The TELLTALE Dynamic Hypertext Environment: Approaches to Scalability

Claudia Pearce
U.S. Department of Defense
9800 Savage Rd.
Fort Meade, MD 20755-6000
cepearc@afterlife.ncsc.mil

Ethan Miller
CSEE Department
University of Maryland Baltimore County
Baltimore, MD 21250
elm@acm.org

March 28, 1997

Abstract

Methods and tools for finding documents relevant to a user's needs in document corpora can be found in the information retrieval, library science, and hypertext communities. Typically, these systems provide retrieval capabilities for fairly static corpora, their algorithms are dependent on the language for which they are written, *e.g.* English, and they don't perform well when presented with misspelled words or text that has been degraded by OCR (optical character recognition) techniques. In this chapter, we present the TELLTALE system. TELLTALE is a dynamic hypertext environment that provides full-text search from a hypertext-style user interface for text corpora that may be garbled by OCR or transmission errors, and that may contain languages other than English by using several techniques based on n-grams (n character sequences of text). In this chapter, we identify methods and techniques that we have applied to the n-gram data structures. We also discuss algorithms that we used to enhance the scalabilty of the TELLTALE Dynamic Hypertext System.

Keywords: n-gram, hypertext, information retrieval, full-text, similarity, OCR, multilingual.

1 Introduction

Knowledge workers of many kinds, from scientific researchers, to reporters, office workers, and others, all need to process large growing collections of textual documents on a daily basis. This environment is characterized by changing corpora that may contain text that deviates considerably from standard English. Office workers must sift through a morass of electronic information such as electronic mail, optically scanned documents, and other machine-readable text. International business must also be capable of processing multiple languages, sometimes within the same document. Text may be corrupted through faulty transmission, translation, or optical character recognition. Unfortunately, traditional tools for finding information in text are geared primarily for one language (and often are restricted to one domain within that language) and are not well suited to corrupted data.

Appeared as a chapter in Advances in Intelligent Hypertext, J. Mayfield and C. Nicholas, eds. Lecture Notes in Computer Science, Springer-Verlag, October 1997, pages 109–130.

Contemporary full text search tools fall into two major categories: *traditional text retrieval* and *hypertext*. Traditional text retrieval tools process and scan text corpora against which users can pose queries [SM83]. Queries can range from highly-structured Boolean queries to completely-unstructured natural-language queries. Query results in traditional systems can consist of an unordered set of documents – as is the case in most commercial systems – or a ranked list of documents presented in decreasing order of relevance. Hypertext provides a somewhat different method of accessing the underlying corpus [Nie90]. Access to related items of information is provided through a navigational-style interface in which a user selects highlighted areas of text in a document which then causes related information to be presented to the user. Types of links (*e.g.* associations from one document to another) can vary, but an underlying collection of linked segments of text is essential.

Each category of tool, traditional text retrieval and hypertext, has its strengths and weaknesses. For example, traditional text retrieval tools often contain many language dependent features such as stop word lists and stemming algorithms. In addition, these traditional text retrieval tools often use the roots of all non-stop words as index terms. The reader is referred to Croft [CT87] and Salton [SM83, 118-146] for descriptions of traditional systems. This process provides variability in the endings of words in a document but not in the roots of the words. Hypertext tools are traditionally of the static-corpus, hand-generated variety [EE68, Nel88] or, as in many newer versions, incorporate traditional text retrieval tools [CT89, FC89, CCA89, ACR+90] and are subject to the same problems as traditional text retrieval tools. Hypertext has the advantage of providing an intuitive user interface that allows users to jump from one document to related documents that have been selected solely on the basis of the text in the original document. This is in sharp contrast to traditional full text retrieval which relies on complex query languages.

TELLTALE draws on the strengths and addresses the weaknesses of both technologies to provide a dynamic hypertext environment using new variations on traditional full text retrieval. With this marriage of technologies and the incorporation of unique techniques to build in tolerance to garbles and to remove language dependencies, TELLTALE is a versatile full-text retrieval and dynamic hypertext tool. The purpose of this chapter is to provide an overview of the TELLTALE dynamic hypertext environment and its dynamic linking methods, to describe how language independence and garble tolerance are achieved in those linking methods, and to illustrate recent efforts to enhance TELLTALE's scalability for large document collections. Section 2 provides an overview of the TELLTALE Dynamic Hypertext Environment. The design and development of a TELLTALE-like research platform for testing a variety of scalability issues is covered in Sections 3. The chapter concludes with a discussion of related efforts and future work in Section 4.

2 TELLTALE – A Dynamic Hypertext Environment

TELLTALE provides dynamism in its approach to full text retrieval in three ways. First, TELLTALE provides dynamism in selection of the *anchor* (selected area of text used as the source of links to other related portions of text) which allows users complete flexibility in their search for information. Second, the choice of link computation by the user at run-time provides another area of dynamism. By giving users a choice of link computation, they can select the choice most appropriate to their needs at the moment. Third, by actually

calculating the link at run-time, the system can incorporate any new data that has been added since starting the system. The actual links are computed dynamically, not manually generated for a specific corpus. This combination of link type selection, anchor selection, and link computation serves as the basis for relating and querying the full text of documents in an underlying corpus. These dynamic mechanisms allow users to pose questions about the underlying corpus using a passage of text, to investigate the relationships between documents through navigation, and to browse a dynamic corpus.

The bases for the dynamic linking capabilities in TELLTALE are enhanced versions of the traditional statistically-based information retrieval tools. To overcome any dependence on a particular language, *e.g.* English, and to build in tolerance to spelling errors, we use *n-grams*, n-character sequences of text, in our statistically-based tools to supply some of the needed robustness. This use of n-grams, along with unique scoring techniques, provides the required robustness for garble tolerance and language independence.

Since anchors provide explicit visual cues for users, specialized highlighting tools are included in TELLTALE so that users can readily identify meaningful information in a hypertext without explicit anchor points. So that the highlighting tools in TELLTALE are also garble resistant and language independent, n-gram-based techniques are used in these tools as well. Two types of highlighting tools are provided in TELLTALE. First, *topic* highlighting is provided which highlights the n-grams contained in the anchor that occur in a selected target document (a document that resulted from traversing a link). The effect of the topic highlighting is to provide users with quick visual recognition of words and phrases from the anchor that are present in a target document while allowing for variability in spellings and endings. Second, a *statistical* highlighting technique, based on a technique by Cohen [Coh95], is provided which highlights words and phrases that contain n-grams that occur more frequently than statistically expected based on a large sample. The effect of this statistical highlighting is to provide the gist of a selected target document.

In this section we discuss the mechanics of TELLTALE's linking mechanisms and the features of its user interface. A more complete discussion of TELLTALE's highlighting techniques can be found in Pearce's dissertation [Pea94]. In Section 2.1 we discuss the n-gram approach used in TELLTALE as well as three dynamic linking mechanisms based on this n-gram approach. In Section 2.2 we discuss the TELLTALE user interface.

2.1 Dynamic Link Types Using n-grams

Partial character sequences of length n extracted from documents, called n-grams, have been used in several automatic document indexing schemes in several systems [DM85, Wil79, ZPZ81, Dam95a]. Zamora [ZPZ81] uses trigram analysis for spelling error detection. Damashek [Dam95a] uses n-gram analysis for similarity scoring with multiple languages and robustness against misspellings. These n-gram approaches vary in the choice of n, the process for extracting n-grams, and the statistics stored about n-grams. Some n-gram approaches extract all unique n-grams in words, but ignore cross-word boundaries [ZPZ81, Sue79]. Other approaches keep statistics on n-grams at various start positions within words [Sue79]. A further variation on n-gram generation is to include interword spaces so that n-grams spanning words can be monitored [YGH82, Cav93]. Damashek's method finds all unique n-grams in a document including interword spaces. This can be

referred to as the *sliding n-gram* approach since a document can be easily scanned for all its unique n-grams by sliding an n-byte window over the text. Language independence is achieved by using all unique n-grams in a document and corpus when building document representation vectors. This process replaces the keyword stemming commonly used in traditional information retrieval systems [SM83] and eliminates other language-dependent features such as stop word lists. Robustness to errors in spelling is gained because of the redundancy introduced with the sliding n-gram approach, which identifies all unique n-grams in a document. Since not all characters in a word will typically be included in each n-gram that contributes to a word, considerable flexibility is built into the approach. In addition to the overlap provided by the sliding n-gram approach, Damashek's weighting scheme is unique in that it removes commonality among documents by generating an "average" document from the full corpus and then removing "average" n-gram weights from individual n-gram weights in a document. This has the effect that n-grams covering stop words and other common words are weighted less highly than n-grams covering other words. In contrast to the complete coverage of the sliding n-gram approach of Damashek, Mah [DM85] collects high-frequency bigrams and trigrams to be maintained as indexes, then combines these bigrams and trigrams to locate specific words and phrases. Low frequency bigrams and trigrams are not used so that the size of the index remains small. As a result, Mah's approach does not have the same level of coverage of characters and inherent robustness as does Damashek's. Cavnar's method tracks interword boundaries by maintaining all bigrams and trigrams in each line of a document, but misses interline boundaries [Cav93]. Yannakoudakis' method [YGH82] collects n-grams containing spaces at word endings only.

Sliding n-gram analysis serves as the basis for the three dynamic linking methods in TELLTALE: Similarity links, Lookup links, and Disambiguated Lookup links. The Similarity link is a method of linking documents based on the closeness of two document's respective vocabularies. Documents are represented by a vector of weights representing the contributions of various n-grams. Similarity is calculated by computing the cosine of the two vectors of n-gram frequencies. The Lookup link is a method of linking documents based on the percentage of the unique n-grams contained in an anchor string that actually occur in a document. The Lookup link can be thought of as a fuzzy string match. It is, as its name implies, a *lookup* of certain n-grams. The Disambiguated Lookup link is a method of linking documents based on a combination of the above two link types. Specifically, the Similarity link is used to provide context and the Lookup link is used to find specific strings. This effectively disambiguates the context in which the search string is used. In Sections 2.1.1, 2.1.2, and 2.1.3 the three methods of calculating dynamic links in TELLTALE are described. A more complete discussion of the implementation details can be found in Pearce [Pea94].

2.1.1 Similarity Link

The weighting scheme in Damashek's method [Dam95a] – the basis for the Similarity link – uses counts of each unique n-gram in the corpus and in each document. To provide these counts, a histogram is maintained for all unique n-grams in each document and in the corpus as a whole. Consider, for example, the following passage of text. To reduce the character set size, punctuation, special characters, and numerics have been removed (only the 26 lower case letters of the alphabet and the space remain) for the purpose of n-gram

detection and accumulation.

> ...expert systems can be used in many different types of problem areas places where expert systems make an important difference include ...

Choosing $n = 5$, the first several 5-grams are "exper", "xpert", "pert ", "ert s", and "rt sy". Notice that these same 5-grams occur again later in the passage, adding to their respective histogram counts.

For fast access we internally store this histogram of n-grams as a hash table of n-grams, using each n-gram as an access key. A *hash table* is a file organization in which records in the file are divided into a collection of numbered buckets. Assignment of records to buckets is determined by a function that transforms the value of a key in each record into a bucket number. This transformation function is called a *hash function*. With proper choice of hash function, storage of records is evenly distributed among the buckets. The hash function used in TELLTALE is based on the ASCII values of the characters in each n-gram. TELLTALE's hash function is described in Pearce [Pea94] and a full description of hashing can be found in Knuth [Knu73]. A collision list of unique n-gram occurrences is maintained in each bucket of the hash table. Tied to each unique n-gram is a list of documents in which that n-gram occurs, along with a count of how often it occurs in each document. For the English alphabet, there are 27^n possible n-grams. However, for any given document, relatively few of these will be represented. An upper bound on the number of unique n-grams in a document of size m is $m - n + 1$. Through experimentation, it has been found that 80,000 to 100,000 hash table buckets for 5-grams are sufficient to ensure relatively short collision lists — fewer than five elements on average — for corpora over forty megabytes in length.

In calculating the similarity of two documents, n-grams are used as index terms. The weight of each term is the difference between the count of a given n-gram for a document, normalized by the document's size, and the average normalized count over all documents for that n-gram. This provides a weight for each n-gram in a document relative to the average for the collection. For example, given the histogram count, $c_{i,k}$, of n-gram k in document i, the total n-gram count, m_i, in document i, and the average normalized count, a_k, of n-gram k over all documents, a document is represented as a vector $d_i = (d_{i,1}, d_{i,2}, \cdots)$ where the individual elements, $d_{i,k} = c_{i,k}/m_i - a_k$, have been normalized and the n-gram's average value removed. This is the same as dividing a document's vector by the document's total n-gram count and subtracting the corpus' *centroid* — the vector composed of the average weight for each n-gram in the corpus. The similarity between document vectors d_i and d_j is then calculated as the cosine of the two representation vectors,

$$SIM_c(d_i, d_j) = \frac{\sum_{k=1}^{t}(d_{i,k} \cdot d_{j,k})}{\sqrt{\sum_{k=1}^{t} d_{i,k}^2}\sqrt{\sum_{k=1}^{t} d_{j,k}^2}}. \quad (1)$$

The numerator in Equation 1 is the dot product of the vectors d_i and d_j, representing documents i and j respectively. The denominator in Equation 1, the product of the sum of squares of each term in the respective vectors, is used to normalize the result. The average n-gram vector can be calculated by maintaining a running total for each n-gram and document size as documents are scanned. Scores computed using this approach range from -1 to 1.

Similarity links provide a mechanism for linking similar documents. This result is achieved because documents about the same subject tend to use the same vocabulary. In TELLTALE, vocabulary consists of n-grams that make up the words, not the words themselves. Queries are processed in the same manner as documents in the corpus. In TELLTALE, complete documents be scored not only against queries, but against any selected areas of text. The selected area of text becomes the anchor of the link and the anchor is treated as a new "document" to be scored against all other documents. To make this computation efficient, TELLTALE's implementation reuses certain terms in the Similarity link score [Pea94]. The effect of the Similarity link computation is that of an associative table of contents. The user provides a sample of the "content" of interest and the hypertext engine supplies a list of relevant documents by using the Similarity link computation. The hypertext system then presents the user with a uniquely tailored selection of the corpus based on her information need.

2.1.2 Lookup Link

The hash table construction for holding the histogram of n-grams and document references discussed in Section 2.1.1 can be used as an inverted index into the document collection. To compute a Lookup link, the n-grams from the selected query phrase are first parsed, then each unique n-gram is hashed to find the documents in which it is contained. In other words, we "look-up" every unique n-gram in the hash table to find documents in which it occurs. If all of the n-grams of a query appear at least once in some document, then that document has a high probability of containing that query phrase. Also of interest are documents that contain lexically close matches of the word or phrase in the anchor. Thus, instead of looking only for documents in which all of the n-grams from the query phrase are present, those containing some percentage of the query n-grams, say 50 percent, are also selected. With this approach, some documents that do not contain the specified phrase will be selected, resulting in false hits. Those documents were selected, however, because they contained at least half (or some other percentage chosen by the user) of the unique n-grams present in the query. Precision is sacrificed for recall since with degraded text (text with many spelling errors), relevant documents would be overlooked if the required percentage was too high.

The Lookup link functionality can be expressed mathematically as the following asymmetric binary similarity score:

$$SIM_l(q, f_i) = \frac{\sum_{k=1}^{t}(q_k \cdot f_{i,k})}{\sum_{k=1}^{t} q_k} \tag{2}$$

where q is the query or link anchor, f_i is the ith document in the corpus, t is the total number of unique n-grams in the corpus, and q_k and $f_{i,k}$ are binary values, 0 or 1, representing whether the query and document, respectively, contain n-gram k at least once. The numerator is the dot product of query and document representation vectors in which binary values are used in the weighting instead of the more precise weighting used in the Similarity link. This score is asymmetric since one cannot reverse the order of the query, q, and document, f_i, without possibly getting a different score. The score is geared specifically toward the query since the denominator reflects only n-grams from the query.

The Lookup link associatively and robustly indexes documents. It is associative in the sense that pointers or offsets to specific locations in the text are not used in the hash table as they are in other inverted file indexing schemes. Instead, the lookup is based purely on whether a document contains various n-grams and not on direct pointers to exact locations where the specified information resides. In contrast to the Similarity link scoring, the Lookup link provides a method for finding a word or phrase that may not have sufficient weight to perform well as a query when using the Similarity link. As a side effect, this technique will bring up documents on a variety of topics that happen to contain a percentage of the requisite n-grams in the lookup query. Additionally, the Lookup link provides users with a "browsing" tool. It lets the user examine the senses and contexts in which a term is used while tolerating varied spellings.

2.1.3 Disambiguated Lookup Link

Disambiguated Lookup links provide a way to narrow the scope of either a Similarity link or a Lookup link by combining the two methods. The Lookup link is designed to find documents that contain strings closely matching the anchor of the Lookup link. For example, "dolphin" (or some close variation) can be used in different contexts in different documents. One document might use "dophin" in the context of a football team, while another document might use it in the context of large aquariums. In the Disambiguated Lookup link, the current document is used to provide context for disambiguating the many senses of a word or phrase such as "dolphin." Conversely, from a collection of documents that result from a Similarity link, *i.e.* documents on a given topic of interest, Disambiguated Lookup links can select documents within the set of interest that contain a specific word or phrase. This refinement procedure can be thought of as an AND operation. Mathematically, the set of documents retrieved from a Disambiguated Lookup link can be thought of as the intersection of the set of documents selected from the Similarity link and the Lookup link. Given $SET_l = \{d_i | SIM_l(q, d_i) > threshold_l\}$ and $SET_c = \{d_i | SIM_c(q, d_i) > threshold_c\}$ for a query q we represent the disambiguated set as:

$$SET_d = SET_c \bigcap SET_l \tag{3}$$

The threshold values, $threshold_l$ and $threshold_c$, are left to the discretion of the user; lowering thresholds improves recall at the expense of precision, while raising the threshold has the reverse effect. Based on *ad hoc* experimentation, it has been found that a good value for $threshold_l$ is approximately .5, or 50 percent when translated from the range $[0, 1]$ to a percentage, and a good value for $threshold_c$ is approximately .2, or 60 percent when translated from the range $[-1, 1]$ to a percentage.

It should be noted that all of the dynamic link methods described here produce a set of documents instead of a single document. The set of documents selected is based on the values of the score of a document versus the query (*i.e.* anchor). The documents that pass the established threshold are then ranked in descending order of score.

2.1.4 Link Retrieval Performance

Pearce has shown TELLTALE to be an effective retrieval tool for both clean and degraded data in [Pea94, Pea95]. The reader is refered to these works for a complete description of

retrieval performance analysis for TELLTALE's linking methods.

2.2 Hypertext Interface Overview

Figure 1 shows the basic blocked window design of TELLTALE. Figure 1 illustrates five window areas. The area in the bottom left-hand corner is a writable area in which a user can place any selection of text to be used as a query (link anchor) either by typing into the area, by dropping in selected text from other windows, or by loading text from a selected file. This area is labeled "**Topic Text**" and it serves as a location from which the user can pose an initial query. Such an initial query provides the user with a potentially relevant set of documents and provides an entry point into the corpus. The user chooses the type of dynamic query computation from either the "**Lookup**" or "**Score**" buttons located below the **Topic Text** window. The "**Browse List**" button allows users to save lists of interesting queries. **Lookup** computes a Lookup link and **Score** computes a Similarity link. The choice selected for the calculation in the example pictured in Figure 1 was **Lookup**. The **Topic Text** area can be used at any time, but is particularly effective when a user starts the environment and loads new data. In this case, the **Topic Text** area serves as an entry point into the system after data has been processed. The window on the lower right, titled "**Pearce Lookups**," displays a ranked list of documents resulting from the query. In the figure, documents are ranked as a result of the Lookup link computation. The **Pearce Lookups** window is also used to rank and list documents that are the result of a **Score** (Similarity link) computed from the **Topic Text** area. A slider bar underneath this window allows the user to adjust the threshold of scores for documents retrieved. The values on the slider bar range from zero to 100 percent where 100 percent indicates complete similarity and zero indicates no similarity. The upper left-hand window, labeled "**Main Text**," displays the selected document, in this case the contents of the document whose identifier is highlighted in the **Pearce Lookups** window. The document displayed is an Associated Press document from the TIPSTER data corpus [Har93]. In the window labeled "**Damashek Documents**" in the upper right-hand corner of the interface is a list of documents similar to the document found in **Main Text** using the Similarity link. This list is updated every time the document in the **Main Text** changes. The **Damashek Documents** window is accompanied by a slider bar as in the **Pearce Lookups** window. The slider controls the threshold of similarity. The middle window on the right-hand side is used to display words of statistical significance called **Cohen Words**. Statistical highlighting gives the user a quick survey of the contents of a document. When the statistical highlighting is activated, words and phrases containing important n-grams are highlighted within the document in the **Main Text** window, in addition to being listed in the window labeled "**Cohen Words**." In Figure 1 the highlighting of the **Cohen Words** within the text in the **Main Text** window has been turned off to enable the anchor selection of "Bekaa Valley." In the true spirit of hypertext, the user can select any phrase of any length from the **Main Text** window to be used as the anchor of a dynamic link illustrated by the selection of "Bekaa Valley." A popup menu, activated by anchor selection and mouse click, in the middle of the **Main Text** window provides the selections of dynamic link computations, Similarity, Lookup, and Disambiguated Lookup for the next link operation. This and other features of the TELLTALE interface are covered at length in Pearce [Pea94].

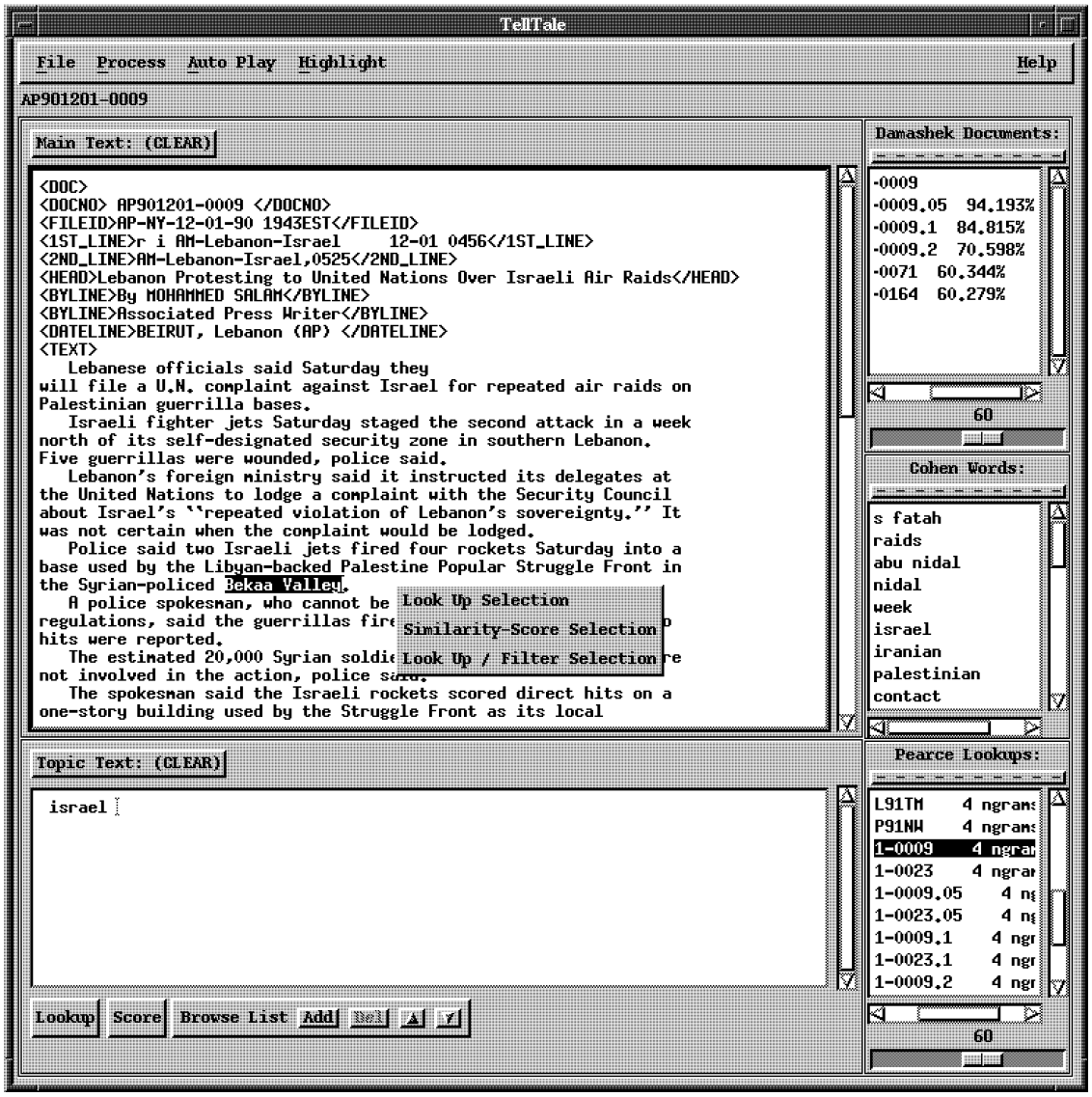

Figure 1: TELLTALE user interface with highlighted anchor and hypertext selection window.

2.3 Text Highlighting Techniques

As discussed earlier in this section, the dynamic hypertext environment provides two types of highlighting techniques. Topic highlighting is activated in Figure 2. Topic highlighing emphasizes any n-grams in the topic text that are present in the main text. By highlighting n-grams, some variablitiy in spellings is accommodated. For example, in Figure 2, protions of "Israeli", "Israel's", and "Arab-Israeli" are highlighted in response to the topic text "israel". Statistical highlighting (Cohen words) is activated in Figure 3. In this example, words and phrases containing high scoring n-grams, based on Cohen's statistic, are high-

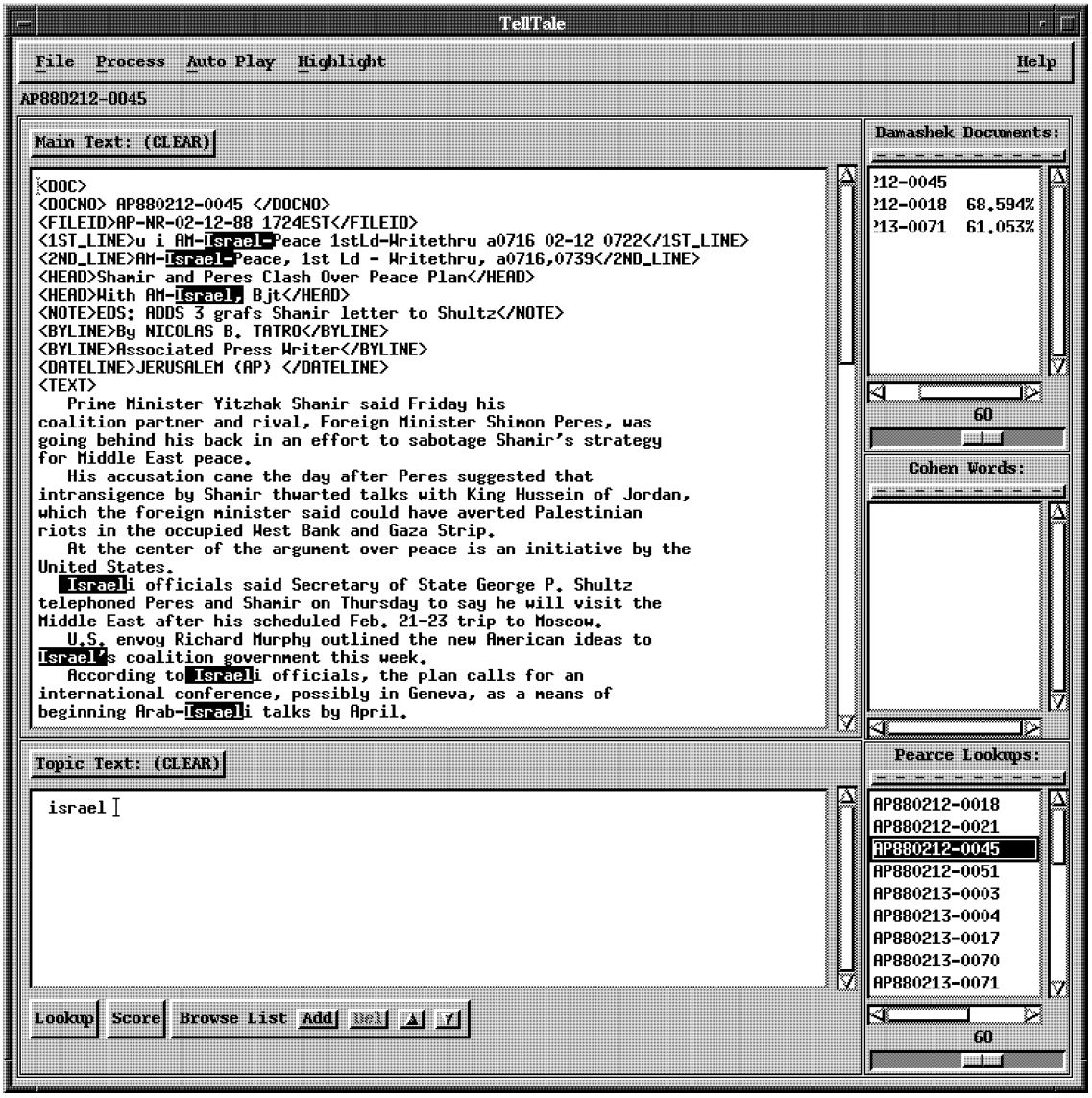

Figure 2: TELLTALE user interface with topic highlighting.

lighted. The effect of this highlighting is to provides the reader with a quick gist of the document's content. With a quick glance, the user can determine whether or not the read a doucment in more detail. The reader is refered to Cohen [Coh95] for a complete description of statistical highlighting calculations and to Pearce [Pea94] for the variation on Cohen's highlights used in the TELLTALE prototype.

2.4 Multilingual Text

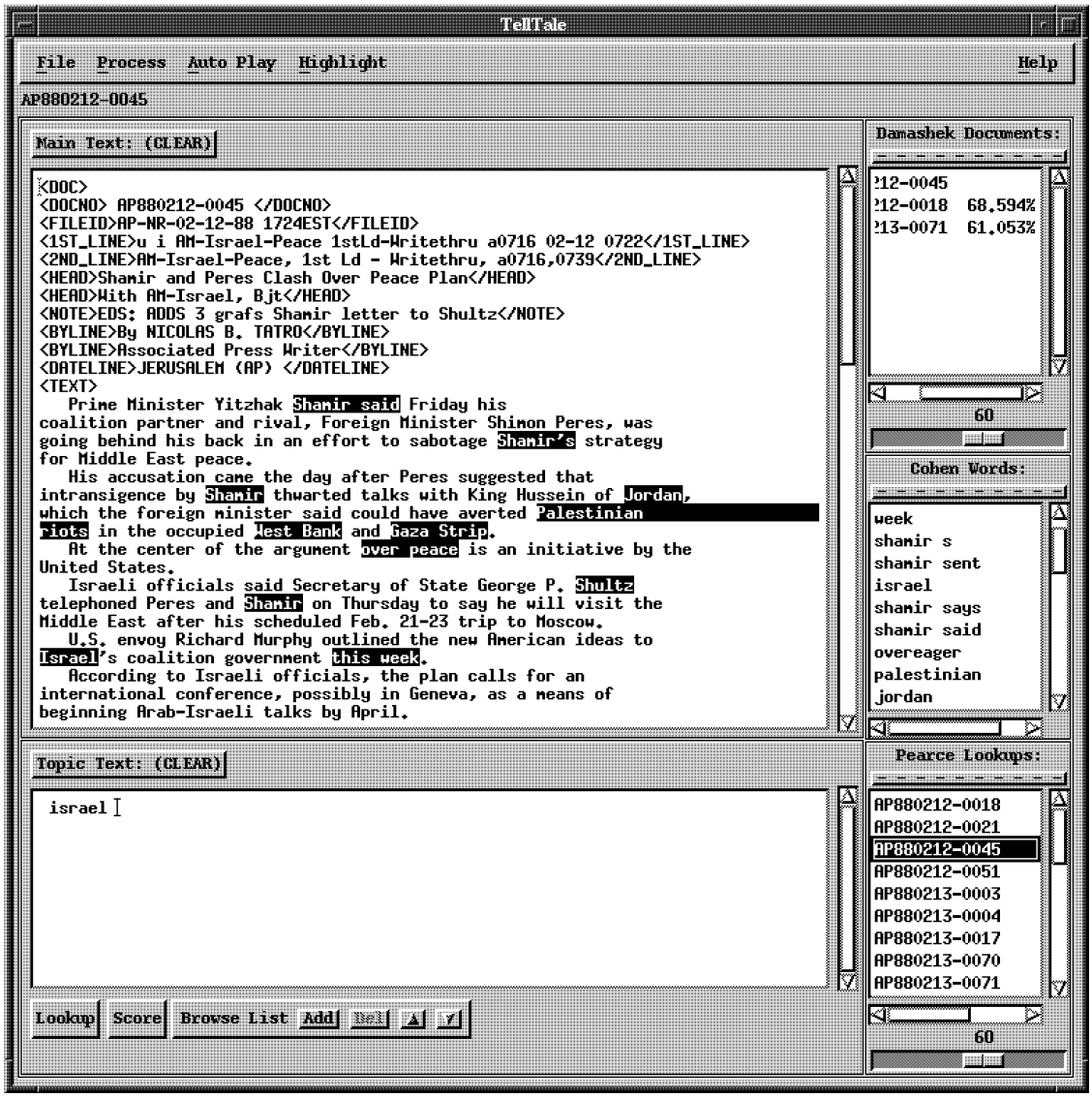

Figure 3: TELLTALE user interface with statistical highlighting of n-grams.

TELLTALE is multilingual in several senses. First, TELLTALE is multilingual because the algorithms used in TELLTALE are independent of any specific language texts to be analyzed or displayed. Second, TELLTALE is multilingual because it has the ability to display a variety of fonts, thus allowing languages to be displayed in their native scripts. Figure 4 shows the TELLTALE interface displaying Russian text extracted from an Internet newsgroup. The Russian displayed in this example was encoded as 8-bit ASCII using a KOI Cyrillic font encoding. By using the upper 128 values of the possible 256 available values in the full 8-bit ASCII encoding, fonts such as the KOI Cyrillic allow both English (Roman alphabet stored in the lower 128 bits of ASCII along with numbers and special characters)

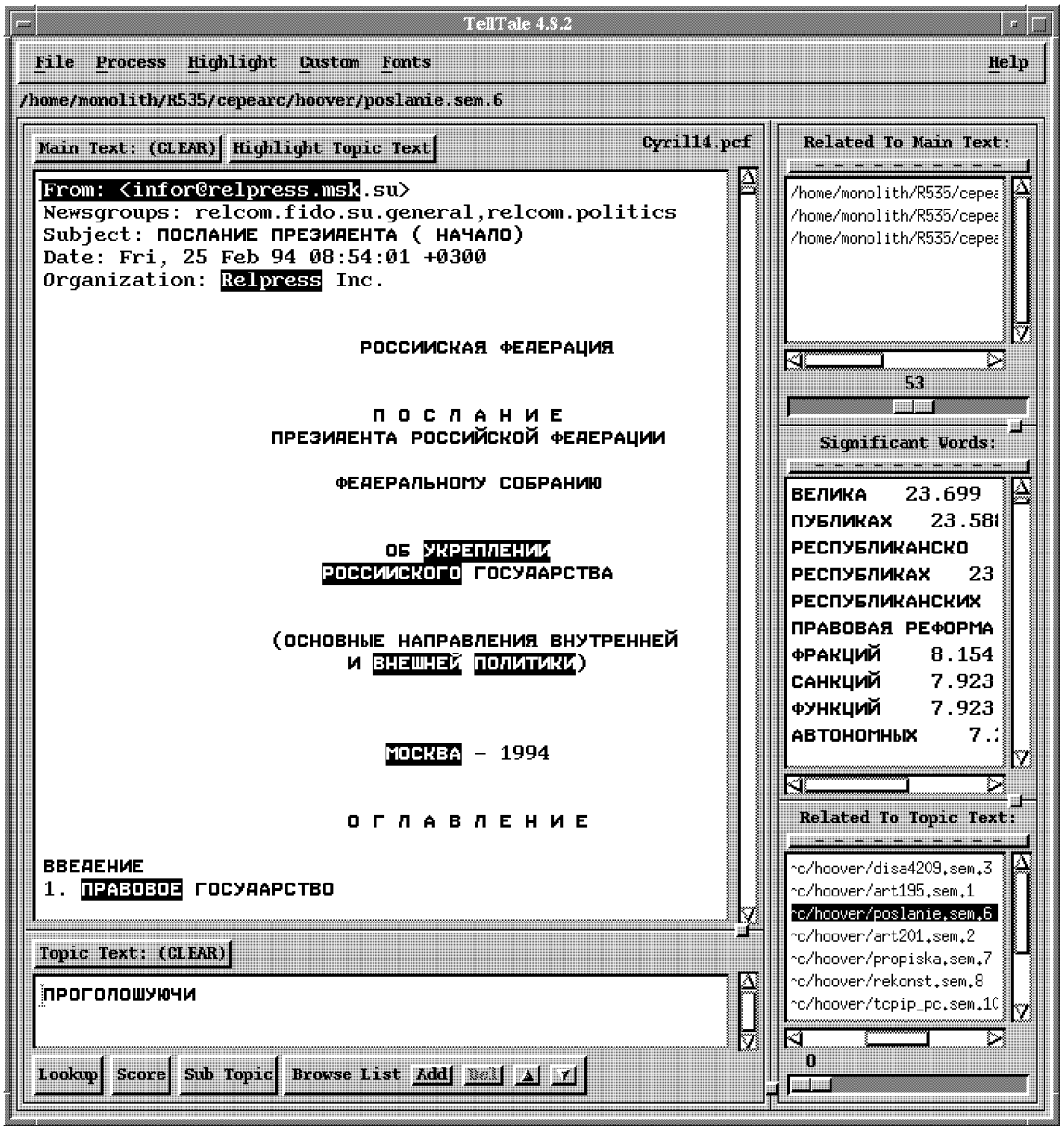

Figure 4: Russian document from an Internet newsgroup displayed in KOI encoding of the Cyrillic alphabet.

and Cyrillic (stored in the upper 128 values) to be displayed alongside of one another. Since TELLTALE's linking and highlighting algorithms are based on n-grams and not on words, the algorithms are not effected by the language used.

TELLTALE's original implementation did not contain the ability to display different language fonts. However, implementation of this new capability has proven to be very simple. Only minor changes in the interface were required and virtually all of the scoring and highlighting algorithms remained unchanged. In future implementations we hope to

include a more comprehensive multilingual capability that incorporates 16-bit encoding standards (*e.g.* UNICODE [Con92]) to enable the processing and display of languages that require much larger coding schemes, such as Japanese.

Precision/recall analysis of the link types with multilingual data was not conducted due to the lack of available test data and relevance judgments for multilingual data. Damashek, however, has conducted a number of tests with the sliding n-gram approach (the basis for our Similarity link) to demonstrate that languages can be sorted and identified using this approach [Dam95b].) Further experiments with multiple languages will be conducted as appropriate data sets become available.

3 Scalability Research Platform

The original TELLTALE prototype, while favorably viewed by users for its interface style, characteristics, and performance with several megabytes of data, was limited, however, in the volume of data that it could load and score. Users of this prototype were limited by the memory constraints of their workstation since TELLTALE precomputed (at startup) the similarity of each document in the corpus to every other document. This process is both lengthy and memory-intensive; starting up a copy of TELLTALE with 4,000 documents each 1 KB long could take several minutes, and the resulting data structures would consume over 32 MB of memory. We constructed a research prototype to explore methods that would allow us to bypass these limitations, allowing TELLTALE to handle corpora containing 10^9 bytes in 10^6 documents.

3.1 Prototype Goals

We built the prototype system for several reasons. First, we could experiment with different algorithms in a non-production environment. Currently, TELLTALE is used by a sizable community; separating the creation of new features from the maintenance of the current version allowed us to "break" old features in the pursuit of desirable new features. Using a prototype version also freed us from the need to implement a full user interface. As with many projects, the user interface is a large part of the code. Eliminating much of this code would make the system less suitable for production use, but it would not hamper the evaluation of new algorithms for the retrieval engine.

Building a separate research prototype also allowed us to evaluate new data structures and algorithms without necessarily including the baggage of previous versions. In some cases, the new methods we used in the prototype proved superior to the original algorithms. These improvements were merged into the next version of the production system. Additionally, we were able to implement new algorithms and data structures that supported scalability, as discussed in Section 3.2.

Our research prototype had one major feature not present in the production version — it was built using a `Tcl` (Tool Command Language) interpreter [Wel95]. The `Tcl` interface allowed us to quickly build a graphical interface using `Tk` [Wel95]. While this interface lacked some functionality present in TELLTALE, it was sufficient for our purposes and was implemented quickly. The `Tcl` interface has another major benefit, though. Since `Tcl` is a command language, the prototype could be controlled directly through scripts. This allowed

researchers to write batch scripts to test the prototype's performance without having to use the graphical interface. Since the only interface to the indexing engine was through `Tcl`, we were guaranteed that the scripts would have the same effects as the equivalent commands selected through the graphical interface.

3.2 Data Structures

The research prototype includes three main data structures. Each keeps information about a particular type of "object" within the system. The object types are n-grams, documents, and files. As with TELLTALE, each file contains one or more documents, and each document contains one or more n-grams. Rather than build our own data structures from scratch, we used classes from the GNU `libg++` class library. The data structures are built using hash tables; `libg++` hash table classes include automatic table resizing as well as standard operations such as insertion, deletion, access, and traversal.

As Figure 5 shows, the three main data structures in the prototype are interconnected; they may be used to find the documents that contain an n-gram or to locate the file that contains a document. Absent is a similarity matrix that provides the similarity for any two documents in the corpus relative to the centroid. This matrix is absent for several reasons. First, it occupied too much memory — space which could be better spent storing other data structures. Second, it required a lot of time to precompute. Computing the similarity of all documents against a single document is not too time consuming; however, it becomes very expensive when done for each document. Scalability is the third reason for this change. The prototype could not assume that all documents would be fully read in when the program was started because doing so would require too much time just to convert files into n-grams; thus, it would be impossible to compute the matrix. Even if the matrix could be computed, it would be far too large to reside in memory — the matrix would require 500 MB for just 50,000 documents.

The central data structure in the prototype is the n-gram hash table. This table contains an entry for each unique n-gram in the corpus; this entry contains information such as total count for the n-gram as well as data necessary for similarity calculations. In addition, each n-gram may have a list of postings — one for each document that contains one or more occurences of a particular n-gram. Our experiments using 5-grams showed that the 40 MB Wall Street Journal corpus contained just over 250,000 unique n-grams. In the same corpus, we found that a document with k n-grams had more than $k/2$ unique n-grams. As a result, the n-gram table required fewer than 8 MB to store the per-n-gram information, but would have required nearly 200 MB to store all of the n-gram postings. For large corpora, the prototype's memory limitation arises largely from n-gram postings rather than the n-gram hash table itself.

The document table contains one entry for each input document. It holds statistics such as document size and location (via pointer to the file table) for each document. It also contains per-document information used to quickly compute document similarities; the algorithm used is detailed in Section 3.3. This data structure uses fewer than 100 bytes per document. While this presents little difficulty for corpora of fewer than 50,000 documents, it may require some modification to accomodate a corpus of one million documents.

The file table serves as an interface between the document "world" of the TELLTALE

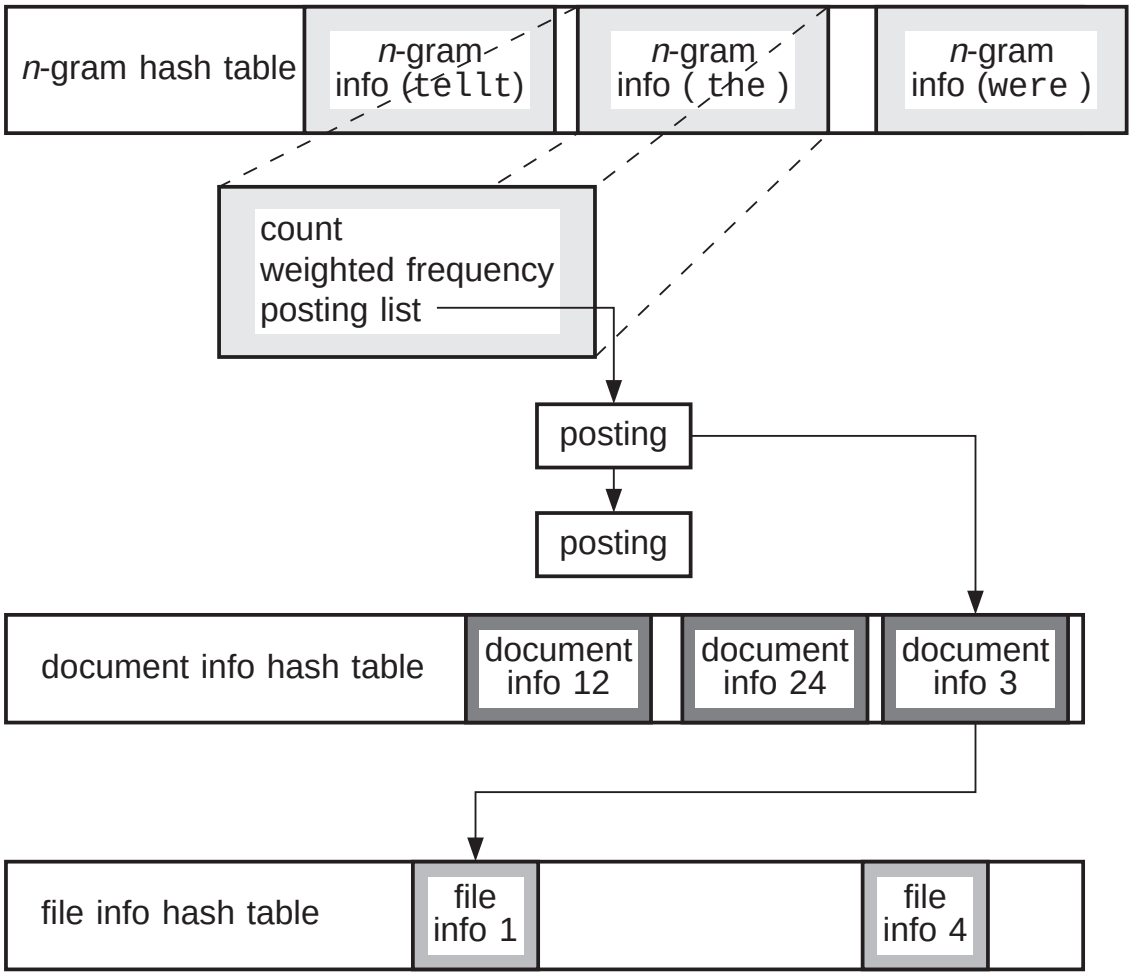

Figure 5: Data structures in the TELLTALE research prototype.

prototype and the file "world" of the Unix file system. Since each file may contain hundreds of documents, we decided to store per-file information such as file name separately; this provided a large savings in memory usage by allowing document hash table entries to hold a pointer to a "file" rather than keep all the file information themselves. While the table uses about 100 bytes per file, this cost is minimal even for a one gigabyte corpus containing 20,000 files of 50 KB each.

As we discovered, the most space-intensive portion of the prototype was the postings list. Thus, this is the data structure whose size must be reduced when the prototype is scaled to larger corpora. Solutions to this problem are discussed in Sections 3.4 and 3.5.

3.3 Algorithms Used

The TELLTALE research prototype used the same basic algorithms for similarity used in the production version, as detailed in Section 2.1.1. However, the calculation method

differed from that in the production version. TELLTALE precomputed all of the pairwise document similarities at startup, placing the results into a large matrix. The prototype, on the other hand, computes as little as possible when documents are read in. It postpones most of the similarity computation until it is actually needed.

The prototype must maintain both per-n-gram and per-document information to allow it to quickly compute the similarity between two documents. We can rewrite Equation 1 as

$$SIM_c(d_i, d_j) = \frac{\sum_{k=1}^{t}(x_{i,k}x_{j,k} - x_{i,k}a_k - x_{j,k}a_k + a_k^2)}{\sqrt{\sum_{k=1}^{t}(x_{i,k} - a_k)^2}\sqrt{\sum_{k=1}^{t}(x_{j,k} - a_k)^2}} \tag{4}$$

where $x_{i,k} = c_{i,k}/m_i$. The only part of this sum that must be recomputed for each query document is the sum of $x_{i,k}x_{j,k}$. However, this product is zero unless *both* values are non-zero, i.e., if both documents contain the n-gram in question. Similarity computations can thus be done on the fly by taking a single trip through the hash table and keeping a running total of this product for each document. This product is then combined with precomputed values for $\sum_{k=1}^{t} x_{i,k}a_k$ computed for each document i, $\sqrt{\sum_{k=1}^{t} x_{i,k}^2}$ computed for each document i, and $\sum_{k=1}^{t} a_k^2$. These values only change when new documents are added to the centroid, and are not recomputed for every similarity computation. Instead, they are figured the first time a user requests a similarity computation after the centroid has been modified.

By separating the similarity calculations into two parts, the prototype provides the user with better performance. Documents can be scanned in at the rate of approximately 100 KB/sec on a Sparc-10, allowing a 4 MB corpus to be "digested" in 40 seconds. The precomputations previously mentioned require an additional 30 seconds, bringing the total startup time for a 4 MB corpus to just over one minute. Then, additional similarity computations take around 5-10 seconds. To further enhance performance, we added a small cache of similarity vectors; this permits users to get instant response when requesting a similarity computation they had done recently.

3.4 Scaling Experiments

Exploring ways to scale TELLTALE was one of the main reasons for building the prototype. We considered several mechanisms that would allow TELLTALE to handle multi-gigabyte corpora. These algorithms included reduction of the metadata associated with the corpus, and distribution of the corpus across several cooperating processors.

Our experiments to date have focused on the second method for scaling TELLTALE: distributing the metadata and processing across several processors. As mentioned in Section 3.2, the postings list occupied the largest fraction of memory in the TELLTALE prototype, while the corpus centroid was considerably smaller. By not keeping a postings list for each document, a single processor running the prototype could compute the centroid for a far larger corpus than it could index. This centroid could then be distributed to several CPUs, each of which maintains the postings list for a subset of the documents in the corpus.

We modified the `Tcl` interface to the TELLTALE prototype using the `dp` package [dp], which allows `Tcl` applications to communicate via Unix sockets. This package allowed `Tcl`-controlled applications to act as servers that could receive commands, process them, and

return responses in a way similar to Unix RPC. This process was entirely transparent to the application; only the client sending commands knew that they were being sent over a socket. This modification allowed us to write clients that did not include the TELLTALE prototype back end. Since their sole purpose was to send commands to TELLTALE servers, the clients did not need any TELLTALE-specific code.

We used 40 MB of articles from the *Wall Street Journal* for our experiments. First, we generated the corpus centroid (for 5-grams) on a single Sparc 10; this procedure processed about 100 KB of data per second, requiring 400 seconds to digest the entire corpus. The centroid was then written to disk. We then started prototypes on several different machines, and loaded the entire centroid into each one. Next, each machine read in a subset of the documents from the corpus, generating the necessary postings lists. Since each instance of the prototype was only responsible for a portion of the 40 MB corpus, they were able to fit the postings list in memory. Because each server used the corpus-wide centroid, its reponse to a similarity query against a piece of text was the same as if it had contained the postings list for the entire corpus. Of course, this method only worked because the "base" document had a postings list on each server – thus, each server generated such a list for the query text as part of the query processing.

Using this structure, a client was able send similarity queries to each of the servers and merge the responses. This method gave the same results as if the query were processed by a single instance of the prototype with a very large corpus. Because of the parallelism, though, the per-node memory requirement was reduced, as was the response time for the query. Our preliminary experiments showed the reduction in response time to be linear with the number of processors, though the reduction in memory usage was less than linear because of the overhead of storing the centroid multiple times.

We have not yet run experiments on reducing the size of the postings list; possible directions for this work are discussed in Section 3.5.

3.5 Future Prototype Experiments

The `dp` package allows TELLTALE to handle larger corpora by distributing them across several workstations. However, it does not increase the capacity of a single computer. To address this problem, we are pursuing three different approaches, each of which would allow a single workstation running TELLTALE to manage hundreds of megabytes of textual data. Combining single-node scaling with `dp` will allow a TELLTALE system to perform similarity computations over a corpus with a gigabyte or more of text.

The most obvious approach to indexing large amounts of text is to move the hash table from memory to disk. While a workstation memory is usually smaller than 100 MB, multi-gigabyte disks are both common and inexpensive. Storing the hash table on disk will allow a single workstation to index 200 MB of text using approximately 1.5 GB of disk, at a 1996 cost of under $400. Since the basic data structures remain unchanged, this approach will produce the same similarity results as TELLTALE would get using an in-memory hash table. The only possible drawback is poor performance — computing similarity would involve reading the entire hash table "bucket" for each n-gram in the query. While many buckets will be small, others (for example, " the ") may require reading several megabytes of data. Retrieving the on-disk data and processing it will likely require several minutes;

however, this is the only one of the three methods that is guaranteed to generate the same list of linked documents as the original TELLTALE system did.

The second method to improve TELLTALE's scalability is to build the in-memory hash table using a subset of the n-grams from each document. Many of the n-grams in a document do not help to distinguish it from other documents in a corpus, either because they are too common or because they only occur a few times in the entire corpus. By eliminating the postings for these n-grams, we can reduce the amount of memory used by the hash table. In addition, n-grams that overlap might be removed because they convey little additional information. Reducing the number of postings from 1000 to 50-100 per document will shrink the hash table's memory usage by a factor of 10-20, increasing the number of documents that can be indexed by the same factor. However, this reduced hash table cannot be used to generate similarity scores because similarity depends on *all* of the n-grams in a document. Thus, this method will use two passes. The first pass will use the reduced hash table to find documents that are likely to be relevant. This pass will have high recall, but low precision. The second pass will increase precision by scanning in *all* of the n-grams for each document identified in the first pass and computing their similarity to the query text. So long as the centroid is that of the full corpus, the similarity values computed will be the same as if the similarity had been computed over the entire corpus. However, the first pass may miss some documents that should have been included near the top of the similarity list. This approach should be faster than the first method, but it may be less accurate — finding good algorithms for choosing n-grams for the reduced hash table and selecting appropriate documents in the first pass will be crucial to ensuring results similar to those of TELLTALE with a full in-memory hash table.

Using signature vectors [WMB94] to characterize documents is the final method we will be using. Each document will have a long (2 Kbit or more) bit vector associated with it. Bits in this vector will be set by hashing n-grams to find an offset. As with the reduced hash table method, we must choose the n-grams that are appropriate for storing in this vector. Since the vector will require a fixed space regardless of how many bits are set, more n-grams can be in this list. With a 2 Kbit vector, the n-gram list might contain 20,000 entries. This allows more n-gram postings to be associated with each document at the cost of reduced precision. As with the reduced hash-table approach, this method will make two passes over the corpus. The low-precision first pass will identify documents "likely" to be interesting, and the second pass will proceed as for the reduced hash table. Since the first pass is not precise to begin with, the slight reduction in precision from signature vectors should not cause too many problems, and the increase in recall from storing more n-grams per document may reduce the number of documents that need to be identified for the second pass.

The three approaches will be compared in two areas: difference in results from in-memory TELLTALE, and performance. Since the disk-based hash table will produce the same results as the original TELLTALE implementation, it can be used as the benchmark for the other two approaches. The result of these experiments will show the tradeoff between speed and accuracy for the three approaches.

4 Summary and Conclusions

TELLTALE is a hypertext tool that dynamically computes links between related items of text using novel full text search techniques. TELLTALE's uses sliding n-gram based indexing and scoring techniques to provide robustness to degraded text and to remove dependencies on language. Linking mechanisms in TELLTALE tolerate high error rates (up to 30 percent of the characters), outperforming existing systems in its tolerance to high character error rates. TELLTALE is versatile and truly dynamic in the selection and calculation of links. A combination of three link types, Similarity, Lookup, and Disambiguated Lookup, provide the user with a collection of methods from which he can choose based on his immediate need. The dynamism in TELLTALE's link computation and link selection methods is augmented by link anchor selection and novel highlighting techniques. These dynamic linking and highlighting capabilities work together to give the TELLTALE dynamic hypertext environment its navigational and browsing capabilities. In addition to the high resistance to garbles provided by TELLTALE's linking and highlighting mechanisms, these mechanisms also provide TELLTALE with the necessary robustness to analyze and display multiple languages. Further, TELLTALE's navigation and browsing tools provide additional value and capability to that provided by traditional full text search and retrieval techniques. Finally, the ability to tolerate degraded and multilingual text in TELLTALE represent capabilities not shared by traditional full text retrieval tools.

5 Acknowledgments

Special thanks go to Marc Damashek for the invention of the sliding n-gram scoring technique which is so prominently featured in the TELLTALE dynamic hypertext system; to Jonathan Cohen for his work on n-gram based highlights and his numerous insightful comments during the course of this research; to Tom Nelson who slaved over the code for the original TELLTALE prototype; and to Bill Rye for his many speedups and enhancements to the production version of TELLTALE.

References

[ACR+90] M. Aboud, C. Chrisment, R. Razouk, F. Sedes, and C. Soule-Dupuy. Querying a hypertext information retrieval system by the use of classification. *Information Processing and Management*, 29(3):387–396, 1990.

[Cav93] W. B. Cavnar. N-Gram-Based text filtering for TREC-2. In Donna Harman, editor, *Proceedings of TREC-2: Text Retrieval Conference 2*, Gaithersburg, MD, 1993. National Institute of Standards and Technology.

[CCA89] Donald B. Crouch, Carolyn J. Crouch, and Glenn Andreas. The use of cluster hierarchies in hypertext information retrieval. In *Hypertext '89 Proceedings*, pages 225–237. ACM Press, November 1989. Pittsburgh, PA, Nov 5-8.

[Coh95] Jonathan Cohen. Highlights: Language- and domain-independent automatic indexing terms for abstracting. To appear in JASIS, 1995.

[Con92] The Unicode Consortium. *The Unicode Standard: World Wide Character Encoding*. Addison-Wesley, Redwood City, CA, 1992.

[CT87] W. B. Croft and R. Thompson. I^3R: A new approach to the design of document retrieval systems. *Journal of the American Society for Information Science*, 38:389–404, 1987.

[CT89] W. B. Croft and H. Turtle. A retrieval model for incorporating hypertext links. In *Hypertext '89 Proceedings*, pages 213–224. ACM Press, November 1989. Pittsburgh, PA, Nov 5-8.

[Dam95a] Marc Damashek, 1995. U. S. Patent Number 5,418,951.

[Dam95b] Marc Damashek. Gauging similarity with N-Grams: Language-independent categorization of text. *Science*, 267:843–848, 10 February 1995.

[DM85] R. D'Amore and C. Mah. One-time complete indexing of text: theory and practice. In *Proceedings 8th International ACM Conference on Research and Development in Information Retrieval*. ACM Press, 1985.

[dp] The `dp` package for Tcl/Tk. Available for ftp from ftp://aud.alcatel.com/tcl/extensions/tcl-dp3.3b1.tar.gz.

[EE68] Douglas C. Engelbart and W. K. English. A research center for augmenting human intellect. In *Proceedings of the Fall Joint Computer Conference*. AFIPS Press, Montvale, NY, 1968.

[FC89] Mark E. Frisse and Steven B. Cousins. Information retrieval from hypertext: Update on the dynamic medical handbook project. In *Hypertext '89 Proceedings*. ACM Press, November 1989. Pittsburgh, PA, Nov 5-8.

[Har93] Donna Harmon, editor. *TREC-2- Text REtrieval Conference-2*. National Institute of Standards and Technology, August 1993.

[Knu73] Donald E. Knuth. *Sorting and Searching*, pages 561–562. Addison Wesley, 1973.

[Nel88] Theodor H. Nelson. Managing immense storage. *BYTE*, 13(1):225–238, January 1988.

[Nie90] Jakob Nielsen. *Hypertext and Hypermedia*. Academic Press, San Diego, CA, 1990.

[Pea94] Claudia E. Pearce. *A Dynamic Hypertext Environment Through n-gram Analysis*. PhD thesis, University of Maryland Baltimore County, 1994.

[Pea95] Claudia E. Pearce. Dynamic hypertext links for highly degraded data in TELLTALE. In *Fourth Annual Symposium on Document Analysis and Information Retrieval*, pages 89–106. Information Science Research Institute, University of Nevada Las Vegas, University of Nevada, 4505 Maryland Parkway, Box 454021, Las Vegas, Nevada 89154-4021, 1995.

[SM83] Gerard Salton and Michael McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill Book Company, 1983.

[Sue79] C. Y. Suen. n-gram statistics for natural language understanding and text processing. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-1(2):164–172, 1979.

[Wel95] Brent B. Welch. *Practical Programming in Tcl and Tk*. Prentice-Hall, Inc., 1995.

[Wil79] P. Willette. Document retrieval experiments using indexing vocabularies of varying size. II. hashing, truncation, diagram and trigram encoding of index terms. *Journal of Documentation*, 35:296–305, December 1979.

[WMB94] Ian H. Witten, Alistair Moffat, and Timothy C. Bell. *Managing Gigabytes*. Van Nostrand Reinhold, 1994.

[YGH82] E. J. Yannakoudakis, P. Goyal, and J. A. Huggil. The generation and use of text fragments for data compression. *Information Processing and Management*, 18(1):15–21, 1982.

[ZPZ81] E. M. Zamora, J. J. Pollock, and A. Zamora. The use of trigram analysis for spelling error detection. *Information Processing and Management*, 17(6):305–316, 1981.