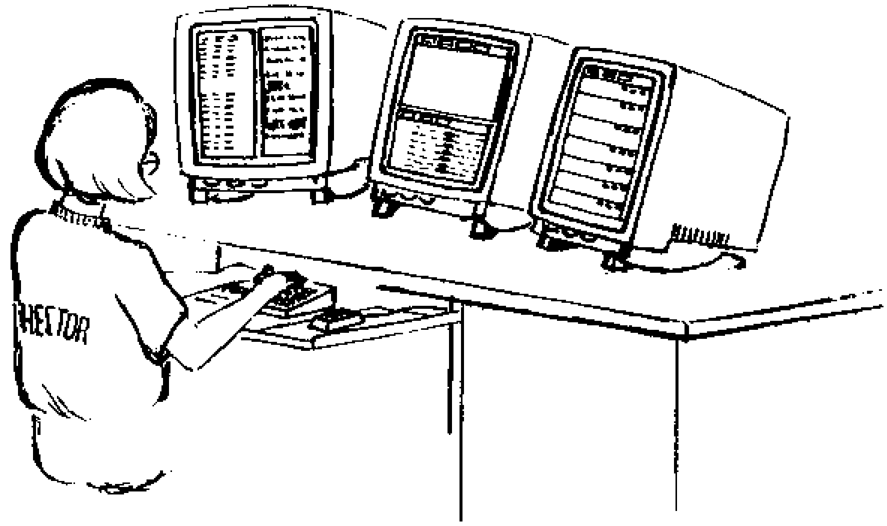
Later we added another set of target words, the wave-2 words, which occur in the corpus more than 100 and fewer than 1500 times and occur in the vicinity of the initial targets. So for instance the wave-2 word "pint" and its inflections occur 284 times in the corpus, and they occur in the vicinity of the band-7 words "bitter," "boil," "cream," "equivalent," and "sugar" 28 times. The wave-2 words appear in Appendix C.

The idea of the wave-2 words is to see whether the work already done on the target words aids future work; does this kind of corpus lexicography get easier as you go along? Many of the wave-2 words appeared to be profoundly uninteresting, but of course one of the amazing things for us is to watch the lexicographers tease out threads of meaning and usage in words we hadn't realized were at all complicated. To date the lexicographers have done only a very few wave-2 words, so we have no results to report now.

3 The Lexicographer's Workbench

This is what Hector looked like from the lexicographers' point of view when they arrived in January:

[3] Open Text is at 180 King Street South, Suite 550, Waterloo, Ontario N2J 1P8. We used Pat extensively in other parts of the project and found it quite satisfactory.



The workstation is a DECstation 5000 with three color screens and a single mouse and keyboard. The center screen displays the corpus viewer, which we call Argus. The right-hand screen displays the dictionary-entry editor, which we call Ajax. The left-hand screen we call Atlas, to maintain the theme of classical A's, and it displays mail, document editing, various small utilities, and of course a solitaire game to keep the lexicographers occupied when the corpus viewer and the dictionary-entry editor are malfunctioning.

In a highly oversimplified scenario, suppose that the lexicographer decides to write the dictionary entry for the word "tap." First she [4] uses the corpus viewer (on the center screen) to look at "tap" as a noun:

```
chen windows let out the promising tap of fork on plate and the s
tat last summer, when suddenly the tap was turned off. Now, just
un away to Manchester and become a tap dancer?" Leonard Ford, who
fe like royalty. If you are royal, tap dancing is out. Another re
mother turning on the back-kitchen tap to fill the kettle for tea
e division in Washington. From the tap on Bloch's phone, it knew
e voice. Gregory Hines, the superb tap dancer turned movie star,
 controlling the church's main gas tap from under his personal pe
ae up to an inch long polluted the tap water of eight London boro
 Miller when, changing feet like a tap dancer, he somehow kicked
f select female genes have been on tap in recent years through th
```

... and so on. She decides to divide "tap" the noun into three senses: tap a little sharp sound, tap a spigot, and tap a surreptitious listening device. She sketches out these senses in the dictionary-entry editor (the right-hand screen) and assigns each a mnemonic label: CLICK, VALVE, and SPY. Back in the corpus viewer, she hypothesizes that "tap" as a noun (an attributive noun) followed by "water" will always be the VALVE sense of "tap," so she tries a search with those constraints:

[4]Half the lexicographers were men, the other half were women; we're using "she" as a generic pronoun.

```
ae up to an inch long polluted the tap water of eight London boro
 ministers have convinced her that tap water really is safe and w
,000 people were warned not to use tap water after diesel oil lea
G has gone out to MPs not to drink tap water in some buildings at
e country about the quality of our tap water; fears of the conseq
 change in the fish's health. Pure tap water might be all right f
```

Sure enough, that was right, so she tags all those occurrences with the VALVE tag. Now because the sense-tags are immediately available for further searches and sorts, she can ask for all uses of "tap" as a noun followed by any other noun but not tagged as VALVE:

```
fe like royalty. If you are royal, tap dancing is out. Another re
e voice. Gregory Hines, the superb tap dancer turned movie star,
 keeping the game moving by taking tap penalties instead of letti
bbies.  'It's got door handles and tap fittings which can be chan
t, a true cat-lover and a splendid tap dancer. He even taught him
tland B prop, trundled over from a tap penalty and Hull converted
Happy". (Field had once mastered a tap routine to the Youmans son
als. The answer is to re-grind the tap seating with a simple but
eating with a simple but effective tap re-seating tool which will
> HAVE you ever wanted to tango or tap dance? If so, why not pop
```

She decides that any occurrence of "tap" followed by any inflection or derivative of "dance" is the CLICK sense of "tap", so she tries a search with those added constraints:

```
un away to Manchester and become a tap dancer?" Leonard Ford, who
fe like royalty. If you are royal, tap dancing is out. Another re
e voice. Gregory Hines, the superb tap dancer turned movie star,
 Miller when, changing feet like a tap dancer, he somehow kicked
t, a true cat-lover and a splendid tap dancer. He even taught him
> HAVE you ever wanted to tango or tap dance? If so, why not pop
l> IF YOU fancied a quick tango or tap dance then Gosford Hill Co
in Cornwall is to observe seagulls tap dancing on the lawn after
 and went to Egypt. I did my usual tap dancing on the table, but
in pork-pie hats and bell bottoms, tap dancing on the bollards li
```

Again the search pays off and she marks all those occurrences as CLICK.

52 instances of "tap" down, 513 to go.

3.1 Hooking Words and Definitions Together

The corpus viewer and the dictionary-entry editor have to work together to get the corpus sense-tagged. The corpus viewer knows about the corpus, the dictionary-entry editor knows about the word senses. The dictionary-entry editor has to tell the corpus viewer about the senses. The lexicographer tells the dictionary-entry editor which entries are of interest to the corpus viewer by "activating" them. Any number of entries can be active at once; when an entry is active, all its senses are active.

For each active sense, the dictionary-entry editor tells the corpus viewer its mnemonic label (e.g. CLICK, VALVE, SPY), its sense uid (the true identifier of the sense, a number unique over the whole dictionary), and other information the corpus viewer uses for sorting purposes (headword, homograph number, and sense number). The active mnemonics must be unique. Hence there must be no duplicate mnemonics within an entry, or in two entries that will be active at the same time.

While an entry is active, the dictionary-entry editor tells the corpus viewer about every change to its set of senses: additions and deletions of senses, and changes to the mnemonics, homograph numbers, sense numbers, and headword fields.

Why not just have the whole dictionary active at once? It would make the whole system too slow, and the mnemonics (which the lexicographers far prefer to the six-digit uids) would have to be unique across the whole dictionary, not just across the active entries.

3.2 Argus: The Corpus Viewer

Named for the mythological creature with a hundred eyes, Argus provides a dynamic, interactive concordance for the lexicographer. It occupies the middle screen because we think of it as being at the center of the work the lexicographers are doing during the pilot project. There are two main windows in the corpus viewer, the query window and the concordance window. In the query window, the lexicographer specifies what she wants to search for. Then when she clicks the Search button, the corpus viewer displays the matches in the concordance window, where she can rearrange them at will, and where she links occurrences in the corpus with dictionary senses in the dictionary-entry editor.

3.2.1 Searching for a Word or a List of Words

The first step in using the corpus viewer is to specify a query. A single word is the simplest query. The lexicographer can search for a whole list of search words at once:

```
hand | hands | handing | handed
```

The search words need not be related:

```
hand | any | hogwash | yellow | Patricia
```

(although it's hard to imagine why anyone would make such a query).

Occasionally the lexicographer may type in a list of words by hand, but usually she types in only one word and then has the corpus viewer generate a list from that word.

The simplest generated sets are the case-variants. All the words in the corpus have been indexed, and the index is case-sensitive. For "hand," the generated case-variants are "Hand" and "HAND." The initial-capital variant is useful for matching words at the beginning of sentences; the fully capitalized variant is useful for matching words in newspaper headlines. Other possible case-variants, such as "hAnD," are not generally useful; when they occur at all in the corpus, they usually indicate an acronym or an initialism.

The corpus viewer can also generate inflections for nouns, verbs, adjectives, and adverbs. If the lexicographer types the word "hand" and asks the corpus viewer to inflect it as a noun, the corpus viewer will provide the list:

```
  hand   | Hand   | HAND
| hands  | Hands  | HANDS
| hand's | Hand's | HAND'S
| hands' | Hands' | HANDS'
```

The inflection code can handle regular and irregular inflections and has been steadily improving over the course of the lexicographers' stay with us, as they point out errors. We plan to publish it at the end of the project.

Told to expand "hand" as an adjective, the corpus viewer will blithely do so:

```
  hand    | Hand    | HAND
| hander  | Hander  | HANDER
| handest | Handest | HANDEST
```

Non-words in the list such as "handest" don't present a problem because they won't occur in the index. Of course, some words look like non-words but turn out to be real. "Hand, hander, handest" looks quite bogus, but in another setting "hander" is a perfectly sensible word:

```
A strong left hander, she is a pupil at Banbury School.
```

By editing the type-in area, the lexicographer can delete unwanted words from the list.

The facility that naive users find the most mysterious is the one for generating all the "related" words, that is, all the words found in the Hector corpus that may be derived from the search word, or compounds of the word, or otherwise related to it. The words related to "cake," for example, include:

```
anti-caking | cakehole    | fruitcake    | salt-caked
beefcake    | cakewalk    | layer-cake   | sheepcake
cake-eaters | cheesecake  | mud-caked    | shortcake
cake-holes  | cupcakes    | oatcakes     | wedding-cake
cake-icing  | filth-caked | pancake-like | worm-cake
cake-mixes  | fishcake    | rock-cakes   | yellowcake
```

This information has already been stored for all the Hector target words. If the lexicographer asks for words related to a word that's not a target, the corpus viewer sends electronic mail to us, and we then find all the related words and store them so that the results are available for subsequent requests. [5]

3.2.2 Searching for a Wordclass or Wordclasses

The lexicographer can specify that the search be restricted to certain wordclasses. For example, if the query word is "butter," the lexicographer can search for it used only as a verb.

To provide wordclass information for the search, the corpus viewer uses the union of the information from Adam and the Houghton Mifflin parser. That is, if the search is restricted to, say, adverbs, then the concordance will include words that either Adam or the Houghton Mifflin parser marked as an adverb. The corpus viewer doesn't let the lexicographer specify wordclass by tagger ("adverb according to Adam" or "reflexive pronoun according to HM"), but it should—and will, shortly after we've finished writing this paper. (One of the reasons to write up results is to be shamed into correcting small errors and infelicities.)

The lexicographers can choose a wordclass or wordclasses from this list:

```
noun                determiner
proper noun         number
verb                ordinal
adjective           modal
adverb              auxiliary
degree adverb       possessive
preposition         infinitive marker
```

[5]The lexicographers had requested such a facility long before they arrived but were unable to suggest any principled way to provide it, so we settled on the low-tech solution. We do "grep -i" using the simplest of simple Unix tools for the word in the index (with a small bit of cleverness about trailing vowels and adjectival forms) and then read through the results and reject ones that aren't related to the word. If anybody knows how to write a program that can find "cakehole" as a derivative of "cake" or tell that "inclemencies" is related to "clement" while "encirclements" is not, our hat's off to them.

```
personal pronoun      negative
reflexive pronoun     conjunction
pronoun               other
```

We are still struggling with how to give the lexicographers finer control over wordclass constraints.

If the lexicographer constrains the search to a certain wordclass or wordclasses, she needn't provide a search word; the corpus viewer will search for whole wordclasses such as pronouns or determiners. In practice, the lexicographers don't have occasion to use this ability for the main search because dictionaries are arranged by word, not wordclass.

3.2.3 Searching for Syntax, Position, Genre, Authorship, Etc.

The Houghton Mifflin parser identifies the sentence and clause boundaries and the subject, predicate, and prepositional-phrase boundaries in the corpus, but the corpus viewer doesn't permit the lexicographers to use those boundaries as constraints in searches. We also know the genre and authorship of every document in the corpus, but again, the corpus viewer doesn't permit the lexicographers to use this knowledge as constraints in searches.

The corpus viewer used to permit all these kinds of constraints when the lexicographers first arrived, but Hector had a lot of starting-up problems at that point, and we found ourselves simply jettisoning some kinds of searching constraints. We plan before the end of the project to experiment with adding back in constraints on syntax, position, genre, authorship, etc., to see whether the lexicographers find them useful.

3.2.4 Searching for a Sense or Senses

In addition to wordclass constraints, the lexicographer can place sense-tag constraints on the search. The tags themselves are simply the lexicographers' mnemonics for dictionary senses, and the entries need to be active to be used in a search. ("Active," you may recall, means that the dictionary-entry editor is telling the corpus viewer about the senses and monitoring changes to them.)

There are three predefined tags, which are always active: P for proper name, T for typographical error, and U for unassignable. The word "Twist" in the name "Oliver Twist," for example, would be tagged P.

The lexicographer can also ask for all sense-tagged words or no sense-tagged words even when only some or none of those senses are active. By excluding all sense-tagged words, for example, the lexicographer can easily see how much work is left to be done on a particular word.

3.2.5 Searching for Words with Collocates

The corpus viewer also lets the lexicographers specify search words that occur in the context of other words, their collocates. The lexicographer specifies a distance and a direction between the search word and the collocate. For example, -5,+3 means that the collocate must occur within 5 words to the left of the search word or within 3 words to the right. +1 means that the collocate must be the word immediately following the search word. -5,-2 means that the collocate must occur no more than 5 but at least 2 words to the left of the search word.

Like search words, collocates can be lists rather than single words; the corpus viewer can generate the lists automatically; and collocates can be constrained with wordclass and sense-tag restrictions. Like search words, collocates need not be any particular word as long as they're constrained by at least one wordclass or sense-tag. The lexicographers don't have any reason to search for any noun, but they often search for some specific word with any noun as a collocate.

The lexicographer can specify any number of collocates, including nested collocates: "drop" with "potato" as a -5,+5 collocate and "hot" as a -1 collocate of "potato"—i.e. drop something like a hot potato.

Although the corpus viewer imposes no limit on the number of collocates or the depth of nesting, the lexicographers rarely construct complex queries.

We were very surprised that the lexicographers preferred in specifying collocates to use pure position rather than some kind of syntactic constraint like “within the same sentence” or “within the same clause.” We might try to convince them to experiment with syntactic constraints before the project is over.

3.2.6 Looking at Concordances

The result of doing a search is a concordance. As each example is found, it is written in the concordance window, which scrolls up to accommodate successive lines.

Each line of the concordance contains three fields: the sense-tag, the source name, and the text.

The sense-tag field shows what sense-tag, if any, has been assigned to the target word of this line. If the tag is active, then the mnemonic is shown; otherwise the uid is shown.

The source name is a 6-letter abbreviation for the corpus document in which the citation appears.

The text of the citation shows 80 characters from the corpus, and the search words are vertically aligned. (The name for this kind of format is KWIC, keyword in context.) We experimented with other display formats, particularly with whole sentences in a variable-width font. The lexicographers hated it. The search words didn’t get lined up in the display, so they needed to be highlighted; we chose to highlight them in red. The lexicographers referred to the highlights as “the river of blood.”

If the query includes collocates, they are highlighted on the concordance line. (The highlight is green, and since the collocates are dotted around, one lexicographer has suggested they might call it “the meadow of shamrocks.”)

The lexicographer can get a quick preview by asking for a count of how many concordance lines a query would produce. The concordance window also contains facilities for expanding a concordance line in a pop-up window containing either a few paragraphs or the entire document. The wordclass and syntactic information for the citation is available in another pop-up window.

Each time the lexicographer clicks Search, a new concordance appears in the window, replacing the previous contents. There is no facility for seeing more than one concordance at a time (although of course the lexicographer can write a complicated query that produces what she thinks of as two concordances combined); we could easily provide one, but the lexicographers haven’t asked for it. There is a facility for saving a concordance in a file, either in KWIC format or as whole sentences.

The initial display of the concordance lines is roughly in genre order (actually in corpus order, with the corpus arranged roughly in genre order). Once the concordance lines are displayed, the lexicographer can sort them. There are five primary sorts:

- Sort by the first word following the target word, and if there’s a tie, the second, and so on.
- Sort by the first word preceding the target word, and if there’s a tie, the second, and so on.
- Sort on the search words. For instance, when the lexicographer has searched for “hand | hands,” this sort puts all “hand” concordance lines together followed by all “hands” lines.
- Sort by the order of the documents in the corpus (if the concordance lines have been sorted into some other order and the lexicographer wants them back in corpus order).
- Sort by dictionary sense.

The lexicographers can break ties in the primary sort by specifying a secondary sort, which can be another of the five orderings above or “Don’t care.” “Don’t care” is the default secondary sort.

The corpus viewer also contains a few other utilities. One, for instance, pops up an edit-window that the lexicographers use to compose electronic mail; the most recent error message from the corpus viewer is automatically copied into the body of the e-mail text. This makes it simple for the lexicographers to send more meaningful questions—and bug-reports—to the developers. There is also a button that pops up the complete on-line manual for the corpus viewer.

3.2.7 Sense-Tagging Concordance Lines

The lexicographer can sense-tag the search word on a concordance line by typing into the sense-tag field. A number of different things can go there:

- An active sense-tag mnemonic. The tags P, T, and U are always active.
- An active sense-tag mnemonic followed by a question mark, showing that the lexicographer feels a degree of uncertainty about the tag. The question mark is only a note to the lexicographer; it does not influence searches in any way.
- An active sense-tag mnemonic followed by one of the following suffixes:
  - P Even though the word appears in a proper name, its original sense is still relevant. For example, the word "Diet" in the proper name "Diet Pepsi" is related to the food-related sense of the word "diet." The -P suffix contrasts with the P tag. Dickens may have chosen Oliver Twist's name for some reason connected with a sense of the word "twist," but that sense is no longer relevant in the name.
  - M A metaphorical use of the sense.
  - X An exploitation of the sense—some kind of odd syntax or setting. For instance, the lexicographer might note that it's generally one person who twists another's arm; but "cruel fate twisted her arm" is still that same sense of twist even though it has an atypical subject.

  These suffixes can be followed by the question mark.
- Any number of active sense-tags mnemonics (with suffixes) connected by the word OR. This indicates that more than one sense is involved or that it is difficult to distinguish between them. A word marked with more than one tag will be found during a search for any of its tags.

The corpus viewer gives the lexicographer several ways of batch-tagging whole groups of concordance lines in one fell swoop. As a precaution against accidents, the corpus viewer requires the lexicographer to click special buttons before overwriting or removing a sense-tag.

Finally, there is the button labelled Commit. No changes in the sense-tag assignments take effect until the lexicographer clicks this button. When she does, the corpus viewer conveys the new assignments to the index server, which makes the new information available to all the lexicographers.

3.3 Ajax: The Dictionary-Entry Editor

The lexicographers use the dictionary-entry editor (the right-hand screen) to create new dictionary entries and to look at existing entries. The dictionary-entry editor permits the lexicographer to work on as many entries at a time as she wishes, each in a separate window.

Here are the goals that the dictionary-entry editor set out to achieve:

The entries that the dictionary-entry editor produced had to be suitable both for typesetting programs and for computer analysis. It was a goal for the entries to contain clear marks on each type of information, such as register and grammar, so that no one in the future would have to extract such information by attempting to analyze more general text. Another goal was to capture accurately the hierarchical relationship of the entry, to avoid duplicating information within the entry.

We wanted to shield the lexicographers from the details of the representation as much as possible, so that they could focus on the content of an entry rather than its form.

The dictionary-entry editor had to permit the structure of entries to evolve, to accommodate new lexicographic insights as the project progressed.

Because an overall goal of the project was to mark the corpus with sense-tags, the dictionary-entry editor was responsible for assigning and maintaining the "identity" of senses in such a way that the sense-tags would continue to be valid as the lexicographer revised an entry. In particular, the lexicographer needed to be able to add a sense, merge two senses into a single sense, make one sense a subsense of another, or change the order of senses within the entry without invalidating the sense-tagging that had already been done.

It had to be easy to copy examples from the corpus into dictionary entries.

3.3.1 Writing a Dictionary Entry

The dictionary-entry editor permits three different views on an entry, a simulated print view, a full view, and a skeletal view. The simulated print view is displayed in a separate window and can be seen at the same time as the full view or the skeletal view. The lexicographer must, however, choose between the full view and the skeletal view in the main window; only one can be seen at a time.

The simulated print view permits the lexicographer to see how an entry will look; this view is provided by a program called Sid, written at Oxford for viewing dictionaries. It permits the lexicographer to proof the entry for the dictionary and is familiar and easy to read when a lexicographer wants to check an entry quickly, for instance, to compare it with a similar entry that she's actually working on. This view is read-only; it is not possible to edit the contents of the simulated print view directly.

The full view presents an explicit representation of the entry hierarchy. The goal of the full view is to make the complete entry structure visible and accessible. In the full view, the lexicographer can produce any legal entry, no matter how complex. The hierarchy itself can be modified in a fairly straightforward manner. For instance, it is easy to take a sense and all its subsenses and move it anywhere in the hierarchy—even make it a subsense of another sense.

The full view, however, can deal only with a complete entry. The hierarchical framework must always be in place. So the lexicographer has to consider the structure of the entire entry – sometimes before she is ready. Worse yet, early versions of the full view wasted so much screen space making the details and relationships of the hierarchy clear that the lexicographer was forced to think about the whole entry, and she couldn't even see it.

The skeletal view in the dictionary-entry editor was motivated by the goal of making it easy for the lexicographer to assemble an entry bottom up—to start identifying senses of a word and worry about their relation to one another after they have been identified. The skeletal view tries to make as many senses as possible visible on the screen and makes it easy to add and delete senses.

But the lexicographer can create only a limited set of fields in the skeletal view. In particular, the fields must all be fields of a particular sense; they can't, for instance, be fields of a homograph. This restriction has led lexicographers to encode information in inappropriate fields. For instance, if the lexicographer wants to note variant forms of the headword, she can't record this information directly in the skeletal view. Instead of switching to the full view, she may record variant forms in a note field, or as part of the definition, or leave them out altogether. At some level the lexicographers understand that if they do that we will no longer be able to analyze variant forms in the entries. But in the heat of frenzied composition they forget, or maybe they don't yet believe in the utility of having us analyze variant forms.

Both the full view and the skeletal view in the dictionary-entry editor support an operation called folding. Folding a sense removes from the screen all its fields except the tag, sense number, grammar, and definition. Folding a sense reduces the amount of space it needs on the screen, so folding an entire entry lets the lexicographer get an overview of the structure and content of the entry.

Lexicographers do most of their work in the skeletal view. Although the range of entries it can produce is restricted, it is adequate for most of the work in the pilot project. Its model of how a lexicographer develops an entry corresponds to the way the lexicographers actually work; the full view corresponds to what we as users and computer scientists want from an entry, but not to how the lexicographer wants to build it up. It takes anywhere from 5 to 60 seconds to switch between the full view and the skeletal view, so lexicographers are reluctant to change views.

Final editing of an entry is also awkward because of sluggish performance in maintaining the form hierarchy as text fields grow larger. Until we tune the performance, we won't be able to judge the dictionary-entry editor as a tool for the entire process of composing an entry.

3.3.2 Numbering the Components of an Entry

In its original design, the dictionary-entry editor managed all sense and homograph numbering automatically, based on the position of the sense or homograph in the entry hierarchy. This ensured that the numbering was always correct and consistent. When the lexicographer wanted to change a sense number, she did so by moving the sense into the proper position in the entry. (In the full view, lexicographers can move fields before, after, or into another field.)

The lexicographers found this design awkward because it required a number of mouse and menu operations to move a sense; most of the time they have their hands on the keyboard, so using the mouse is slow, and the awkwardness was compounded by slow response times. Also, it was frequently the case that they couldn't see both the original location of the sense to be moved and the location where they wanted to move it. They had to spend time and attention navigating the hierarchy when they knew in principle where they wanted the sense to go.

We changed the way numbering worked so that it was always the responsibility of the lexicographer to manage the sense numbers. At that point in the evolution of the dictionary-entry editor, the lexicographer could move a sense only by assigning it the desired sense number. The dictionary-entry editor would then sort the entry to reflect the assigned sense numbers. The entry editor also checked that the lexicographer had assigned values that were internally consistent, that is, that she hadn't assigned the same sense number to two senses and that the values of the sense-number fields were indeed valid sense numbers.

This was an improvement for the lexicographers, but managing all the numbers proved tedious, particularly for entries with many senses. Adding a new sense in the middle meant renumbering all the senses that followed.

To simplify such renumbering, we added a new command which automatically renumbers entries. It assumes that the current order and nesting level is correct and assigns new numbers in increasing order. The lexicographers can thus type in numbers themselves to get a rough cut at the numbering and then ask the dictionary-entry editor to renumber when things start to get messy.

3.4 Atlas: Supporting Information

Atlas is a name that no one uses for a number of small programs that provide the lexicographers with additional information. We have learned that each lexicographer has her own way of working and her own idiosyncratic set of tools. No lexicographer uses all the Atlas tools; probably no lexicographer uses none of them. Most of the Atlas tools are low-tech programs without fancy graphical interfaces.

“tally” reports lexicographic progress on the Hector project—how many words have been tagged in the corpus, how many entries have been written, which lexicographers have been working on which entries, how many senses each entry has, and how many target words and wave-2 words have been completed. For instance:

```
> tally
Number of tokens tagged: 81124
   Which is 00.4% of the 17301331 tokens in the total corpus
and roughly 22.1% of the   366670 target tokens in the corpus

TARGET TOTAL: 133 (23.3% of the 570 target entries)
 WAVE2 TOTAL:   7  (2.0% of the 355 wave2 entries)
 OTHER TOTAL:  24
```

“stats” tells about occurrences of words in the corpus—occurrences, distribution peaks (documents in which the word occurs more often than usual), wordclass information, and related words. For instance:

```
> stats -w grace
grace     noun Ad 463 HM 445  (Grace GRACE)
grace     verb Ad  14 HM  24  (Grace)
===
graced    verb Ad  33 HM  33
===
graces    noun Ad  25 HM  22  (Graces)
graces    verb Ad   0 HM   3
===
gracing   verb Ad   4 HM   4
```

The first line is the invocation of the program (give me stats about wordclasses on the word “grace”). The second line says that “grace” was identified as a noun 463 times by Adam (Ad) and 445 times by the Houghton Mifflin parser (HM) and that the case-variants “Grace” and “GRACE” occurred and are being lumped in with “grace.” And so on. The lexicographers use “stats” to get an initial fix on a word.

“coll” tells about the significant collocates of a word in the Hector corpus, case-free or case-significant, using Mutual Information and t-score, both of them standard statistical tools, as measures of significance. For instance:

```
> coll -cs shorter
shorter
CASE-SENSITIVE
  a+b      a       b      MI    t
    3     271     241    7.66  1.72  shorter tours
    3     271     248    7.62  1.71  shorter varieties
   32     271    4431    6.88  5.56  shorter working
    3     271     463    6.72  1.70  shorter periods
   19     271    3579    6.43  4.26  shorter hours ...
CASE-SENSITIVE
  a+b      a       b      MI    t
    7     364     271    8.29  2.63  inches shorter
    5    2316     271    5.13  2.11  claim shorter
```

```
  6    3232      271    4.92  2.29  longer shorter
 25   15713      271    4.69  4.63  much shorter
  3    2437      271    4.32  1.57  union shorter ...
```

The first line invokes the program (tell me about singificant collocates of "shorter" case-sensitive, that is, without the case-variants "Shorter" and "SHORTER"). Collocates with "shorter" on the left are given first, then collocates with "shorter" on the right. There are three instances where "tours" appears to the right of "shorter"; " shorter" appears altogether 271 times, "tours" altogether 241 times. The Mutual Information score for the significance of the collocation is 7.66, the t-score is 1.72. And so on.

Unlike the corpus viewer, "coll" gives the lexicographer no control over the position of the collocate; it must occur within -5,+5 of the search word. The lexicographers have been asking that we revise "coll" to use lemmas rather than wordforms, and it's on our list.

"beth" tells about Beth Levin's verb patterns, using information she kindly sent to us in advance of the publication of her forthcoming book by the University of Chicago Press. For instance:

```
> beth barter
barter 5.7
5.7        EXCHANGE

vtr             to   VERB           <A>        for  <B>
EXCHANGE        to   exchange       the dress  for  the skirt
SUBSTITUTE      to   substitute     the cup    for  the glass
```

Here the program invocation asks for information on the verb "barter." The response says that "barter" fits pattern 5.7, transitive verbs of exchange. Their pattern is "to VERB A for B," and some examples are the verb "exchange," as in "to exchange the dress for the skirt," and "substitute," as in "to substitute the cup for the glass."

"corpusdoc" gives information about documents in the corpus: their source, authorship, length, and so on. For instance:

```
> corpusdoc Militr
        code: Militr
       title: Military Illustrated: Past and Present
     comment: monthly magazine on military events, uniforms and
              artefacts, March
        date: 1991
      author: unknown
         age: unknown
    authmode: corporate
         sex: unknown
 nationality: unknown
    domicile: unknown
      compos: composite
   publisher: Military Illustrated Ltd
       place: London, UK
       genre: written; published; periodicals; magazines
     samplen: 25265
```

"checkentry" checks some of the more technical fields of an entry (subject, register, and grammar) for conformity to a policy document about what those fields should contain. We've just discovered that through an error on our part the program hasn't been running since early in May, and none of the lexicographers seems to have complained.

"printentry" prints out a paper copy of an entry using typography that suggests the appearance of the final printed copy of the dictionary, not unlike the simulated-print view in the dictionary-entry editor. The lexicographers complain piteously when it malfunctions, so we can tell that it gets lots of use.

And then there are shell commands to put up windows containing simulated-print views of entries from various Oxford reference works such as the Oxford Dictionary of Quotations and the new edition, still in preparation, of the Oxford Shorter English Dictionary. We use Pat and modest home-made front ends in these commands.

3.5 Gritty Details

The corpus viewer and the dictionary-entry editor are written in Modula-3 [4]. Their user-interface code uses the X Window System [5]. It is built on top of SRC's FormsVBT library, which is built in turn on top of SRC's Trestle and VBTkit.[6] The corpus viewer consists of approximately 15,000 lines of source code, the dictionary-entry editor of approximately 25,000. Because they are built from standard user-interface libraries, copying text between one windowed application and another is straightforward. The lexicographers can, for instance, copy examples from the corpus viewer into the dictionary-entry editor.

The dictionary-entry editor produces entry files that are valid SGML-marked text files. The entry editor ensures that lexicographers produce only valid entry files by managing the structure of the entry itself, and by checking the text that has been entered to ensure it contains nothing that might be mistaken for SGML marking.

We use a specification file to describe the structure and elements of an entry. This gives us a central location for modifying this structure and lets us separate the details of the structure from the rest of the dictionary-entry editor, which can manipulate any such structure. The lexicographer can add new information only in ways that are consistent with the spec file. For instance, a field can be moved only to a location that is consistent with the structure described in the specification.

When the lexicographers want to change the entry structure for the dictionary, it is relatively straightforward to produce a version of the dictionary-entry editor that understands the new structure. The hard problem is modifying existing entries so they conform to the new structure. For some changes, such as adding a new field, no changes to the existing entries are needed. However, when the entry structure is changed so that previously legal entries become illegal, it is necessary to convert illegal entries into legal ones. Since this can be a difficult task, Hector has grown increasingly conservative about making changes to the entry structure that are not upwardly compatible.

The dictionary-entry editor manages the storage and retrieval of the entries, so the lexicographer calls up an entry based on the value of the headword. All senses and homographs of the same headword go into the same entry file. Hence, an entry may contain information that is transformed into several dictionary entries, depending on the dictionary style guidelines.

The dictionary-entry editor maintains a history of past versions of entries with RCS, a Unix version control program. It is possible to retrieve any version of an entry, although there is not currently a convenient user interface to this facility. The existence of the previous versions has proved invaluable to the project a number of times, both for studying the evolution of an entry and for recovering from both human and program errors.

We hoped to use the sense uids to enable entries to contain reliable cross-references to one another. We entered the uids and the cross-reference representation of a sense (e.g. the uid 774662 and the representation "bear 1.1a") into a database and encoded the cross-reference as the uid of the sense. However, this mechanism proved clumsy to use, since it required that the entry being referred to not only exist but be loaded into an

[6] VBTkit, Trestle, and Modula-3 are available via anonymous FTP on gatekeeper.dec.com.

entry-editor window in order to establish a cross-reference. So uids have not been used for cross-reference in the pilot project.

4 What Have We Learned So Far?

4.1 Natural Language Is Hard

It's probably not an exaggeration to say that every seemingly reasonable assumption we made about natural language turned out to be inadequate. Even Houghton Mifflin, which had a lot more experience than we did with real-world language, didn't foresee some of the situations we encountered. What's a reasonable limit for sentence size? 256 words seemed bounteous—until we started processing contracts.

```
Party of the first part undertakes: not to incur any
liability on behalf of party of the second part or in any
way pledge or purport to pledge party of the second part's
credit or purport to make any contract binding upon party
of the second part; to involve party of the second part in
any important contract negotiations including but not
restricted to international sales contracts reaching beyond
the Agreed Territories and sales contracts ...
```

Etc. 256 words come and go and the sentence continues, unstinted. How many procedures should it take to calculate noun plurals? A dozen? 25? So far we have 72. What's a plausible specification for a word? It's hard to come up with a specification that's going to stand up to words like "Aah!s" or "county(ies)."

In Adam, the core of the algorithm for identifying wordclasses—the part that deals with the search-space of probabilities—consists of about 50 lines of code. But it is surrounded by over 4,000 lines of code to handle real-text problems.

We learned that we had to be prepared to deal with sentences like this:

```
They come in purple, ref JA8698, age 3-4, #6.99; turquoise, age
7-8, ref JA8701, #7.99.
```

That's prose—not an entry in a table or chart—but it's in an advertisement, which has its own set of conventions for language use.

Even ordinary prose can temporarily switch gears. Street addresses in the middle of ordinary sentences use proper names and punctuation in a way that other contexts do not. Every subject area has its own vocabulary, of course, but it may have its own syntactic rules as well:

```
Tony Simmons set a world age 16 best at Brache in 1965, with
30:16 for six miles.
```

Sometimes the complexity of natural language fuddled us. Take for instance the problem of contractions. The usual way of assigning wordclasses to contractions is to assign them to the separate components. "I'm," for instance, is pronoun + aux or pronoun + be.

Early in our work with Adam, we noticed that while the LOB corpus contains all the components for contractions, it does not contain examples of all the possible combinations, even for the smallest closed sets such as personal pronouns, undoubtedly because the LOB texts were more formal than the bulk of ours and contained fewer examples of dialogue. The Hector corpus contains many examples of contractions from the larger closed sets ("there'll," "what'll," "who'll"), and a small number of completely open-ended contractions:

```
My granddaughter'll be here in a few minutes ...

If it gets any colder, this stuff'll turn to ice.

If we send the boy, Nick'll feel responsible for him.
```

The corpus also contains an example of a double contraction:

```
You shouldn't've done that.
```

For some reason we went from this observation to the conclusion that we should fly in the face of common wisdom and expand the wordclass list to cover contractions and other special cases: "I'm" for us is not pronoun + aux or pronoun + be but rather the special wordclass pronounAux or pronounBe. In retrospect, that was not a wise decision, since the total number of wordclasses in Adam now reaches 313. We added complexity without reaping any reward for it.

We don't have any new insights into dealing with the difficulty of natural language, but like others who have gone before us we treasure certain gems we encountered in our work:

```
Late gonzoid Detroit/NYC journo Lester Bangs has his memory
enshrined with ``One Horse Down'' and its dub partner --
3D colour tracks, crisp production.
```

4.2 User-Interface Design Is Hard

One of our problems in user-interface design was that the lexicographers wanted to keep everything as fluid and malleable as possible, while we wanted to make everything crisp. This tension occurred at every level, from the composition of the corpus to the contents of the register and subject fields in an entry. We had endless discussions of the kind where we asked, "Is this A or B?" and the lexicographers answered, "Sometimes it's A, sometimes it's B, usually it's a combination, I prefer to call it C –" at which point another lexicographer would break in to say, "No, it's not C, it's a combination of B and D." Some of the difficulty was just a matter of learning to work together. Some of it was that the lexicographers didn't understand the power that computers offer those who are willing to make simplifications. Some of it was that they didn't think the power was worth the simplifications. We would imagine that any project that tries to build tools for sophisticated workers in the intellectual trades will encounter such difficulties.

A more serious problem was that the building blocks for our user interfaces and our own facility in putting them together simply aren't up to the sort of complexity the lexicographers handle daily. The wordclasses are a good case in point. Internally, Adam and the Houghton Mifflin parser use very specific wordclasses, e.g. "capitalized plural common noun with word-initial capital." The Houghton Mifflin parser has 171 such wordclasses; Adam has 313. We made a few unsuccessful attempts to design a user interface that would make all of these tags available to the lexicographers. There is some structure to the two tagsets, and they are related—they're both derivatives of the Brown corpus set. There's no question that the lexicographers can handle 171 wordclasses, 313 wordclasses, any number of wordclasses we want to throw at them. But we couldn't figure out how to present the choices on-line in a way that fit on the screen, didn't degrade the rest of the system, and matched the fluid and changing models of wordclass hierarchy that the lexicographers have in their heads. We probably should have recruited an expert in user-interface design to help us with the project.

4.3 Speed Is Functionality

Speed is functionality; when tools are slow, the lexicographers don't use them. Performance always makes a difference, but we couldn't tell where it was really important until we understood the tasks that the lexicographers do.

We knew that a major challenge in Hector would be manipulating large amounts of data quickly. Our initial efforts were focussed on searching the corpus quickly. We had no performance problems in searching itself, but ran into problems in several other areas, notably in user-interface functionality.

For instance, switching between the full view and the skeletal view of an entry in the dictionary-entry editor is slow, so the lexicographers limit themselves to one view, even if it is inappropriate for the task at hand.

Similarly, displaying a concordance was quite slow in early versions of the corpus viewer. As a result, the lexicographers would make one broad search for a word and then work with the resulting concordance as if it were static. They wouldn't test hypotheses about collocates and senses, because it would take too much time and they would lose the results of their first search.

Poor performance is especially damaging when it interrupts what feels like a single action. When a lexicographer reaches a decision after hard thought, she wants to be able to act on it without losing the thought. For example, when a lexicographer determines that she needs a new sense to tag a corpus line, she wants to be able to create the sense and tag the line without losing context or having to relocate the line. Delays in the communication between the dictionary-entry editor and the corpus viewer about active senses make it irritatingly slow to tag that first line after creating the sense. Or again, when the dictionary-entry editor needs to expand the size of a field because the contents have overflowed, it takes several seconds before the lexicographer can safely resume typing.

We have improved the functionality of sense-tagging in the corpus viewer to make it faster to assign sense-tags once the sense divisions have been determined. Shortcuts and batch sense-tagging render the actual tagging of corpus words much faster once the hard work of sense division has been done.

4.4 Data Integrity

The lexicographers in the Hector project create two important sets of data: dictionary entries and corpus sense-tag assignments. Because the human effort and expertise involved is hard to come by, this data is one of the most valuable results of the project. Hence, it is particularly demoralizing when it is lost or has to be regenerated.

We made some effort in designing our tools to protect ourselves against data loss. We keep all the versions of an entry file, and we log all changes to the sense-tag database. But total data integrity was not one of our original design goals.

Data was lost primarily because the dictionary-entry editor or the corpus viewer crashed, losing sense-tags or entry edits that had not been committed. Naively, we had believed we could build programs that would not crash. Modula-3 provides good type checking and exception handling, and we tried hard to make the programs robust.

We also lost data due to program errors. For instance, errors in the code for storing sense-tags caused assignments to be lost.

Since we relied on our design efforts to build robust programs, we didn't build mechanisms to recover work in progress when the programs inevitably crashed. In retrospect, this was a bad decision.

4.5 And in a Cheerier Vein ...

It wasn't our aim in the Hector project to see what a group of amateurs would do with a set of tools for corpus lexicography, but our lab includes a good many tinkerers and casual philologists, so we weren't surprised to find them playing with the corpus viewer. Several people use it regularly to check their intuitions about words and idioms and to supplement information they find in a dictionary. For example, one lab member recently questioned the apparently inconsistent use of the words "gantlet" and "gauntlet" in the New York Times. He searched for information in a dictionary but also consulted the corpus viewer.

While the corpus has some usefulness for decoding information, people go to it more often for encoding information. One colleague invoked the corpus for evidence on whether "noir" is now an ordinary English word or whether it should still be italicized. Another recently requested help in deciding which of three possible phrases he should use in a particular mathematical context, and again, corpus evidence was cited in the ensuing discussion.

Publishers take note. A matching dictionary and corpus set, bound in the electronic equivalent of morocco, might be *the* Christmas gift for 1996.

Acknowledgements

Thanks to our friends at Houghton Mifflin: Win Carus, Kathy Good, and Jeff Hopkins.

To Ken Church, for his algorithm and his encouragement.

To the lexicographers from Oxford: Sue Atkins, Katherine Barber, Peter Gilliver, Patrick Hanks, Helen Liebeck, Rosamund Moon, and Bill Trumble.

To our colleagues on the computing side at Oxford: Jeremy Clear, James Howes, Chris Rust.

To our colleagues at SRC: Ken Beckman, Judith Blount, Marc H. Brown, Mike Burrows, John DeTreville, Steve Glassman, Jim Horning, Catherine Kaercher, Bill Kalsow, Eric Muller, Lyle Ramshaw, Richard Schedler, Julie Swanson, and Ted Wobber.

To our extraordinary managers: Tim Benbow and Bob Taylor.

To our children: Margaret Brown, Naomi Glassman, Elizabeth Reid, and Vanessa Reid for setting aside their selfish personal needs for the greater good. We hope that Ethan Glassman will follow in their footsteps.

References

[1] Ken Church. A stochastic parts program and noun phrase parser for unrestricted text. In *Proceedings of the Second Conference on Applied Natural Language Processing,* Austin, Texas, 1988.

[2] Patrick Hanks. Evidence and intuition in lexicography. In Jerzy Tomaszczyk and Barbara Lewandowska-Tomaszczyk, editors, *Meaning and Lexicography.* John Benhamins Publishing Company, Amsterdam/Philadelphia, 1990.

[3] Stig Johansson and Knut Hofland. *Frequency Analysis of English Vocabulary and Grammar.* Oxford University Press, 1989.

[4] Greg Nelson, editor. *Systems Programming with Modula-3.* Prentice Hall, 1991.

[5] Robert W. Scheifler, James Gettys, and Ron Newman. *X Window System: C Library and Protocol Reference.* Digital Press, 1988.

[6] J. M. Sinclair, editor. *Looking Up: An account of the COBUILD Project in lexical computing*. Collins, 1987.

A Appendix: SGML Constructions

We call SGML bracketing constructions “tags” and write them with angle brackets:

```
<date>..</date>
```

We call SGML typographical constructions “sorts” and write them with an opening ampersand and closing dot:

```
&alpha.
```

We call more complicated constructions “sundries” and write them with curlies:

```
{typo bad="asgood",good="as good"}
```

Here’s the list:

<address>..</address> of a letter
&alpha.
&acute.
&and. ampersand
<author>..</author> name of the author appearing at the head of a document or a section of a document
&back. diacritical mark in, for instance, Arabic names
<brochure>..</brochure>
<caption>..</caption> on a photograph or illustration
&cedilla.
&cent. cent-sign
&circ. circumflex
<close>..</close> of a letter
<column>..</column> same as <story> (a remnant of overambition)
&copyright. copyright sign, not the word “copyright”
&dash.
<date>..</date> dateline of a news story, date of a letter
&degree.
&ellip.
&ft. feet written as a stroke or a single quote
<fu>..</fu> footnote
&grave.
<guess>..</guess> in transcribed speech
&hacek.
<hdl>..</hdl> headline (not necessarily newspaper—any title line)
<ignore>..</ignore> attributes: foreign (with language noted), verse, deadGuys, table
{inaudible}
&ins. inches written as two strokes or a double quote
<lecture>..</lecture>
<letter>..</letter>
<meeting>..</meeting>
<memo>..</memo>
&num.

&oob. o oblique
&pi.
<ps>..</ps> postscript of a letter
&ring.
&rule.
<sal>..</sal> salutation of a letter
<sig>..</sig> signature, byline
<speech>..</speech> transcribed speech
&sqrt.
<story>..</story> in a newspaper
&stroke.
<T>..</T> entire document
&theta.
&tilde.
&times.
{typo bad="string",good="string"}
&umlaut.

B Appendix: The Target Words

absolute
absorb
acquisition
adam
adequate
admire
advocate
agenda
agriculture
airline
albert
alcohol
allegation
alliance
allowance
alongside
alter
altogether
amateur
amaze
ambition
ambulance
analyst
angle
anniversary
anxiety
anxious
dancer
dawn
dealer
dean
declaration
defendant
deficit
definition
delegate
delivery
departure
deposit
depress
depth
derby
derive
description
desert
designer
desperate
destruction
detective
determination
device
devote
diet
disable
intense
interim
interior
interval
invasion
invest
invitation
involvement
israeli
jewish
judgement
kingdom
knee
laboratory
layer
leap
lecture
leicester
lend
liability
liberation
liberty
licence
literary
literature
lobby
loose
resistance
restrict
restriction
retail
retirement
revenue
reverse
revolutionary
rider
riot
roman
rome
rough
routine
ruin
rumour
runner
rural
sack
sacrifice
sake
salary
salt
sample
sanction
satellite
scientific

anywhere
apparent
appointment
appreciate
approval
arab
architect
architecture
asian
assess
assumption
attendance
attraction
auction
awful
ballet
ballot
banker
barely
bargain
bathroom
beating
behalf
behave
bench
beneath
besides
bitter
blind
boil
bone
boom
bore
boss
bother
boundary
brazil
breach
bread
breathe
breed
brick
brush
burden
bury
businessman
butter
discipline
discount
discovery
dish
distant
distinction
distinguish
distribution
disturb
dividend
divorce
draft
drag
drain
drift
duck
duke
dutch
echo
economist
edition
efficiency
efficient
eighty
embarrass
embassy
emotion
emotional
empire
enemy
enhance
entertain
entertainment
enthusiasm
entrance
equity
equivalent
establishment
everywhere
evil
exception
excess
exclude
expansion
expectation
exploit
explore
lorry
luck
luxury
maintenance
maker
margin
mayor
meat
mechanism
medicine
membership
mental
merchant
merger
merit
minority
mixture
modest
monthly
moreover
musician
muslim
mystery
nato
necessarily
nervous
notion
objection
objective
obligation
oblige
observer
occasional
olympic
operator
opponent
orange
origin
ourselves
outcome
outline
output
outstanding
overseas
overwhelm
pact
palestinian
scream
script
seize
sensitive
sequence
seventy
severe
shed
sheep
shell
shelter
shirt
shortly
sick
significance
silent
sixth
sixty
slight
smooth
snow
socialism
solve
sophisticated
speaker
spectacular
speculation
spite
stable
stair
stamp
statistic
steady
steam
steer
sterling
strengthen
striker
stroke
submit
subsequent
subsidiary
substitute
sudden
sufficient
sugar
suitable

cake
calculate
calm
cancel
capture
carbon
carpet
category
cease
cell
characteristic
charlie
charter
cheer
cheese
chicken
childhood
cinema
circuit
civilian
classical
climate
coalition
collective
colonel
column
comic
commander
commissioner
commonwealth
comparison
compensation
competitive
competitor
composer
comprehensive
compromise
concede
concentration
concrete
condemn
confine
confusion
connect
constant
constituency
constitution
explosion
extension
extensive
extreme
false
fancy
fantasy
fare
fascinate
fate
federal
federation
fence
flag
flavour
fleet
float
formula
fortune
forty
forum
fulfil
fundamental
gear
generous
genuine
gesture
global
gloucester
govern
governor
grace
grade
graduate
greek
greet
grip
guerrilla
gulf
habit
halt
healthy
height
hint
historic
historical
holder
parallel
partnership
passage
passport
patten
peak
permanent
personality
phrase
pile
pilot
pipe
platform
plead
pleasant
pledge
plot
poetry
pole
portrait
pose
possess
possession
poverty
practise
pre-tax
precisely
premise
premium
preparation
preserve
presumably
pride
priest
princess
principal
privilege
proceeding
profession
prompt
proof
prosecution
proud
province
provoke
publicity
pure
suite
supreme
surplus
surrey
survival
suspend
suspicion
sustain
swallow
sweat
symbol
sympathy
temporary
tempt
tenant
tension
territory
terrorist
text
thrust
tonne
tool
tragedy
trail
transaction
transform
traveller
treasury
treaty
trend
truly
tune
tunnel
turnover
twelve
twin
twist
ultimate
uncertainty
underground
undermine
unemployment
unfortunately
uniform
unique
unity
unknown

constitutional
construct
consult
consultant
context
contrary
controversial
convention
conventional
conversion
conviction
core
corporate
corporation
counter
craft
cream
creation
crucial
cultural
currency
curtain
hollywood
horror
humour
identity
illegal
illness
illustrate
imagination
implement
implication
imply
impress
impressive
incorporate
indication
inevitable
inner
innocent
install
instruction
intellectual
intelligence
pursuit
raid
rally
rape
rapid
rarely
realize
reception
reckon
recognition
recovery
recruit
referee
regret
reinforce
relevant
religion
renew
replacement
reporter
resignation
resist
urban
vegetable
victorian
villa
violent
voluntary
volunteer
voter
weak
wealth
westminster
whereas
widely
widespread
widow
wooden
wound
yacht
yield
zone

C Appendix: The Wave-2 Words

abandon
abuse
accident
accountant
adjust
administer
adverse
advertise
airport
ally
altar
applicant
appropriate
assault
assistant
asylum
atmosphere
atomic
avenue
background
bake
balcony
diamond
directive
disability
disappear
discharge
disclosure
disgrace
dismay
disposal
disruption
distance
distress
disturbance
dominate
donation
donor
drainage
drill
edit
editor
eliminate
employ
kettle
kick
kidney
kindness
kinship
kiss
kitchen
lager
landlord
laugh
laughter
lava
layout
library
license
livestock
location
lock
lung
maggot
magistrate
magnetic
recommend
redundancy
regulation
relate
reluctant
remind
reminder
restless
retire
reward
roar
rotten
rubble
rush
sandwich
satisfactory
sausage
scenery
scramble
scrap
screen
sculpture

barrister	employee	male	search
beam	employer	manufacture	secret
bike	employment	manufacturer	sensible
biscuit	endless	marriage	shadow
bishop	engagement	masterpiece	shock
bleed	enthusiastic	mathematical	sickness
bomber	erupt	medal	skirt
bombing	ethics	medallist	sleeve
booking	exam	methane	slip
borrow	examine	microwave	smell
bowl	examination	mild	snack
brain	exhibit	mill	sofa
bright	exhibition	mineral	specialise
brilliant	explanation	mirror	specialist
burglary	expression	missile	spiritual
cabaret	extract	murmur	statistical
cabbage	extradition	mushroom	stimulation
candle	extraordinary	nationalist	stockbroker
carrier	ferry	negotiator	storm
cater	file	notice	strand
celebrate	fill	novice	stream
cemetery	filmmaker	obey	string
cereal	fixture	onion	subscription
characterise	flood	options	substance
cheap	flower	ordination	suburb
cheek	focus	overlook	succeed
chip	footstep	overthrow	sulphur
cholesterol	fragment	packet	supervision
chub	franchise	pack	swing
circulate	frontier	package	tackle
citizenship	frost	participation	takeover
classroom	frustration	passion	temperament
clock	furnish	pasta	textile
coat	geographical	patch	thigh
collector	giant	percent	thin
colt	glaze	percentage	thriller
combine	gravel	phenomenon	toilet
comfort	grid	phone	tomato
companion	grill	photograph	tonight
compile	grind	photographer	torture
comply	guess	pint	tournament
compound	header	plaster	transition
cone	heavyweight	poison	transplant
confess	hedge	potato	treasure
confident	helicopter	practitioner	triangle
confuse	heroin	predict	triple
congestion	honesty	prediction	turkey

consecutive	horizon	preference	twenty
constable	hostage	prevail	tyre
consume	hurdle	proceed	conditional
contest	identify	proceedings	undergo
contrast	immigrant	processor	unsatisfactory
convert	impression	progress	welcome
convoy	incident	progressive	vacant
correct	indefinite	prosecute	venue
correspond	industrialist	publication	vessel
counsel	initiative	punch	visa
cousin	injunction	purse	warrant
crack	institution	qualification	weakness
crash	institutional	quote	weed
criticise	intrinsic	quota	width
crush	inventory	rabbit	wild
cycle	irony	radical	willow
dangerous	responsible	radically	woke
deed	jacket	rank	wonderful
defect	journey	reassure	workshop
deterioration	junction	receipt	