LEXICALLY-GENERATED SUBJECT HIERARCHIES FOR BROWSING LARGE COLLECTIONS

Craig G. Nevill-Manning[*], Ian H. Witten[†] and Gordon W. Paynter[†]

[*] Department of Computer Science
Rutgers, the State University of New Jersey
Piscataway, NJ 08854, USA
nevill@cs.rutgers.edu

[†] Department of Computer Science
University of Waikato,
Hamilton, New Zealand.
{ihw, gwp}@cs.waikato.ac.nz

Developing intuition for the content of a digital collection is difficult. Hierarchies of subject terms allow users to explore the space of topics that a collection covers, to form and specialize useful query terms, and to directly identify interesting documents. We describe two interfaces for navigating such hierarchies, and present a technique for inferring hierarchies automatically from large corpora. We also discuss scalability issues for the techniques involved, and our solutions to these problems.

1 Introduction

How can you browse a digital library? Its appearance gives little clue as to what lies inside. At least physical collections occupy physical space, present a physical appearance, and exhibit tangible physical organization. When standing on the threshold of a large library one gains a sense of presence and permanence that reflects the care taken in building and maintaining the collection inside. No-one could confuse it with a dung-heap! Yet in the digital world the difference is not so palpable. What lies beyond that front page—a carefully-selected collection or a morass of worthless ephemera?—half a dozen documents or many millions? Have you no choice but to believe the home-page blurb—to judge thousands of books by a single cover? How can you *experience* a digital library, as you would stroll through the stacks of a physical one?

The opacity of digital collections is enormously frustrating. Precisely because the collection is available digitally it should be amenable to automatic indexing, summarization, and visualization techniques, which ought to make browsing particularly easy and meaningful. Studies of browsing have shown that it is a rich and fundamental human information behavior, a multifaceted and multidimensional activity (Chang and Rice, 1993). Research is progressing on an arsenal of techniques that head in quite different directions: physical-space metaphors like virtual-reality libraries (Hearst & Karadi, 1997), navigation metaphors like hypertext, information-space ones like topic clustering and visualization (Cutting, *et al.*, 1992), people-oriented approaches based on formally-defined roles like network librarians or informal ones like intellectual encounter groups, market-oriented schemes like negotiation between agents, ethological ones like foraging, and agricultural ones like harvesting and berry-picking.

Despite the interest and activity in these exciting new possibilities, it is direct, explicit, searching that dominates the digital library scene today—for browsing as for everything else. Full-text retrieval makes it possible, in principle, to locate relevant information very efficiently in a huge collection. Retrieval of matching documents involves both *precision* (not returning irrelevant documents) and *recall* (not overlooking relevant documents) (Salton, 1989), and there has been much research on how to maximize precision and recall given a particular query (Harman, 1992–96). In practice, the question posed at the beginning of this article tends to be answered by making a selection of queries more or less haphazardly to gain a feeling for what the collection holds.

This paper describes a new way of getting to grips with the content of a collection. The idea is to build a hierarchical subject index automatically from the text of the collection itself. This index is a hierarchical structure of phrases that appear frequently. Presented interactively to the user, it provides a new foundation for browsing. Figure 1 shows an example of the interface, which will be discussed more fully in due course. Briefly, users can select any word from the lexicon of the collection (the word *index* has been selected in the left-hand column), see which phrases it appears in (center column), select one of them (*indexing and retrieval*) and see the larger phrases in which *it* appears (right-hand column). Only part of the window is shown in the figure, and columns continue to the right. This is reminiscent of the permuted title or keyword-in-context (KWIC) indexes of days gone by. However, there are two crucial differences. First, a *hierarchical* structure of phrases is identified. This greatly reduces the size of the index and allows the user to home in on useful information in logarithmic time. Second, the phrases are restricted to those that occur more than a preset number of times—usually twice or more. This shifts attention from individual items towards the content of the collection as a whole.

Phrase browsing allows users to gain a feeling for the kind of topics that are treated in the collection. As a bottom-up, lexical approach, it lies at the opposite end of the spectrum to holistic, semantic, methods like document clustering. Both kinds of technique are important, and future solutions will incorporate a variety of different approaches, offering different things to people with different cognitive styles or different kinds of requirements.

This paper describes the technique: how the index can be browsed, what it feels like to use, and also how the index is created. We have developed an algorithm (called SEQUITUR) that infers a hierarchical structure of phrases from a sequence of discrete symbols. It is very efficient and can easily process large amounts of text (our main example in this paper is based on a corpus of 62 million words). The details of the algorithm are not central to this paper and so are deferred to an appendix. More important, for present purposes, are the kind of phrases that are identified, and how they can be displayed in a browsable form. This paper gives many examples of the phrases generated to convey a feel for this new way of browsing, and describes interfaces for both local X-windows access and remote access over the World Wide Web. Finally, we discuss the distinctive features as well as some shortcomings of this approach to browsing the content of document collections, and review opportunities for future work.

2 Identifying index terms

The SEQUITUR algorithm infers a hierarchical structure of phrases from a sequence of discrete symbols. The technical details of the algorithm are described in the Appendix: here we indicate, through examples, what it can do.

SEQUITUR can operate on any sequence of tokens and infers structure from repeated substrings. When individual letters are used as tokens, it successfully identifies most words, including, in some cases, their morphological structure, and some short phrases. Figure 2 shows parts of three hierarchies inferred from the text of the Bible in English, French, and German. The hierarchies are formed without any knowledge of the preferred structure of words and phrases, but nevertheless capture many meaningful regularities. In Figure 2a, the word *beginning* is split into *begin* and *ning*—a root word and a suffix. Many words and word groups appear as distinct parts in the hierarchy (spaces have been made explicit by replacing them with bullets). The same algorithm produces the French version in Figure 2b, where *commencement* is split in an analogous way to *beginning*—into the root *commence* and the suffix *ment*. Again, words such as *Au*, *Dieu* and *cieux* are distinct units in the hierarchy. The German version in Figure 2c correctly identifies all words, as well as the phrase *die Himmel und die*. In fact, the hierarchy for *the heaven and the* in Figure 2a bears some resemblance to the German equivalent.

For the subject hierarchies that are the topic of this paper, words rather than individual letters are used as tokens. Figure 3 shows a small example of such a hierarchy. The graphical version in Figure 3a mirrors the structure of Figure 2, whereas Figure 3b shows its native representation as a grammar. Each branch in the hierarchy corresponds to a rule in the grammar. The rules that

SEQUITUR creates will be illustrated by a grammar constructed from a large body of 7000 computer science technical reports, part of the 1.9 Gb corpus comprising the Computer Science Technical Report collection of the New Zealand Digital Library (Witten *et al.*, 1996). Pertinent details of the collection are summarized in Table 1. The reports were presented as a sequence of words, and all words were mapped, somewhat arbitrarily, to lower case before processing. This produced a vocabulary of 270,000 words, nearly half of which occur only once in the corpus.

In any hierarchy produced by SEQUITUR, the rule headed by the start symbol *S* expands to reproduce the entire sequence (of 62 million words, in this case). In Figure 3b, for example, the ellipses before and after the non-terminal *A* in rule *S* indicate the rest of the document surrounding the phrase, which we have not bothered to write out (the ellipses expand to 62 million words less five). This initial rule has rather a different character to the others. All other rules express regularities in the original sequence, in that their contents must occur at least twice. The top-level rule receives the leftovers; the unique sequences that do not recur. In the grammar produced for the sample of Table 1, the initial rule contained 16 million symbols—about a quarter of the number of words in the original sequence. There were 2.6 million other rules, that is, phrases; having an average of only 3.3 symbols on their right-hand side. Because the grammar is hierarchical, some of these symbols are non-terminals that refer to other grammar rules. On average, a grammar rule expands to 9.6 words if all the non-terminals in it are expanded recursively.

Figure 4a shows the first few rules that involve the word *grammar,* which occurs a total of 5109 times in the text. The rules appear in decreasing order of occurrence frequency, and the number of times each occurs is given on the left. Thus the most frequent phrase is *the grammar* (which occurs 426 times in the text), followed by *attribute grammar* and *a grammar*. The first and third phrases are of little interest. However the second, *attribute grammar*, is just the kind of entry that one would like to see in a subject index.

Words that occur very frequently obscure information. Tokens like *a* and *the* degrade the index because SEQUITUR often uses them to form rules, but they add little meaning to the phrase. It is of little interest that the most common phrase involving the word grammar is *the grammar*. Consequently, in Figure 4b we have suppressed rules that differ from their parent rule (or parent word) merely by the addition of a common word.[1] In practice, somewhat over half of the rules formed by SEQUITUR are spurious ones that do not add anything meaningful because they differ from their parent rule only by the inclusion of words that appear in the hundred most common words.

As Figure 4b shows, suppressing uninteresting rules generates a much more worthwhile set of phrases. In fact, half of the twelve phrases visible here are definitely index terms for this collection—*attribute grammar*, *context free grammar*, *montague grammar, bison grammar*, *categorical grammar*, and *reber grammar*. Of the others, *grammar rules*, *grammar rule*, *grammar inference*, and perhaps *interpretation grammar* are arguably index terms as well. The first entry in Figure 4b corresponds to the grammar rule

$$A \rightarrow \textit{attribute grammar}$$

It is unfortunate that the same term appears again a little further down the list, in the phrase *an attribute grammar*. This is because a rule

$$B \rightarrow \textit{an attribute}$$

was created prior to rule *A* above, and this caused a new rule

$$C \rightarrow B \textit{ grammar}$$

[1] In fact, these rules are not actually suppressed, but expanded one further level. This is explained in more detail later.

to be formed instead of

$D \rightarrow$ *an A*

—which would expand to the same thing. This last would have been vastly preferable because it would have increased the strength of rule *A*. However, SEQUITUR is a greedy algorithm: once a rule is formed, it is never reconsidered. A more sophisticated approach would have such serious consequences in terms of computational complexity that it would be completely impossible to form grammars of anything like this size. In fact, the problem of finding a set of phrases that produces the smallest phrase-structure representation is known to be NP-complete (Storer and Szymanski, 1982).

Figure 4c shows expansions of the second phrase of Figure 4b, *context free grammar*. This section of the hierarchy corresponds to rules

$E \rightarrow$ *context free grammar*
$F \rightarrow$ *probabilistic E*
$G \rightarrow$ *reduced E*
$H \rightarrow$ *E cfg*
$I \rightarrow$ *E recognition*
etc.

The phrases *probabilistic context free grammar*, *reduced context free grammar*, and *context free grammar recognition* are valuable index terms; the remaining phrases in Figure 4c are not. The phrase *context free grammar cfg* serves to introduce an abbreviation. The occurrence of *a probabilistic context free grammar* indicates that the phrase *a probabilistic* was created before *context free grammar*, another manifestation of the suboptimality of the rule formation procedure. Below this point the phrases that are encountered have only been seen two or three times, so it is not surprising that they do not appear to be useful index terms. However, they are left in the display, because they fulfill another role: they provide access to actual documents in the collection.

There are just nine expansions of *probabilistic context free grammar*, and the seven lines of Figure 4d cover all of them. Now a new phenomenon appears: except for the first two, these phrases do not participate in grammar rules but appear as components of rule *S*, the top-level rule. Recall that this initial rule contains 16 million symbols. Buried somewhere in there is something like

… grammars and show that a F using … definitions and properties a F is just an … work best with current day F pcfg parsers … the highest probability of any F for that … of the text given the F unfortunately …

However, it is likely that many of the terminal symbols shown here are also represented by higher-level rules, and so rule *S* might contain something more like

$S \rightarrow$ *… P and Q F using … R a F is T … U with V F W … the X of any F Y … of Z the F unfortunately …*

with appropriate expansions for the non-terminals *P, Q, R, T, U, V, W, X, Y* and *Z*. Excerpts from the actual technical report text are indicated as such in Figure 4d by the ellipses.

Even at this low level of the hierarchy there are valuable discoveries to be made. The expansion of the second rule of Figure 4d, *probabilistic context free grammar pcfg*, leads to an interesting inference: *pcfg* is being used as a synonym for *probabilistic context-free grammars*. The implication that this acronym is used in place of the phrase is potentially extremely useful: the user may decide to perform another search on the abbreviation, because it is likely that once it has been defined, the acronym alone will be used (in fact it occurs alone 87 times in this collection). It is interesting to consider how this information would have been obscured in a different method of presentation—it is hard to imagine such a discovery taking place in the absence of a phrase-browsing technique, because the co-occurrence of a phrase and an abbreviation usually has no special significance. This phenomenon is not uncommon: in fact it also occurs in the third rule of Figure 4c, *context free grammar cfg*.

3 Interactive browsing

We have built two browsers for this phrase hierarchy, one written in TCL/TK for X-windows, and the other written as a Java applet for Web browsing. Figure 1 illustrates the former; Figure 5 the latter. The screen display in Figure 1 shows a hierarchy based (for variety's sake) on a different corpus, the *Computists' Communique* (www.computists.com). This is an on-line AI research news magazine, operating since 1991, which includes grant and funding opportunities, industry news, Internet and Web information, on-line resources, research discussion lists, software offers, software development resources, and career and entrepreneurial tips. The examples in Figures 4–8 are taken from the corpus described in Table 1.

The X-windows browser

The X-windows implementation displays the full vocabulary of the collection (at the left of Figure 1). This means that a browsing session can take place entirely with a mouse. Every path through the hierarchy leads to a phrase that is guaranteed to occur in the collection. In practice, however, the vocabulary is far too large to permit convenient access purely by scrolling. Keystrokes serve to scroll the list to the appropriate place, so that a prefix can be entered on the keyboard to evoke the appropriate range of words on the screen.

In Figure 1 the user has selected *index* from the vocabulary and the phrases it appears in are listed in the next column to the right. For example, *index htm* appears six times. Note that this particular phrase appears as an artifact of word parsing: it emanates from the filename *index.htm*—as of course does *index html*, further down the list. It is encouraging that these junk entries consume far less space in the list than they would in a conventional query for the term *index*. Each phrase can be selected and expanded in turn. The user has selected the phrase *indexing and retrieval,* which also appears six times in the corpus. In this particular case, each of these six phrases occurs exactly once and cannot be expanded any further—in fact they are all flanked by ellipses that would be revealed by scrolling the third column horizontally.

In general, the user can traverse the grammar, extending and hence specializing the query term. Every word is the root of a tree structure whose leaves are the occurrences of that word in the collection. Occurrences in other rules are internal nodes corresponding to phrases that contain the word. Those phrases are themselves used elsewhere in the grammar, either in the top-level rule or in other rules for longer phrases. It is possible to stop at any internal node and use that phrase as a query term, or continue following the tree to a leaf and retrieve the corresponding document.

SEQUITUR treats different words as completely different symbols even though they may be closely related lexically. The interface overcomes this problem by stemming queries and expanding them to include related words from the lexicon. For example, in Figure 1 the user has selected *index,* but the interface displays phrases including *indexed, indexers, indexes,* and *indexing* as well. This process is explained in more detail in Section 4

Another feature concerns words that occur only once. Generally, between 30% and 50% of the words in the vocabulary of any collection appear only once (in Table 1, the proportion is 45%). These words can never generate a phrase hierarchy, and are unlikely to be of interest to the user of a browsing tool whose purpose is to give a feel for the general content of a corpus—even though they would be highly significant if used in any particular query. Any words that appear less than a small, preset number of times are not included in the vocabulary list and are printed in red whenever they appear in the other columns. The user can change the threshold so that there are no rare words, or so that words that occur fewer than a certain number of times are considered rare; this is done interactively using the *Rare Words* menu item. In Figure 1, the only rare word visible is *syntactica,* which appears on the fourth line of the *indexing and retrieval* column: in fact it appears in red although this is not apparent in the Figure. Incidentally this word is a brand name in *Syntactica indexing and retrieval systems*; a possibly interesting feature of the collection that we would surely never have noticed without such a browsing tool.

As noted earlier, common words are used to weed out profitless phrases like *indexing and* by only displaying phrases that differ from their parents by at least one non-common word. When the user performs a search for *index* the phrase *indexing and* is not returned because *and* is not an interesting word. Conversely, the phrase *knowledge index* is displayed because *knowledge* is an interesting word. Phrases that are not interesting are expanded until they subsume at least one further interesting word. For example, when *indexing and* is expanded it generates several interesting phrases, including *indexing and retrieval* and *automatic indexing and fact extraction from*. Common words are identified as those that occur more than a certain number of times—by default, one hundred times. In this interface, common words are shown in gray, and the threshold can be changed interactively using the *Common Words* menu item.

The Web browser

The Web interface shown in Figure 5 provides slightly different interactive facilities from the X-windows version. The word list to the left of Figure 1 would either need a huge vocabulary to be transmitted in advance, or extremely rapid Web access to provide the same effect using incremental transmission: thus we have (reluctantly) dispensed with this very useful facility. However, a way of cutting off low-frequency phrases has been included. Figure 5 is set to show *all phrases*, but the user could have opted instead to hide terminal phrases, or, in addition, non-terminal phrases with frequencies less than a selectable cutoff value. Moreover, the stemming feature has been improved: the way in which it has been improved is described in the next section because its operation is intimately bound up with the system structure.

This section gives several illustrations to convey the feeling of how the system can be used to browse a huge collection of information. Figure 5 shows phrases containing the word *robot* which was entered textually in the *search* box. Just over half of the top twelve terms are quite clearly subject terms: here they include *mobile robot*, *robot motion*, *robot arm*, *robot navigation*, *robot control*, *robot hands*, and *robot vision*. The phrase *a mobile robot* appears because of the above-mentioned problem of suboptimal rule formation. The page contains panels for further expansion—three panels in all, one of which is barely visible in the Figure. In the second one, the term *mobile robot* has been expanded. The panels can be arranged vertically, as shown in the illustrations here, or horizontally in the fashion of Figure 1.

Figure 6 shows the results for *language*. Nine of the twelve items visible are definitely index terms, two of the remainder (*language constructs* and *language design*) are arguably so. Expanding *programming language* leads to the second panel: here again nine entries are definite index terms, two are caused by suboptimal rule formation (*reference manual for the ada programming language* and *a database programming language*), and the remaining one stems from bibliography entries (*conference on programming language*). Further expansion of *programming language design* leads to the phrase *programming language design and implementation*, along with a number of conference titles and several excerpts from actual technical reports (which would have been suppressed if the user had selected the *hide terminal phrases* option).

If the user selects a terminal phrase, the document in which the phrase occurs is displayed in a separate browser window. As it is possible—even likely—that two or more similar phrases are drawn from the same document, we are investigating ways to show when this occurs to prevent the user from selecting several interesting excerpts from the same document. Another way to avoid this problem is to use a search engine with a higher-level phrase. If the corpus has been separately indexed by a compatible full text retrieval search engine, the user can initiate a traditional search for any phrase by clicking on it with the right mouse button. A separate browser window will display the results of the search.

Figure 7 shows the results for the word *digital*, and the expansion of the terms *digital library* and *visible human digital library*. The lack of stemming on the phrases (as opposed to the query) causes problems: *digital library* is separated from *digital libraries*; *digital computer* and *digital computers* are listed separately. Major digital library projects are identified (*Alexandria*, *visible human*, *Illinois*) along with generic kinds of digital library (*national*, *image*, *medical image*)—although again suboptimal phrase identification causes some redundancy.

Although it is not evident from the scroll bars in Figures 4–7, only a very small fraction of the index entries are shown, and one cannot help wondering what happens if you go further down the lists. Figure 8 shows, in the top two panels, the direct continuation of Figure 4b: entries (with stop-words in force) for the word *grammar*. Many viable index terms appear even at these lower levels: *functional grammar*, *prefix grammar*, *english grammar*, *dependency grammar*, *regular grammar*, *generalized phrase structure grammar*, *segmental grammar*, *control grammar*. At this point the phrases that are found appear only a dozen times; and the scroll bar bears witness to the fact that there is a lot further to go! In the bottom panel of Figure 8, about halfway down the list, the terms are less consistently useful. There are still interesting terms such as *fuzzy grammar* and *random grammar,* but there are also lengthy phrases, such as *replay in an attribute grammar framework with annotation is a*, that occur only twice. For this reason, a menu selection is provided to limit the phrases returned to those occurring more than a specific number of times.

Bear in mind that the collection from which this index was formed contains 62 million words, 7000 technical reports. An enormous variety of topics is covered, and it is not surprising that as well as genuine subject terms there are a huge number of less interesting phrases that appear only two or three times each.

4 System structure

The World-Wide Web subject hierarchy browsing system is implemented as a client-server structure, and the implementation poses significant technical challenges because the amounts of information involved are extremely large. Hierarchies generated by SEQUITUR are roughly proportional to the size of the original text—380 Mb for our main example—and it is clearly infeasible to retain the entire hierarchy in main memory for such large corpora. This creates problems for two parts of the system: the formation of the hierarchy, and its traversal at run-time in order to create the dynamic display. The solution to the first problem involves a modification to the SEQUITUR algorithm, which, while simple, is not germane to the present paper. The second problem is intimately connected to the process of browsing, and our solution is described here.

Figure 9 depicts the structure of the distributed, disk-based browsing system. The top three processes are performed once for the collection, and produce all of the files necessary for the browsing system. The input is the text of the collection. The first process, the tokenizer, parses this text into words and forms a lexicon for it. It also produces a file containing the frequencies of each of the lexicon entries, and URLS for each of the documents in the collection. The tokenizer simultaneously produces a stream of numbers—indexes into the lexicon—that represent the text thereafter. These numbers are treated as atomic symbols by SEQUITUR, which forms a hierarchy from the sequences of numbers, as described in the Appendix, and outputs it as a textual grammar.

This grammar is unsuitable for browsing in its textual form for two reasons. First, during browsing it is necessary to find all occurrences of a particular symbol—whether a word or a reference to a rule—in the grammar. Without an index, this requires the entire file to be scanned. Second, finding a particular rule in order to compute its expansion requires a similar scan. In order to make these operations acceptably fast, a representation of the grammar is created on disk where a symbol is represented by a four byte integer. This allows three auxiliary indexes to be built that record offsets in the grammar file. The first, the *rules* index, specifies the file offset on disk where each rule begins. The second, the *symbol* index, lists all offsets for each unique symbol in the grammar. Because symbols occur a variable number of times, a second-level index records the start of the list of occurrences for each symbol in the symbol index. The third index records the offsets of the start of each document in rule *S*. In addition to the indexes, there are two files that record the frequencies of words and usage of rules. These are used for calculating stop-words, and for ranking rules by usage. All of these files are indicated by the store labeled "disk-based grammar with inverted index" in Figure 9.

Once the files are produced, a server is started that awaits queries on a socket. The smaller indexes can be kept in main memory, but the grammar and the symbol index are accessed by disk seeks. When a word is requested, it is translated to its index in the lexicon and passed to a program that traverses the hierarchy on disk. Each occurrence of the word is located using the symbol index, and the expansion of each rule is calculated and returned. These rules are expressed as word numbers, and the numbers are translated back into words before being returned to the client. Along with each rule, the non-terminal representing that rule is returned, to allow queries to be made on that rule, and so on, recursively. Each query involves at least two disk seeks per item returned, but network transfer time is usually the bottleneck.

Stop-words are implemented using the lexicon frequency file, and are operationally defined as the 100 most frequent words. This level is user-adjustable as shown in Figure 5. If a phrase extends the query word or phrase by merely adding a stop-word, for example the phrase *the grammar* based on the word *grammar*, that phrase is considered uninteresting. In this case, all occurrences of the phrase *the grammar* are sought, and processing continues recursively until all contribute at least one word that is *not* a stop-word to the query word or phrase. This recursive expansion is performed by the "*traverse hierarchy*" process in Figure 9.

As mentioned earlier, an improved stemming feature has been added to the user interface: Figure 10 shows its effect. A collection of six months of the *Journal of Biological Chemistry* was queried with the word *enzymes*. The terms returned include *methods enzymol*, a commonly-used abbreviation for the journal *Methods in Enzymology*, both *restriction enzymes* and *restriction enzyme*, *enzymatic activities*, and so on. Stemming is implemented by invoking a black-box stemming algorithm (we use the Lovins, 1968, stemmer) on the user's query word in the "word translation" block of Figure 9, and expanding that stem against the lexicon file to locate all words that match. Then each such word is processed to find the non-trivial rules that contain it, as described above, and the results are concatenated and sorted before displaying them to the user. In our initial tests, stemming appears to significantly enhance the list of terms. We plan to improve stemming further by merging terms when their stemmed versions are identical. For example, the terms *restriction enzymes* and *restriction enzyme* in the second and third rows of Figure 10 would become a single entry.

5 Discussion

The intention of this work is to give the user a good idea of the subject matter of the text in a large collection, and present it in manageable chunks determined by the branching factor of each rule. This is especially evident at the very top level, where, in the examples of Figure 4b and Figure 8, the list of rules involving the word *grammar* provides a plausible taxonomy of concepts involving grammars. The terms in the list bear some resemblance to entries in a traditional book index. Whereas they may not always be of the same quality as hand-crafted entries, they do have the advantage of being inferred automatically. For multi-gigabyte corpuses, this advantage becomes significant.

The hierarchy produced by our technique contrasts with primitive methods such as keyword-in-context (KWIC) displays, where all occurrences of a search term are displayed along with the surrounding context. KWIC indexes do not scale, because the amount displayed for any given search term depends linearly on the size of the collection: this is presumably why they have fallen out of use. In our hierarchical approach, the amount displayed grows logarithmically with collection size (although we have not yet been able to establish this result theoretically).

Here are two examples that underscore this point. First, consider the discovery of the acronym *PCFGs* that was made when analyzing Figure 4. The fact that this acronym co-occurs with the phrase several times would not be visually apparent in a KWIC display, and would not receive much prominence. Its embodiment in a rule, however, ensures that the SEQUITUR-based method puts it near the top of the list of occurrences. Second, the word *index* occurs 487 times in the collection used to create Figure 1, in two hundred separate news articles. A KWIC display of each of these occurrences would be less than useful. The display would have been cluttered up with many junk occurrences of *index htm* and *index html*, and the key phrase *indexing and retrieval* would be unlikely to be spotted, as would the other key phrases *citation index*, *knowledge index*, *indexing nlp*, *web index*, and so on.

Although SEQUITUR's phrases give a good general idea of the structure of the grammar and the frequency of the phrases, there are several situations where phrases are given too much importance—or too little. Phrases receive artificially high frequency when an author quotes sections of a paper in its abstract, or—worse still—when the title of a paper is repeated in the header of every page. It is usually obvious when this has happened because phrases become very long and occur at the same level of the grammar. Similar problems have been encountered with the headers of news articles, and with references and bibliographies.

SEQUITUR's grammars often result in phrase boundary conflicts. For example, the phrase *indexing and retrieval* occurs twice in the first column of Figure 1. (The first occurrence is made up of the symbols *indexing and* and *retrieval*, and the second of *indexing* and *and retrieval*.) Due to these conflicts, important phrases may be overlooked because they are not listed as prominently as they should be. This suboptimality also gives rise to uninteresting phrases such as *a mobile robot* in Figure 5. This is an unfortunate side-effect of the greedy algorithm that we use for efficiency reasons. However, this problem will be corrected by the new stemming process, which will post-process the phrases before they are displayed to the user and amalgamate entries for phrases that stem to the same thing.

Another problem results from our definition of common words as the most frequently-occurring terms in the collection. Sometimes words are inappropriately classed as "common." For example, in one subcollection of the Computer Science Technical Report library, *neural* was found to be among the one hundred most frequently used words. In the Journal of Biological Chemistry, the word *amino* was deemed a stop-word, so the phrase *amino acid* was expanded as an uninteresting extension of *acid*. We are experimenting with the idea of having a fixed list of stop-words rather than inferring them from the collection on a frequency basis. The static list could be formed based on the intersection of frequent words from a number of diverse collections.

6 Conclusion

The thrust of this research is to build systems that let users become familiar with the content of a digital library by browsing a hierarchical structure of phrases that are repeated frequently within the text. Despite our purely lexical approach to phrase identification, the structures that are obtained in practice frequently correspond to plausible conceptual hierarchies. This permits large corpora of text to be browsed efficiently, and any particular document can be accessed in a number of steps that varies with the logarithm of the size of the corpus.

The method can be used for very large collections, and its operation has been demonstrated on a text base of 62 million words. We have recently developed a bounded-memory version of the SEQUITUR algorithm that allows us to process texts of arbitrary size, although some work still needs to be done to make all processing steps operate successfully with unbounded collections.

We believe that in the context of large information bases such as the New Zealand Digital Library, this interface will obviate the “query and hope” approach to browsing, and allow users to develop an intuition that would otherwise be very difficult to acquire.

References

Bell, T.C., Cleary, J.G. and Witten, I.H. (1990) *Text compression.* Prentice Hall, Englewood Cliffs, New Jersey.

Chang, S.J. and Rice, R.E. (1993) “Browsing: a multidimensional framework,” Annual Review of Information Science and Technology, 28, 231–276.

Cutting, D., Karger, D., Pedersen, J. & Tukey, J.W. (1992) “Scatter/Gather: A Cluster-based Approach to Browsing Large Document Collections,” *Proc. 15th Annual International ACM/SIGIR*, Copenhagen.

Harman, D.K.E. (1992–96) “Proc. TREC Text Retrieval Conference,” Gaithersburg, MD: National Institute of Standards Special Publication, 500-207, 500-215, 500-225, 500-236.

Hearst, M. and Karadi, C. (1997) “Cat-a-Cone: An Interactive Interface for Specifying Searches and Viewing Retrieval Results using a Large Category Hierarchy” Proc. ACM/SIGIR, Philadelphia, PA, July.

Lovins, J.B. (1968) “Development of a stemming algorithm,” *Mechanical Translation and Computation, 11*(1–2), 22–31.

Nevill-Manning, C.G. (1996) “Inferring sequential structure,” D.Phil. thesis, Computer Science Department, University of Waikato, New Zealand.

Nevill-Manning, C.G. and Witten, I.H. (1997) “Identifying Hierarchical Structure in Sequences: A linear-time algorithm ,” *Journal of Artificial Intelligence Research, 7*, 67-82.

Nevill-Manning, C.G., Witten, I.H. & Paynter, G.W. (1997) “Browsing in digital libraries: a phrase-based approach,” *Proc. 2nd ACM International Conference on Digital Libraries*, R.B. Allen and E. Rasmussen (Eds.) Philadelphia, PA, pp. 230-236.

Salton, G. (1989) *Automatic Text Processing: the transformation, analysis and retrieval of information by computer*. Reading, Mass.: Addison Wesley.

Storer, J.A. and Szymanski, T.G. (1982) “Data compression via textual substitution.” *J Association for Computing Machinery 29*(4), 928–951.

Witten, I.H., Nevill-Manning, C.G., and Cunningham, S.J. (1996) “Building a digital library for computer science research: technical issues,” *Proc. Australasian Computer Science Conference*, Melbourne, Australia, 534-542.

Wolff, J.G. (1980) “Language acquisition and the discovery of phrase structure,” *Language and Speech, 23*(3), 255–269.

Appendix: The SEQUITUR algorithm

The basic insight of the phrase-finding method is that any phrase which appears more than once can be replaced by a grammatical rule that generates the phrase, and that this process can be continued recursively. The result is a hierarchical representation of the original sequence. It is not a grammar, for the rules are not generalized and are capable of generating only one string. (It does provide a good basis for going on to infer a grammar, but that is beyond the scope of this appendix.) A scheme that resembles the one developed here arose from the area of language acquisition (Wolff, 1980). Nevill-Manning (1996) and Nevill-Manning and Witten (1997) give a more comprehensive description of the SEQUITUR algorithm and its applications.

SEQUITUR forms a grammar from a sequence based on repeated phrases in it. The key difference from conventional grammatical inference techniques, and from dictionary-based text compression schemes (see e.g. Bell *et al.*, 1990), is that a hierarchical structure is formed from the sequence. Each repetition gives rise to a rule in the grammar, and is replaced by a non-terminal symbol, producing a more concise representation of the sequence. It is this pursuit of brevity that drives the algorithm to form and maintain the grammar, and as a by-product, provide a hierarchical structure for the sequence.

We illustrate the algorithm using characters as phrase structure elements, although when applying the method to browsing we generally use words. At the left of Figure 11a is a sequence that contains the repeating string *bc*. Note that the sequence is already a grammar—a trivial one with a single rule. To compress it, a new rule $A \rightarrow bc$ is formed, and both occurrences of *bc* are replaced by *A*. The new grammar is shown at the right the Figure.

The sequence in Figure 11b shows how rules can be reused in longer rules. It is formed by concatenating two copies of the sequence in Figure 11a. Since it represents an exact repetition, compression can be achieved by forming the rule $A \rightarrow abcdbc$ to replace both halves of the sequence. Further gains can be made by forming rule $B \rightarrow bc$ to compress rule *A*. This demonstrates the advantage of treating the sequence, rule *S*, as part of the grammar—rules may be formed in rule *A* in an analogous way to rules formed from rule *S*. These rules within rules constitute the grammar's hierarchical structure.

The grammars in Figures 11a and 11b share two properties:

p_1: no pair of adjacent symbols appears more than once in the grammar;

p_2: every rule is used more than once.

p_1 can be restated as "every digram in the grammar is unique," and will be referred to as *digram uniqueness*. p_2 ensures that each rule is useful, and will be called *rule utility*. These two constraints exactly characterize the grammars that SEQUITUR generates.

Figure 11c shows what happens when these properties are violated. The first grammar contains two occurrences of *bc*, so p_1 does not hold. This introduces redundancy because *bc* appears twice. In the second grammar, *B* is used only once, so p_2 does not hold. If it were removed, the grammar would become more concise.

The grammars in Figures 11a and 11b are the only ones for which both properties hold for each sequence. However, there is not always a unique grammar with these properties. For example, the sequence in Figure 11d can be represented by both of the grammars on its right, and they both obey p_1 and p_2. We deem either grammar to be acceptable.

SEQUITUR's operation consists of ensuring that both properties hold. When describing the algorithm, the properties act as *constraints*. The algorithm operates by enforcing the constraints on a grammar: when the digram uniqueness constraint is violated, a new rule is formed, and when the rule utility constraint is violated, the useless rule is deleted. The next two sections describe how this is performed.

Digram uniqueness

When a new symbol is observed, it is appended to rule *S*, the top-level rule. The last two symbols of rule *S*—the new symbol and its predecessor—form a new digram. If it occurs elsewhere in the grammar, the first constraint has been violated. To restore it, a new rule is formed with the digram on the right-hand side, headed by a new non-terminal. The two original digrams are replaced by this non-terminal. However, the appearance of a duplicate digram does not always result in a new rule. If the new digram appears as the right-hand side of an existing rule, then no new rule need be created: the digram is replaced by the non-terminal that heads the existing rule. The hierarchy is formed and maintained by an iterative process. Changes ripple through the grammar, forming and matching longer rules higher in the hierarchy.

Rule utility

So far, it seems that the right-hand side of any rule in the grammar will only ever be two symbols long. However, longer rules are formed by the effect of the rule utility constraint, which ensures that every rule is used more than once. When a new symbol is appended to the top-level rule, the new digram that it creates may begin with a non-terminal symbol—which must of course be defined elsewhere in the grammar. Suppose this new digram appears only once in the rest of the grammar. Then a new rule will be defined to replace the digram. The fact that the non-terminal symbol is only used in this new rule violates the rule utility constraint. Therefore the non-terminal is removed from the grammar, its definition being incorporated into the new rule that has just been formed. This is the mechanism for forming long rules: form a short rule temporarily, and if subsequent symbols continue the match, allow a new rule to supersede the shorter one.