This paper appeared in the Journal of Digital Information, an online refereed journal, January 2000.

Performance and Scalability of a Large-Scale N-gram Based Information Retrieval System

Ethan Miller, Dan Shen, Junli Liu, and Charles Nicholas
University of Maryland Baltimore County
{elm,dshen,jliu,nicholas}@csee.umbc.edu

ABSTRACT

Information retrieval has become more and more important due to the rapid growth of all kinds of information. However, there are few suitable systems available. This paper presents a few approaches that enable large-scale information retrieval for the TELLTALE system. TELLTALE is a dynamic hypertext information retrieval environment. It provides full-text search for text corpora that may be garbled by OCR (Optical Character Recognition) or transmission errors, and that may contain multiple languages by using several techniques based on n-grams (n character sequences of text). It can find similar documents against a 1KB query from 1G text data in 45 seconds. This remarkable performance is achieved by integrating new data structures and gamma compression into the TELLTALE framework. This paper also compares several different types of query methods such as TF/IDF and incremental similarity to the original technique of centroid subtraction. The new similarity techniques give better performance but less accuracy.

1 Introduction

Scientists, researchers, reporters and the rest of humanity all need to find documents relevant to their needs from a growing amount of textual information. For example, the World Wide Web currently has over 320 million indexable pages containing over 15 billion words [1], and is growing at an astonishing rate. As a result, information retrieval (IR) systems have become more and more important. However, traditional IR systems for text suffer from several drawbacks, including the inability to deal well with different languages, susceptibility to optical character recognition errors and other minor mistakes common on the WWW, and reliance on queries composed of relatively few keywords.

The TELLTALE system information retrieval system [2] was developed to address these concerns. It uses n-grams (sequences of *n* consecutive Unicode characters) rather than words as the index terms across which retrieval is done. By using statistical IR techniques, the TELLTALE system can index text in any language; the current version has been used unmodified for documents in English, French, Spanish, and Chinese. Additionally, n-grams provide resilience against minor errors in the text by allowing matches on portions of words rather than requiring the entire word to match. A third advantage for TELLTALE is that users need not learn query languages and query optimization methods. Instead, they can simply ask for "more documents like the one I've got know," allowing for greater ease-of-use.

Previously, however, the TELLTALE system was unable to index large volumes of text. While traditional word-based IR systems have a bevy of tricks and tools at their disposal, many of these methods must be modified or discarded when used in n-gram based systems. This paper describes our successful efforts to apply traditional techniques to an n-gram based IR system, showing which methods work, which don't, and describing new techniques we implemented for n-gram

based retrieval. By using our techniques, we were able to construct an n-gram based IR engine that permitted full-document queries against a gigabyte of text. Both the size of the corpus and the speed with which our methods operate allow such a query to complete in a few seconds on inexpensive PC-class hardware. These improvements represent a hundred-fold increase in corpus size over previous n-gram-based efforts. Moreover, the compression techniques we adapted from word-based IR systems reduced the size of the index file from seven times larger than the text corpus to approximately half the size of the original text, a fifteen-fold improvement.

2 Background

Our work builds on a large body of research in information retrieval covering both traditional word-based IR systems and systems based around n-grams. In this section, we discuss some of the most relevant previous work. Of course, a full treatment of prior work in information retrieval would require a full book (if not more), and such texts exist [3,4].

2.1 Word-based information retrieval systems

There are many information retrieval systems in existence, but space prevents us from mentioning more than a small selection. We will only discuss four of them: INQUERY, MG, SMART and TELLTALE; the reader is referred to [3] and [4] for a more detailed treatment of information retrieval systems.

2.1.1 INQUERY

The INQUERY system is the product of the Center for Intelligent Information (CIIR) at the University of Massachusetts at Amherst. INQUERY [5] uses a probabilistic model based on a Bayesian network [6] that considers the probability that a term or concept appears in a document, or that a document satisfies the information need. Because a Bayesian network is a graphical model that encodes probabilistic relationships among variables of interest, it makes a good framework for this style of model. INQUERY has two parts: a document net and a query net. The document net is static for a given collection. Nodes representing documents are connected to nodes representing terms. Thus, INQUERY can calculate, given a document, the probability that a particular term is instantiated. The query net is constructed by connecting terms in the query to nodes representing how those terms should be combined. For example, the probability an “AND” node is satisfied given a number of terms would simply be the product of the individual probabilities of appearance for each term. These combination terms could themselves be combined to represent a user's entire information need. To perform retrieval, the system connects these two networks together, and can thus calculate the conditional probability that the information needed with each given document. The system then ranks the documents by this probability.

The CIIR has some very useful web pages on their system (`http://ciir.cs.umass.edu/demonstrations/InQueryRetrievalEngine.shtml`). They give an overview of the features of the system, and a number of different domains to try it out. They have set up their system to search a WWW page database, but it only contains 100,000 URLs, as opposed to millions for a system like Lycos. Also, while INQUERY has the capability to do sophisticated queries, the interface in the demos requires the user to know the exact formulations.

2.1.2 MG (Managing Gigabytes)

MG (Managing Gigabytes) [7] is a full-text retrieval system that gives instant access to a collection of documents while requiring far less storage space than the original documents. It is a word-based information retrieval system, using words as the basic terms on which matching and lookup are performed. MG uses a vector space model that represents documents and queries as vectors of term frequencies. This approach weights entries in the vectors to give emphasis to terms that exemplify meaning and are useful in retrieval.

The MG system has two main parts: a program that turns a collection of documents into a compressed and indexed full-text database, and a program that services several types of interactive queries for words that appear in the documents. By default, queries are Boolean and are constructed using a collection of terms linked by the Boolean operators AND, OR, and NOT. MG also supports ranked queries, which take a list of terms and use frequency statistics from the documents to determine which of the terms should be given the most weight when they appear in a document. Additionally, MG supports approximate-ranked query which is similar to ranked query but is only an approximation, providing higher speed at the cost of worse retrieval.

While the engine behind MG is quite powerful, it has only a rudimentary command-line interface because it is only a research prototype. Its main use has been as a testbed for research involving large-scale information retrieval.

2.1.3 SMART

SMART [8], developed by Gerard Salton and his students at Cornell University, also uses a vector space model for representing documents. SMART performs automatic indexing by removing stop words (words that are too common to be useful in distinguishing between documents) from a predetermined list, stemming (the process of removing prefixes and suffixes from words in a document or query in the formation of terms in the system's internal model) via suffix deletion, and term weighting. Given a new query, SMART converts it to a vector, and then uses a similarity measure to compare it to the documents in the vector space. SMART ranks the documents, and returns the top n documents, where n is a number determined by the user. SMART can perform relevance feedback, a process of refining the results of retrieval using a given query, based on the results of the retrieval process.

The disk space requirements for the indexed collection require roughly 0.8 times the space of the text version. This space includes a dictionary, display information, and both an inverted file and a sequential representation of the indexed documents. While this system is relatively old, it includes many modern techniques such as stemming and stop word removal. As will be described shortly, TELLTALE mimics both stemming and stop word removal by using statistical techniques based on n-grams.

2.2 N-gram based information retrieval using TELLTALE

TELLTALE [2,9] is a dynamic hypertext environment that provides full-text information retrieval from text corpora using a hypertext-style user interface. The most important difference between TELLTALE and the systems described in the previous sections is that TELLTALE is n-gram-based while the others are word-based. Because of its use of n-grams, TELLTALE has some unique features including language independence and garble tolerance. These features and others will be discussed in this section.

2.2.1 N-gram basics

An n-gram [10] is a character sequence of length *n* extracted from a document. Typically, *n* is fixed for a particular corpus of documents and the queries made against that corpus. To generate the n-gram vector for a document, a window *n* characters in length is moved through the text, sliding forward one character at a time. At each position of the window, the sequence of characters in the window is recorded. For example, the first four 5-grams in the sentence " character sequences…" are " char", "chara", "harac" and "aract". In some schemes, the window may be slid more than one character after each n-gram is recorded.

The concept of n-grams was first discussed in 1951 by Shannon [11]. Since then the concept of n-grams have been used in many areas, such as spelling-related applications, string searching, prediction and speech recognition.

Most information retrieval systems are word-based because there are several advantages for word-based systems over n-gram based systems. First, the number of unique words is smaller than unique n-grams (for $n>3$) in the same text corpus, as shown in Figure 1. As a result, the index for

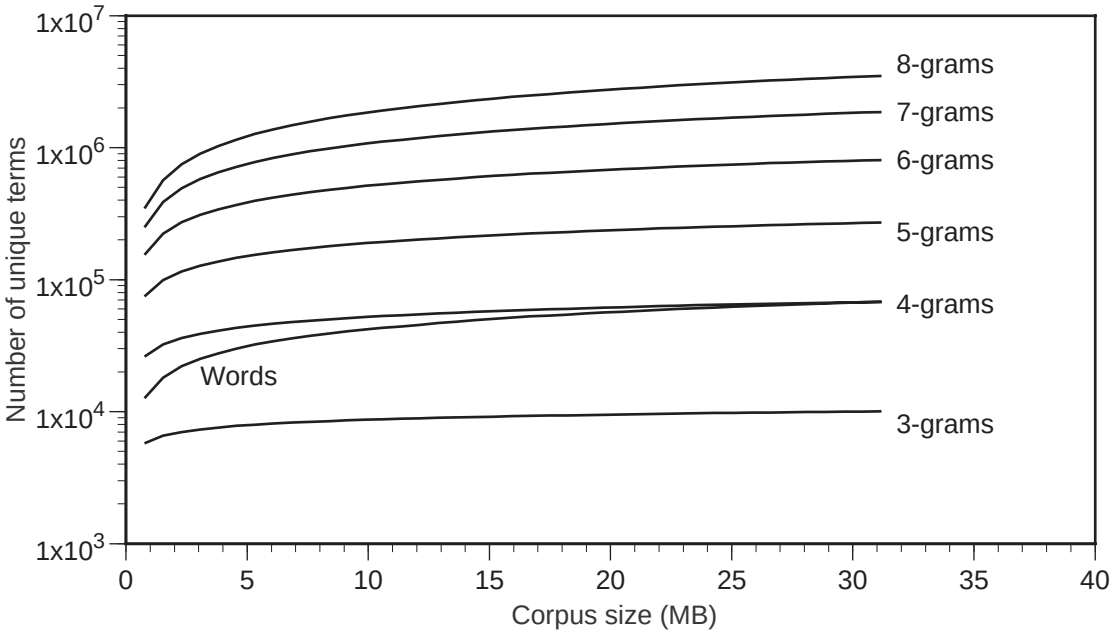

Figure 1. Number of unique terms (words and n-grams) in corpora of varying sizes.

an n-gram-based system will be much larger than that of a word-based system. Second, stemming techniques can be used in word-based systems but not in n-gram-based systems. Stemming is the process that removes prefixes and suffixes from words in a document or query in the formation of terms in the system's internal model. This is done to group words that have the same concept meaning, such as "walk", "walked", "walker" and "walking," freeing the user from needing to match the particular form of a word in a query and document. Stemming also reduces the number of unique terms to be indexed. Third, in word-based system, a table can be established for each word to list all of its synonyms. By doing this, if in the query there is a word "home," according to that table, the system will also retrieve the documents containing the word "house." Finally, most word-based systems use stop words. Since stop words appear in most documents, and are thus not

helpful for retrieval, these words are usually removed from the internal model of a document or query.

At the same time, there are several advantages for using n-grams. First, the system can be garble tolerant by using n-gram as basic term. If a document is scanned using OCR (Optical Character Recognition), there may be some misread characters. For example, suppose “character” is scanned as “claracter”. The word-based system will not be able to match this word because it is misspelled, but an n-gram based system will still match the other n-grams such as “aract”, “racte”… and take their frequency into account. From this we can see that by using n-gram technology system can be garble tolerant.

Second, by using n-grams the system can achieve language independence. In a word-based information retrieval systems there is language dependency. For example, in some Asian languages, different words are not separated by spaces, so a sentence is composed of many consecutive characters. Grammar knowledge is needed to separate those characters into words, which is a very difficult task to perform. Using n-grams, the system does not need to separate characters into words.

Additionally, n-gram based systems do not use stop words. This is because the number of unique n-grams in a document is very big and distribution is very wide. There are few n-grams that have high frequency. From Ekmekcioglu’s research [12], stop words and stemming are superior for word-based system but are not significant for an n-gram based system.

2.2.2 Document similarity computation

Similarity is the measure of how alike two documents are, or how alike a document and a query are. In a vector space model, this is usually interpreted as how close their corresponding vector representations are to each other. One way of determining this is to compute the cosine of the angle between the vectors.

In TELLTALE, each document is represented by a vector of n-grams. That is, a particular document i is identified by a collection of n-grams $ngram_1, ngram_2, \ldots$. For each n-gram, a count c_{ik} records how many times $ngram_k$ occurred in document i. The frequency f_{ik} of $ngram_k$ is its count c_{ik} normalized by the total number of n-grams m_i in document i, or c_{ik}/m_i. The weight of each n-gram is the difference between f_{ik} and the average normalized frequency over all documents $a_k = \sum_i f_{ik}$ for $ngram_k$. This provides a weight for each n-gram in a document relative to the average for the collection. A document is then represented as a vector $\vec{d}_i = (d_{i1}, d_{i2}, \ldots)$, where the individual elements $d_{ik} = (c_{ik}/m_i) - a_k$ have been normalized and the n-gram’s average value has been removed. The similarity between two document vectors $\vec{d}_i$ and $\vec{d}_j$ is then calculated as the cosine of the two representation vectors,

$$SIM_c(\vec{d}_i, \vec{d}_j) = \frac{\sum_k (d_{ik} d_{jk})}{\sqrt{\sum_k d_{ik}^2} \sqrt{\sum_k d_{jk}^2}}.$$

Since $\cos(\theta) = 1$ when the vectors are colinear and $\cos(\theta) = 0$ when the vectors are orthogonal, the similarity measure can be taken as the cosine of the angle between the document and query vector — the larger this cosine value, the greater the similarity.

2.2.3 Multilingual operation

The language independence of TELLTALE is achieved by its n-gram techniques, unique algorithms, Unicode [13] and display system based on Tcl/Tk [14]. As mentioned in Section 2.2.1, using n-grams can eliminate language-dependent features because the program need not know about grammar for stemming, stop words, or even matters as simple as where to break individual words. In languages such as German, for example, words are often built from smaller words without including spaces between them. A German IR system would thus have to know how to break up a long word, or risk missing similarities between long compound words. For TELLTALE, however, this is not a problem because the text is broken into relatively short sequences of characters without knowledge of where individual words begin or end.

Second, the algorithms used in TELLTALE are independent of the language texts to be analyzed. We have found that our algorithms work well not only for English, but also for other languages such as Spanish and Chinese. We do not rely upon particulars of the English language to attain good retrieval accuracy.

Third, TELLTALE uses Unicode to represent a character. Unicode is a 16-bit encoding standard in which each character is represented in two bytes. This encoding is necessary because many Asian languages require 16 bits for each character; one byte has no meaning in those languages. Thus, the use of Unicode in the algorithms is necessary to achieve language independence. While it may be necessary to convert a document from “native” format into Unicode, such conversion is mechanical, and would be unnecessary if all documents were encoded in Unicode.

TELLTALE’s fourth advantage is the ability to easily include non-English fonts in Tcl/Tk. This allows us to quickly build a system that has the ability to display a variety of fonts, such as Russian, Hebrew, and Chinese. Using this ability, the system can display documents in their native scripts.

2.2.4 TELLTALE interfaces

Because TELLTALE has a Tcl/Tk interface, implementing the user interface was relatively easy and fast. A sample view of the interface is shown in Figure 2. The interface has several main areas, as noted in Figure 2. These include the main document window (A), the query text window (B), the document list (C), the status area (D), and various controls (E). Additionally, the interface supports the execution of any Tcl/Tk commands via a small floating window.

When TELLTALE is first started, a user can initiate a search either by listing all documents and selecting the one she wants or by entering some text into the query window and finding documents “similar” to the entered text. In response, TELLTALE shows a ranked list of all documents that satisfy minimum similarity criteria specified by the controls. If the user clicks on one of the documents in the list, the full text of the document will be displayed in the document window. The searching process can be continued by either looking for documents similar to the one in the document window or by cutting some of the document text and placing it into the query window. Of course, other text can be used in the query in addition to or instead of text from the displayed doc-

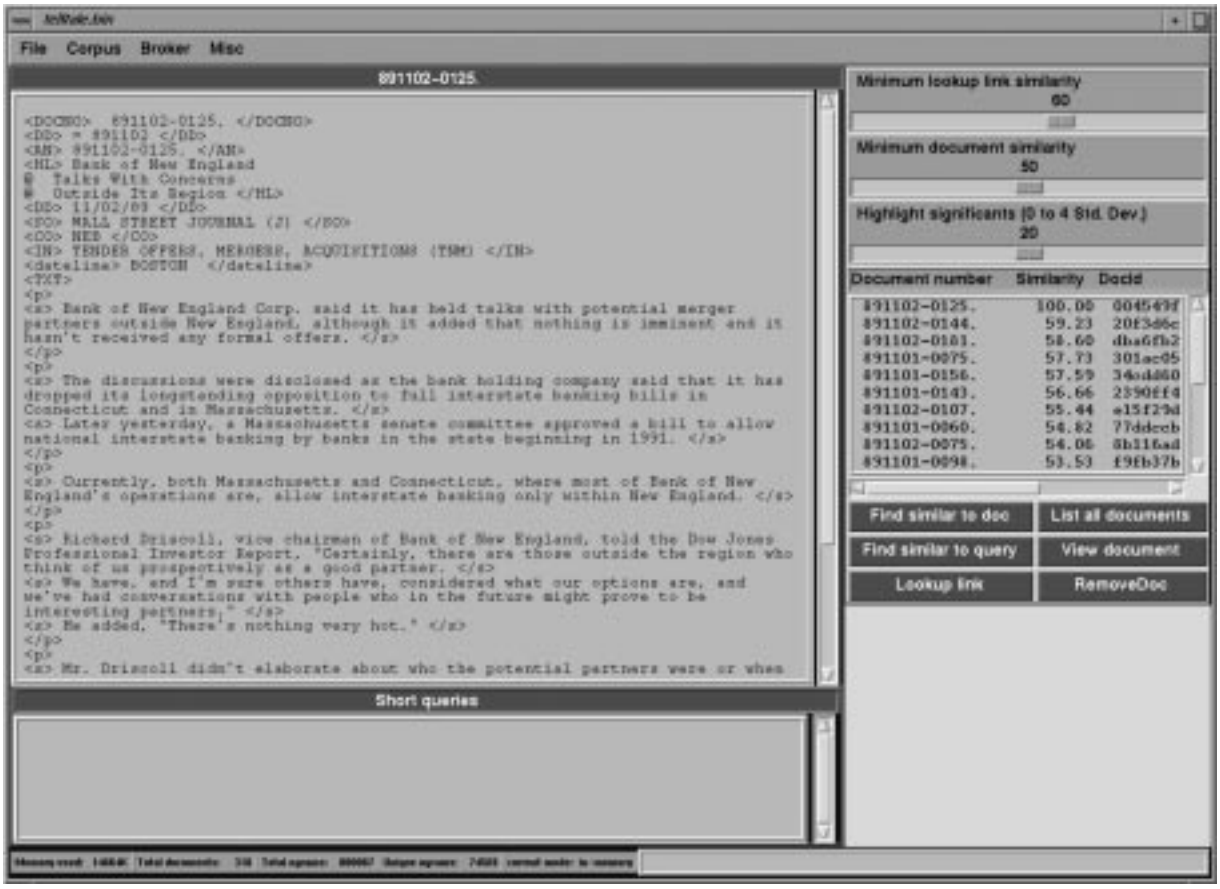

Figure 2. TELLTALE's graphical user interface.

ument. This method allows the user to incrementally approach the documents that match her needs. The user may also limit the documents selected by changing minimum similarity thresholds.

The display also includes a good deal of status information about the TELLTALE system. This information includes corpus statistics such as the amount of text indexed, number of unique n-grams, and number of postings. It also includes memory usage statistics; these are particularly useful when trying to ensure that TELLTALE fits into RAM for best performance.

3 Approaches to large-scale retrieval in TELLTALE

Because TELLTALE is n-gram based and the number of n-grams in a document is much larger than the number of words in the same document, the index for TELLTALE is much larger than that of a word-based system. Thus, building a large-scale n-gram based retrieval system is a technical challenge. In this section, we describe the way in which we increased the capacity of TELLTALE to handle gigabyte-sized queries in a reasonably short time.

All of the performance figures reported in this paper were measured by running on a Silicon Graphics Origin200 with two 180 MHz MIPS R10000 processors, 256 MB of main memory, and 32 KB each of instruction and data cache. While this may seem an impressive machine, it is cur-

rently possible to purchase a more powerful machine relatively inexpensively from commodity PC vendors. Thus, we expect that our techniques will be applicable to those who can't afford large-scale computers as well as those who can.

3.1 Textual data used in experiments

To allow practical comparison of various algorithms and techniques, we performed our experiments on real-world collections of data obtained from TIPSTER [15], a DARPA (Defense Advanced Research Projects Agency)-funded program of research and development in information retrieval and extraction. The TREC [16] (Text REtrieval Conference) is part of the TIPSTER Text Program, and provides a very large text data collection. Three types of text data from TREC are used in this paper: a selection of computer magazines and journals published by Ziff-Davis (ZIFF), the Associated Press newswire (AP), and the Wall Street Journal (WSJ). Here we use: ZIFF1 to represent the collection from 1989's ZIFF, ZIFF2 to represent the text data from 1988, AP1 to represent the text data from 1989's AP, AP2 to represent the data from 1988 and WSJ to represent the data from 1989's Wall Street Journal.

	ZIFF1 (1989)	**ZIFF2 (1988)**	**AP1 (1989)**	**AP2 (1988)**	**WSJ (1989)**
Documents	75,029	56,903	83,719	78,789	12,046
Unique 5-grams	562,492	498,653	499,807	478,519	268,810
Total 5-grams	185,159,683	134,110,587	202,636,790	186,822,009	31,863,179
Size (MB)	257	180	260	240	40

Table 1. Statistics for the document collections.

Every corpus is composed of tens or hundreds of individual files, each of which averages one megabyte long and contains one or more documents. Individual documents within a file are separated by SGML (Standard Generalized Mark-Up Language) tags. The overall characteristics of all of the corpora on which we ran experiments are summarized in Table 1. Note that the figures for unique and total n-grams are calculated for $n = 5$; we used this value for n in all of our experiments. XXX - why did we choose 5?

3.2 Data structures

The data structures used in TELLTALE are similar to those used in other information retrieval systems, but with modifications to make them efficient for managing tables of n-grams that are considerably larger than the word-based tables used elsewhere. Over the course of our work, structures evolved from relatively simple solutions to more advanced mechanisms that allow TELLTALE to index gigabytes of textual data, a hundredfold increase over its original capacity.

3.2.1 In-memory data structures

The first set of data structures in TELLTALE are in-memory data structures similar to those used in traditional word-based IR systems. The three hash tables represent all of the information gathered from the raw text scanned into TELLTALE. There is one hash table for n-grams, one for document information, and a third detailing information about the files that contain the documents.

The relationship between these three hash tables is shown in Figure 3. While all three data structures are crucial to TELLTALE, the n-gram hash table and associated postings lists consume by far the largest fraction of memory.

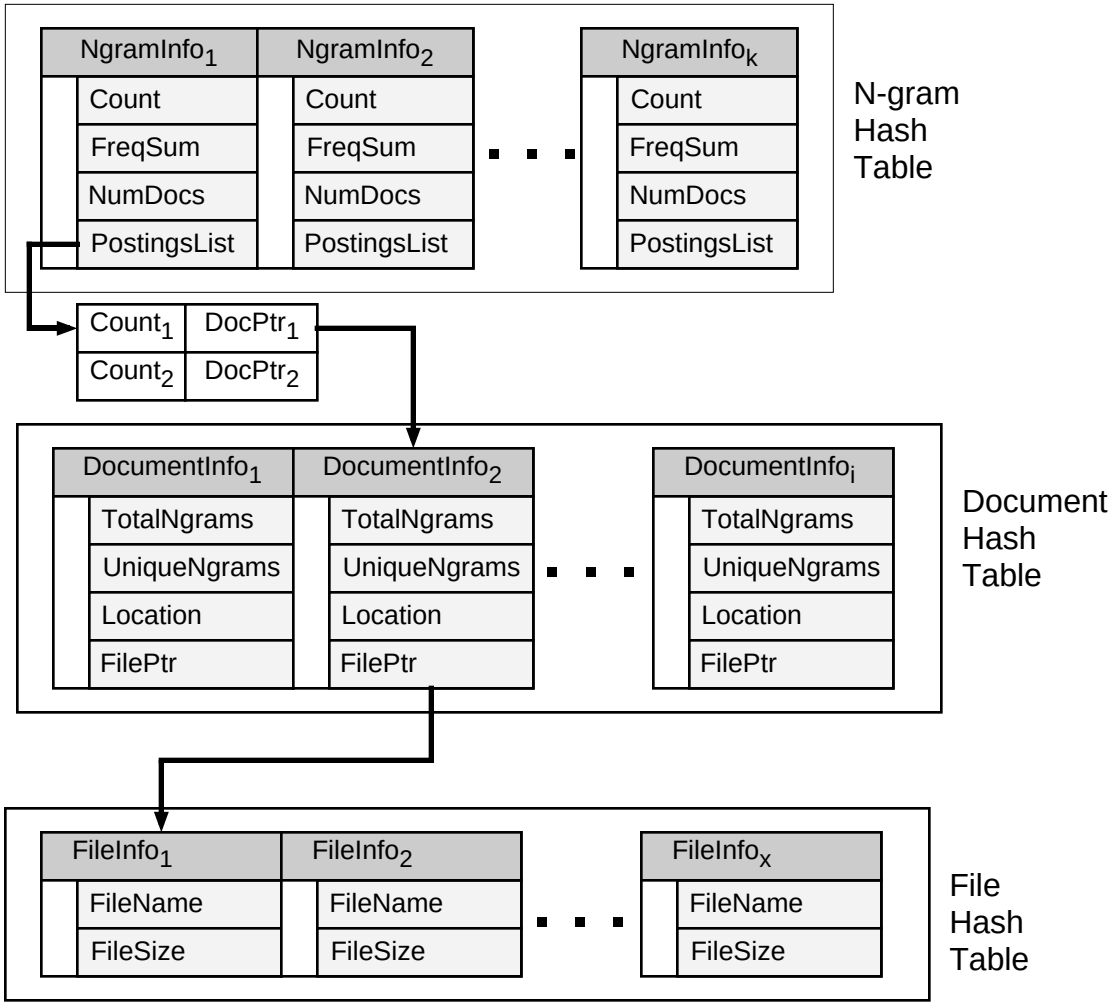

Figure 3. Relationships between hash tables in TELLTALE.

The file table provides a link between documents and the files that contain them. While it would be possible to fold this information into the document hash table, storing it separately results in a large memory savings at little cost because file names are long. For example, a corpus with 500,000 documents at 2 KB apiece might pack an average of 500 documents into each 1 MB file. If file names average 60 bytes in length, the file table requires 60 + 4 = 64 bytes of data and 4 bytes of overhead per file for a total of just 64 KB of storage. On the other hand, storing a file name with each document requires over 3 MB. Thus, large corpora benefit greatly from the savings provided by a separate file table. Additionally, this structure works well for systems that don't use traditional file systems. For example, a system might optimize performance for access via the WWW by consolidating references to a single URL together; pointing all documents from a particular URL to the same place would make retrieval and caching simpler.

The document table contains a great deal of information about each individual document. In addition to the usual information such as document length and location (file and offset), the document table contains a document serial number, which is allocated from a single integer incremented for each document scanned in. Thus, the document serial number is guaranteed unique in a given corpus. The document hash table also stores precomputed values for each document to assist in rapid similarity calculations. As will be described in Section XXX, precomputed per-document values greatly reduce similarity calculation times for some similarity measures.

Additionally, each document in the hash table contains an identifier generated by cryptographic hash. In the current version of TELLTALE, this hash is generated by MD5 [17], though there is no reason that would prevent switching to a different algorithm such as SHA [18]. This document ID is probabilistically unique across all corpora, with a chance of collision below 10^{-9} even for billions of documents. Thus, it can be used to uniquely refer to a document in a massive environment that might contain tens or hundreds of TELLTALE engines. The document ID can also be used to remove duplicate documents from corpora; since the ID is based on the document's content, identical documents will have identical IDs. Memory usage for the document hash table is approximately 48 bytes per document, most of which is used to store the 128-bit (16 byte) document ID. In a system with 10^6 documents, this means that 48 MB will be required to store the document hash table.

The n-gram hashtable is the central data structure in TELLTALE and, when the postings are included, the one that requires the most memory. This data structure is the one that is hardest to optimize for n-grams rather than documents because of the far greater number of both unique n-grams in the corpus and unique n-grams in a document. For example, a typical 1 MB file from the WSJ corpus has XXX documents with XXX postings — one posting for each different n-gram in a document. When terms are words, however, the same file has just XXX word postings, a reduction of XXX. It is this difference that makes it more difficult to build IR systems using n-grams rather than words as terms.

In this version of TELLTALE, each posting contains three things: a normalized frequency for n-gram *k* in document *i*, a pointer to document *i*, and a pointer to the next posting for n-gram *k*. Thus, each posting requires 12 bytes on a machine that supports 32-bit floating point numbers and 32-bit pointers. It is the space required for postings that consumes the lion's share of memory for corpora of almost any size. In typical documents, the number of unique 5-grams is about 65%-75% of the total number of 5-grams, so a 4 KB document will have 2500 - 3000 unique 5-grams, resulting in $12 \times 3000 = 36000$ bytes of storage. On the other hand, the word count for such a document will total perhaps 800 words with perhaps 400 different words — a reduction of an order of magnitude. It is this difference that makes building n-gram based IR difficult.

To obtain good performance on cosine similarity using these data structures, we broke the similarity formula down as follows:

$$SIM_c(\vec{d}_i, \vec{d}_j) = \frac{\sum_k (d_{ik} d_{jk})}{\sqrt{\sum_k d_{ik}^2} \sqrt{\sum_k d_{jk}^2}}$$

$$= \frac{\sum_k ((f_{ik} - a_k)(f_{jk} - a_k))}{\sqrt{\sum_k d_{ik}^2} \sqrt{\sum_k d_{jk}^2}} .$$

$$= \frac{\sum_k (f_{ik} f_{jk}) + \sum_k (f_{ik} a_k) + \sum_k (f_{jk} a_k) + \sum_k a_k^2}{\sqrt{\sum_k d_{ik}^2} \sqrt{\sum_k d_{jk}^2}}$$

Note that, in the final equation, all of the terms with the exception of the first term in the numerator can be precalculated. The remaining term is non-zero only when a term appears in both the query and a document in the corpus. Thus, we can precompute all of the "constant" expressions in the formula for each document, and only need compute the sum of the term frequencies on the fly. Because there are relatively few n-grams in common between any pair of documents, this calculation can be done quickly once the precomputation time has been invested.

Based on our experiments with sample data, scanning in 10 MB of text data requires 88.9 MB for the resulting data structures. Using the data structures described above, TELLTALE can compute the similarity against a 1 KB document in two seconds. The performance is good, but TELLTALE consumes too much memory.

The problem of memory consumption becomes worse as the corpus grows. In particular, the size of the posting list grows dramatically while the other data structures grow considerably more slowly. This is shown in Figure 4, which displays both the total memory consumed by each data structure and the percentage of memory consumed by each data structure for varying corpus sizes using 5-grams. As Figure 4 shows, by far the largest consumer of space is the postings list. Even for a small corpus of 1 MB, the postings list consumes 75% of the space. For larger corpora, the contribution of all other data structures shrinks further. By the time the corpus has reached 1 GB, the postings list consumes over 6 GB of memory, while all other data structures combined use less than 100 MB, or 1/60th the space. This is hardly unexpected — the number of unique n-grams in a corpus grows slowly after the corpus reaches a certain size because the number of unique n-grams in English (or any other language) grows rapidly for the first few megabytes of text but considerably more slowly for additional text. Moreover, some combinations (such as "zzyqv") are unlikely to occur in any documents in a corpus (though this sentence shows that *any* n-gram is possible in any document in a given language...). In essence, the first few documents define the "vocabulary," but later documents add few new n-grams to it. However, the number of postings grows linearly in the number of documents, and consumes far more space than document information or file information, both of which also grow linearly.

Because the postings list was clearly the largest impediment to scaling TELLTALE to handle gigabytes, we spent most of our effort optimizing its usage. The following sections describe our efforts.

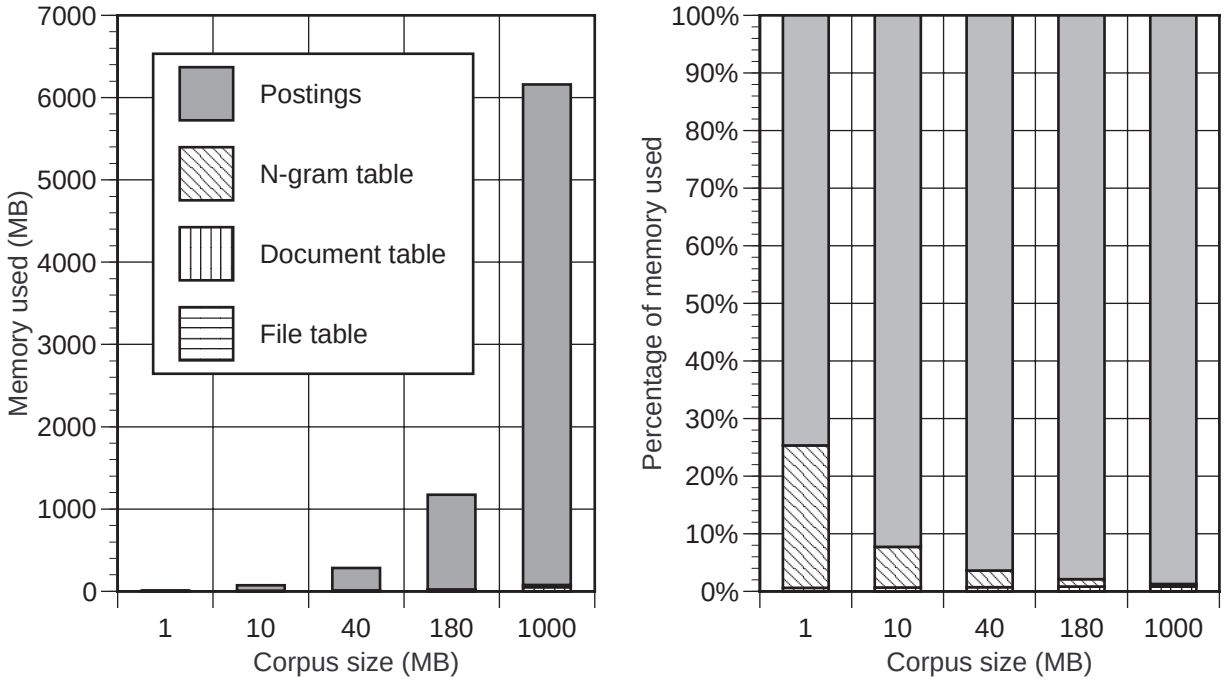

Figure 4. Space consumed by different data structures.

3.2.2 On-disk data structures

The on-disk version of TELLTALE was developed to cope with the limitations of memory space by moving the postings list from memory to disk. Only the main data structures — n-gram hash table, document table, file table — are kept in memory. To accomplish this, TELLTALE generates an on-disk file to record all the information, including a posting list from the files scanned in.

We made several modifications to the original in-memory data structures to handle the on-disk data structures. First, we added variables to each n-gram's entry in the n-gram table to indicate where a postings list is stored in the on-disk file and how long it is. This is only a minor change, yet it increases the memory requirements by over 6 MB for a 1 GB corpus. However, it also allows TELLTALE to rapidly access postings lists on disk with the penalty of just a single disk seek.

The other change we made was to convert pointers in the in-memory data structures into integers, adding a table to translate from the integers into actual entries for files or documents. This change is necessary to allow the data structures to be stored to disk and retrieved, possibly with different memory addresses. We considered using the document ID, which was generated by hashing the document text, for this purpose. However, this ID requires 16 bytes of memory, which is too long to include in every on-disk posting. Instead, we used the serial number assigned to each document. While this number is not unique across corpora, it is unique within a single on-disk file. While its use makes merging corpora together somewhat more complex, it saves a great deal of space; thus, we chose this option. A similar tactic was used to identify entries for particular files. The resulting structures are identical to those shown earlier in Figure 3, except that integers rather than pointers are used to link the tables together and the postings lists are stored on disk.

We chose to allow TELLTALE to operate in one of three modes: `memory`, `update` or `ondisk`. The `memory` mode of this version is basically identical to the original version of TELLTALE, except that it can load a corpus into memory and subsequently conduct similarity searches on it. `Update` mode allows the system to create or update on-disk files. In this mode, TELLTALE can record information about documents that have been scanned in. However, it does not allow queries in `update` mode.

The `ondisk` mode only allows the user to use the files generated in advance under `update` mode. Under this mode the system loads the document hashtable, file table and n-gram hashtable header into memory. However, the posting list is kept on disk only, so each bucket must be loaded into memory before it is used. The format of the on-disk file is shown in Figure 5.

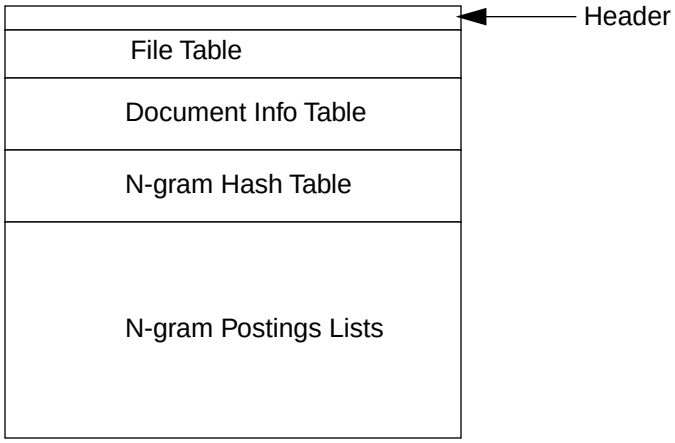

Figure 5. Format for an on-disk corpus index file.

Because of memory limitations, we can only build an index for 10 MB of raw text data in memory. We then dump this corpus index file to disk and clear the memory. Now, the system can scan in another 10 MB of text data, and generate another on-disk file. Thus, we can use a Tcl script to generate a set of on-disk files. TELLTALE implements a function called `mergecorpus` to merge this set of on-disk files into one big on-disk file. Using this method, we can generate a 1 GB on-disk file containing all the information for about 300 MB of raw text data. After the big on-disk file is generated, the `usecorpus` command can load the file table, document table, and n-gram hashtable header into memory. We can then issue queries against this large corpus without having the memory space to read it into memory in its entirety.

The file header records the general information for this file. It includes the number of corpus files, documents and n-grams in this text data set. It also contains the start offset for the other parts of the corpus index — file information, document information, n-gram index and n-gram postings lists. Additionally, several per-corpus summary values are stored here. The remaining sections of the on-disk file contain copies of the data stored in memory. However, the on-disk version must use integers rather than pointers for internal references such as a “pointer” from a document’s information to the structure describing the file that contains it. This conversion requires temporary tables on reading and writing the data, but does not require permanent data structures.

The n-gram hash table is split into two pieces in the on-disk file, one that contains the "header" for each n-gram bucket and the other listing all of the postings for the n-gram. This split is done so that the headers can be easily loaded in by a sequential read while the postings buckets are read off disk on demand. At 32 bytes per header, even a 1 GB text corpus indexed by 5-grams requires only 32 MB for the n-gram header hash table. This contrasts with the several gigabytes required for the postings list.

3.2.3 Performance

Using this naive strategy without compression, the on-disk file consumes 4.5-7 times the size of the raw text data. When the size of raw text data is small, this rate is even larger because the table of unique n-grams dominates the size of the on-disk index. As the corpus grows, however, the number of unique n-grams grows much more slowly, allowing the postings lists to dominate the space used for medium and large corpora.

We generated a 177 MB on-disk file representing the information from 40 MB of text data from the *Wall Street Journal*. Although the index requires a good amount of disk space, it increases the capability of TELLTALE from 10 MB to 40 MB of text data with the same memory in the workstation. The performance of similarity is also acceptable, though somewhat slow. However, we had to reduce the size of the on-disk index if we were to be able to perform retrieval on gigabyte corpora. The following section describes our compression techniques.

3.3 Compression

A large on-disk file not only takes up disk space, but also slows down the speed of calculating similarity due to the time it takes to read posting lists from disk. Since I/O is very slow compared to CPU instructions, compressing the posting list results in two benefits: lower disk storage utilization and faster similarity computations due to reductions in the time needed to read a bucket.

3.3.1 Strategy

The original on-disk file contained postings lists composed a pair of numbers for each document in which the n-gram occurred: an integer identifying the document containing the n-gram and the normalized frequency for the n-gram in the document. This strategy required 8 bytes for each posting, 4 bytes each for the document number and a floating point number for the frequency.

First, we converted all of the n-gram frequencies into integers by storing the actual count of n-grams in a document rather than the normalized frequency. Since we already store the number of n-grams in a document, it is a simple calculation to regenerate normalized frequency from the n-gram count and size of the document. This shift allows us to use standard integer compression algorithms on our postings lists.

Second, we noted that, if we sorted the postings in a list by document number, we could store the difference between an individual posting's document number and that of the previous posting. These gaps are smaller in magnitude than the document numbers themselves, and have the additional desirable property that they are smallest for large postings lists with many entries.

These two strategies can be used to greatly reduce the size of an on-disk index for n-gram based information retrieval. Moreover, compression can be more effective for n-grams than for words because the distribution of term frequencies is more skewed for n-grams than for words. For

example, Figure 6 shows the distribution of term frequencies for 5-grams in the combined 1 GB corpus alongside the distribution of integers describing the "gap" between document numbers in postings.

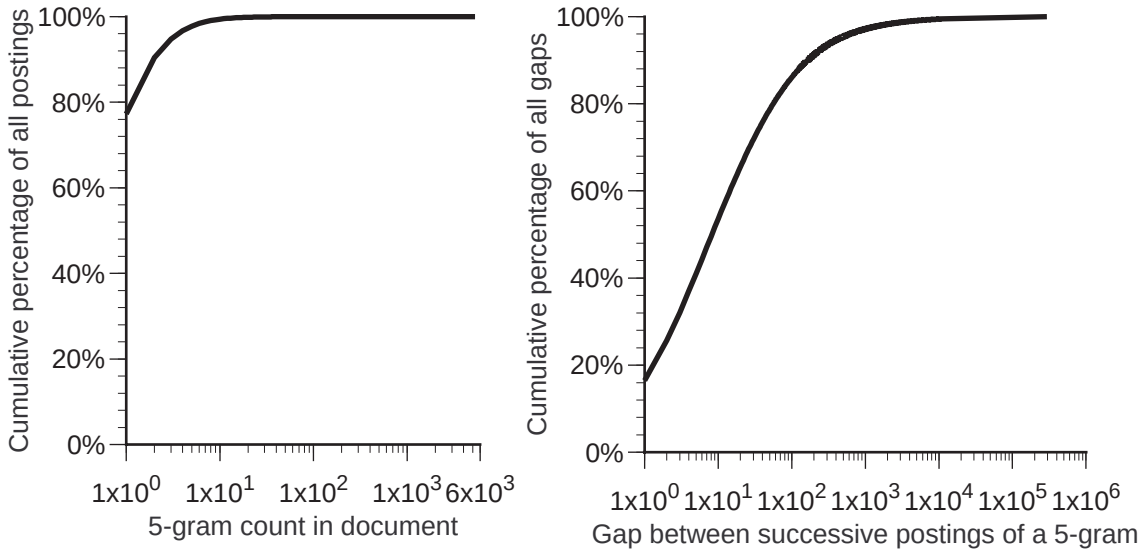

Figure 6. Distribution of 5-gram frequencies and document number gaps. There are a total of 4.62×10^8 postings in this corpus, which contains close to 1 GB of text.

The data in Figure 6 show that posting counts have the greatest potential for compression, though there is also some opportunity for compression of document number gaps as well. The default representation of integers in TELLTALE is 4 bytes long, allowing values up to 2^{32}-1. However, the count of a particular n-gram in a single document is usually a very small number; few n-grams have high frequency within a document. In the corpus we studied, 97.77% of all postings had counts of 5 or less, while over 77% had a count of exactly 1. Based on this finding, we knew that the vast majority of counts could be stored in far less than the standard 4 bytes of an integer.

We also looked at the distribution of document serial number gaps, also shown in Figure 6. This graph shows that most serial number gaps are also relatively small. However, the curve falls off far more slowly than that for posting counts. Half of the gaps were 8 or smaller, and 92.6% were 255 or less. This distribution means that over 92% of all gap values could each be stored in a single byte rather than the 4 bytes required by the default representation.

Based on these findings, we implemented two different compression schemes in TELLTALE. The first was simple to implement and provided reasonable compression for both n-gram counts and document serial number gaps. However, it was considerably below optimal; the `gzip` utility was able to compress the indices by a factor of two. We then switched to gamma compression, yielding files that were approximately the same size as the result of using `gzip` on the original files that used the first compression scheme.

3.3.2 Simple compression algorithm

Based on the statistics discussed in Section 3.3.1, we first considered a simple compression algorithm that saves space for small numbers. We used a single byte to represent numbers from 0 to 2^7-1, two bytes for numbers 2^7 to 2^{14}-1, and four bytes for numbers in the range 2^{14} to 2^{30}-1. These numbers were stored in the format shown in Figure 7.

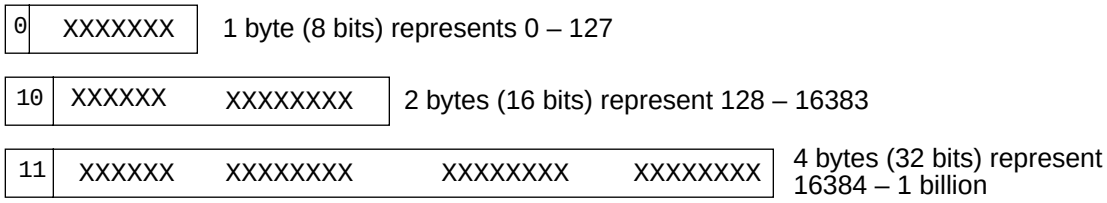

Figure 7. Number formats for the simple compression scheme.

We got good compression results from this scheme. We generated a large on-disk file containing all of the documents in the ZIFF1, ZIFF2, AP1, and AP2 corpora using this compression. The combined corpus has 960 MB of raw text, including 294,440 documents and 889,125 unique n-grams, resulting in an on-disk file requiring 1.085 GB of storage. This provides better than a factor of four compression relative to the uncompressed on-disk file. Additionally, query performance improves greatly. After the system loads the in-memory tables for the large corpus, it can compute, sort, and display the similarity at the rate of 5.5 minutes per 1000 characters in the query.

This simple compression algorithm showed that we can process n-gram queries against 1 GB of text data quite well. However, we did not achieve as much compression as we could. We noticed that gzip was able to compress our on-disk files by a factor of 2, suggesting that we could devise a compression scheme that approached, or even surpassed, this level of compression. Doing so would reduce the size of on-disk indices and improve performance by reducing the amount of data that must be read for each query. Balanced against this is the increased CPU time necessary to encode and decode a more complex compression scheme.

3.3.3 Gamma compression

Our initial experiments showed that compression was very effective at reducing resource requirements, but that we could achieve additional gains with more efficient compression schemes. We primarily considered several standard schemes for our corpus: unary code, gamma compression, and delta compression. All of these compression methods are relatively straightforward to implement and could provide significant improvement over our initial scheme.

The first such code we considered was the unary code [7]. In this code, an integer $x \geq 1$ is coded as $x - 1$ one bits followed by a zero bit. For example, the unary code for 4 is 1110. This represents small integers such as n-gram counts within a document well, but is very inefficient at representing larger integers such as document serial number gaps.

We next examined gamma compression, which represents an integer x as two parts: a unary code for an integer m followed by a k-bit binary value y. The value for k is determined by taking the mth element of a vector of integers that is constant across all compressed values (i.e., constant for a

particular compression scheme). For a vector $\langle k_0, k_1, \ldots, k_n \rangle$, the value of a representation *my* can be calculated as $\left(\sum_{i=0}^{m-1} 2^{k_i} \right) + y + 1$. Table 2 shows examples of representations using two different vectors for values of *k*. Note that the largest number representable in each scheme is limited by the largest integer in the *k*-vector. For this reason, gamma compression implementations often contain a large terminal value to handle the occasional integers larger than the range in which most values fall.

Prefix	$\langle 0, 1, 2, 3, 20 \rangle$		$\langle 1, 3, 5, 7, 15 \rangle$	
	k	**Maximum value**	**k**	**Maximum value**
0 (0)	0	$2^0 = 1$	1	$2^1 = 2$
1 (10)	1	$1 + 2^1 = 3$	3	$2 + 2^3 = 10$
2 (110)	2	$3 + 2^2 = 7$	5	$10 + 2^5 = 42$
3 (1110)	3	$7 + 2^3 = 15$	7	$42 + 2^7 = 160$
4 (11110)	20	$15 + 2^{20}$	15	$160 + 2^{15}$

Table 2. Sample gamma compression representations.

If this table is constructed in memory, both compression and decompression are relatively simple. For compression, the algorithm first subtracts 1 from the value being encoded. It then need only search through the table and find the smallest entry greater than the number being compressed. The unary prefix may be read directly from the table, and the binary portion is obtained by subtracting the previous maximum value entry from the value to be compressed. This value is expressed in the number of bits found in the table entry. For example, compressing the value 18 using the scheme on the right would be done by first subtracting 1, yielding the value 17. Looking in the table, this requires a unary prefix of 2 (110) and a binary portion of 17-10 = 7 expressed in 5 bits (00111). Thus, the final representation is 11000111.

The decoding process is also relatively simple. The unary prefix is extracted and used as an index into the table. The maximum value from the previous entry is then added to the *k*-bit value that follows the unary prefix using the *k* found in the table, and the result is added to 1. For example, decoding the compressed value 10110 using the scheme on the right is done by looking up 2 (10) in the table, and finding that k=3 and the "base value" is 2. The uncompressed number is thus 2 + 1 + 6 (110) = 9.

The delta code is a derivative of the gamma code that uses the gamma code, rather than unary, to encode the prefix. However, the delta code is more complex, and not as efficient for very small numbers such as those found in n-gram frequency counts. There are other, more advanced compression techniques, but these two are the most commonly used algorithms. Since integers in this system are not big and the gamma code is easy to implement, we picked gamma coding to compress the posting list.

Even after selecting gamma compression, however, we had to choose the best vector to use to compress our integers. To do this, we ran several experiments against the data shown in Figure 6 to compute the amount of space that would be required using several different vectors. The results of some of our experiments are shown in Table 3. As this table shows, optimal compression for n-gram counts and document gaps were achieved with different vectors. To simplify implementation, we chose the first vector in Table 3 as our compression scheme, though future versions of TELLTALE may use different vectors to compress different value sets. Even with our choice of a single vector, however, we were within 5.5% of the space required by the optimal two-vector compression scheme.

Vector	N-gram counts (MB)	Document gap (MB)	Total (MB)
<0,1,2,3,4,5,6,7,8,9,10,11,12,14,16,18,20,28>	87.7	409.1	496.8
<0,1,2,3,4,6,8,10,12,14,16,18,28>	87.7	409.2	496.9
<0,0,1,2,3,4,5,6,7,8,10,12,14,16,18,28>	83.2	435.0	518.2
<0,0,0,1,2,3,4,5,6,7,8,10,12,14,16,18,28>	82.3	460.2	542.5
<0,0,0,0,1,2,3,4,5,6,7,8,10,12,14,16,18,28>	**82.1**	484.7	566.8
<0,0,2,4,6,8,10,12,14,16,18,28>	86.4	412.0	498.4
<0,2,4,6,8,10,12,14,16,18,28>	96.6	**388.1**	**484.7**
<0,3,6,9,12,15,18,21,28>	106.9	395.2	502.1
<0,2,3,6,9,12,15,18,21,28>	96.0	398.5	494.5
<0,1,2,4,6,9,12,15,18,21,28>	88.2	408.2	496.4

Table 3. Index sizes for various gamma compression vectors.

Using our gamma compression scheme with the vector in the first line of Table 3, we generated a large on-disk index file covering ZIFF1, ZIFF2, AP1, and AP2. The raw text from these files was 960 MB, but the on-disk index consumed only 647 MB. While the table entry suggests that under 500 MB would be necessary, the table does not include additional data structures necessary to store document info and the n-gram headers; these structures make up the additional 150 MB. Since they are only read in at startup, however, we decided not to attempt to compress them as well. Our experiment gave a compression ration of 0.67, which is nearly as good as `gzip`.

Gamma compression also improves query performance by reducing the amount of data that must be read for a single query. After loading the in-memory information for the 960 MB of text, TELLTALE can compute and sort about 300,000 documents' similarity result at the rate of 50 seconds per thousand characters in the query. This is considerably faster than our original compression scheme, as can be seen in Figure 8.

3.4 Handling gamma compressed postings lists in memory

The dropping price of memory has made it possible to purchase large amounts of memory at a reasonable price. With the help of gamma compression, the compressed postings list is small enough to be loaded into main memory if allocation is handled intelligently (i.e., not by malloc...).

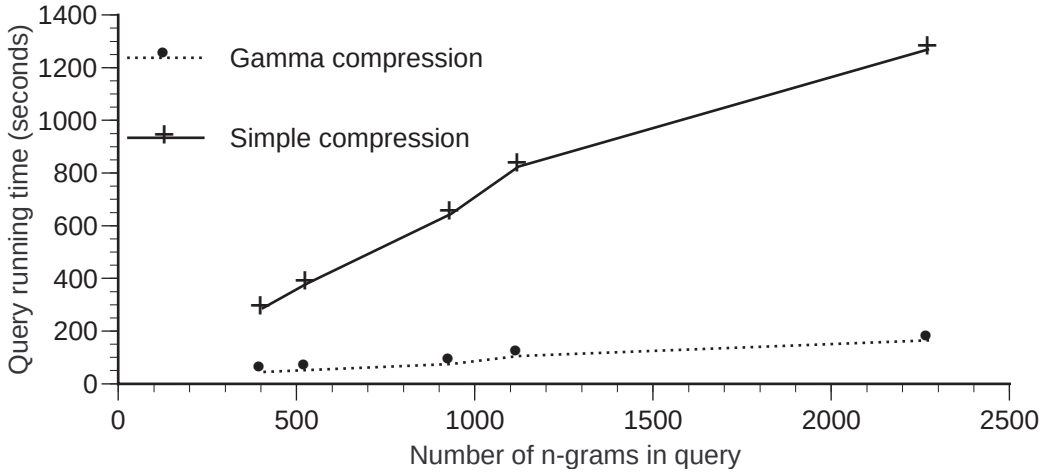

Figure 8. Retrieval time for different query sizes and compression methods.

Since about 1 GB of raw text data can be indexed in under 700 MB, we can handle 1 GB of text data in less than 750 MB of main memory, allowing for a small amount of overhead. Doing so will improve the performance a great deal by eliminating disk I/O during a query.

The major difficulty with handling the compressed postings lists in memory is coping with the many bucket capacities necessary — some postings lists will be only a few bytes long, while others may require many thousands of bytes. Additionally, these buckets must grow dynamically as new documents are scanned in. These requirements are best met using lists built from fixed size "chunks" of space connected in a linked list. The overhead for this scheme is relatively small — fixed size chunks that can hold 32 bytes require only 4 bytes of pointer overhead for an overhead of 12.5%. In addition, fixed size chunks waste some space because part of the last chunk is unused. On average, this will waste half of a chunk per n-gram, 16 bytes in our system. Thus, total overhead for a system that scanned in the 1 GB AP-ZIFF corpus would be 14 MB for unused chunk space and about 62.5 MB for pointers. In future versions of TELLTALE, we will attempt to reduce this overhead by allowing 2-3 different chunk sizes for maximum efficiency.

Operation using in-memory compressed postings lists is similar to that using uncompressed postings lists, but with the additional step that postings lists must be compressed before they are permanently stored. Additionally, postings lists must be uncompressed before they are used in similarity calculations. While this technique uses less memory than uncompressed postings lists, it is somewhat slower because of the time needed to uncompress a postings list. A 200 MHz Pentium laptop is capable of decompressing two million integers a second; while this seems an impressive number, most similarity calculations must process ten million postings or more. Thus, decompression time contributes significantly to similarity calculation time.

By using in-memory gamma compression rather than uncompressed postings lists, TELLTALE can reduce its memory usage by a factor of four or more, as Figure 9 shows. Both versions of TELLTALE keep the entire postings list as well as the other data structures, including the document information table and n-gram information table, in memory. The only difference between the two is that the gamma compressed version uses a great deal less memory for larger amounts of text. Note, however, that the original TELLTALE uses slightly less memory for small corpora; this

occurs because the overhead for the gamma compression version of TELLTALE is slightly higher. However, this higher overhead is more than recovered as the amount of indexed text increases.

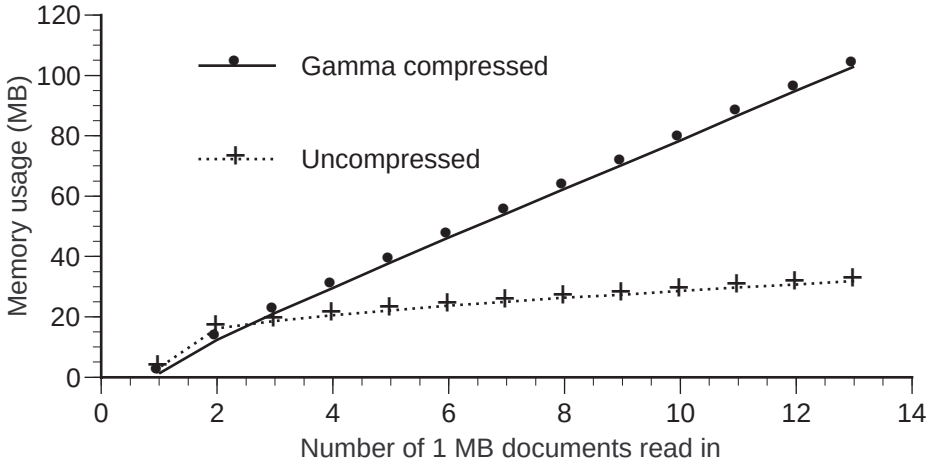

Figure 9. Memory usage for original and in-memory gamma compressed TELLTALE.

The in-memory gamma compressed version also provides increased speed relative to the on-disk version. Comparisons with the original, uncompressed in-memory version are less relevant because the original version can only handle very small corpora. Thus, we focused our attention on the relative performance of the gamma compressed postings lists on disk and in memory. We ran queries against a collection containing the 257 MB of text in ZIFF1, which comprise 75,029 documents, 562,492 unique n-grams, and a total of 185,159,683 postings. We were limited to this size because the machine on which the queries were run, a two processor SGI Origin 200, had only 256 MB of memory. When the entire ZIFF1 corpus was loaded into memory, it used 210 MB of memory, leaving the rest for operating system use. As can be seen in Figure 10, in-memory gamma compression is twice as fast as on-disk gamma compression, though the difference is not as large as we had expected.

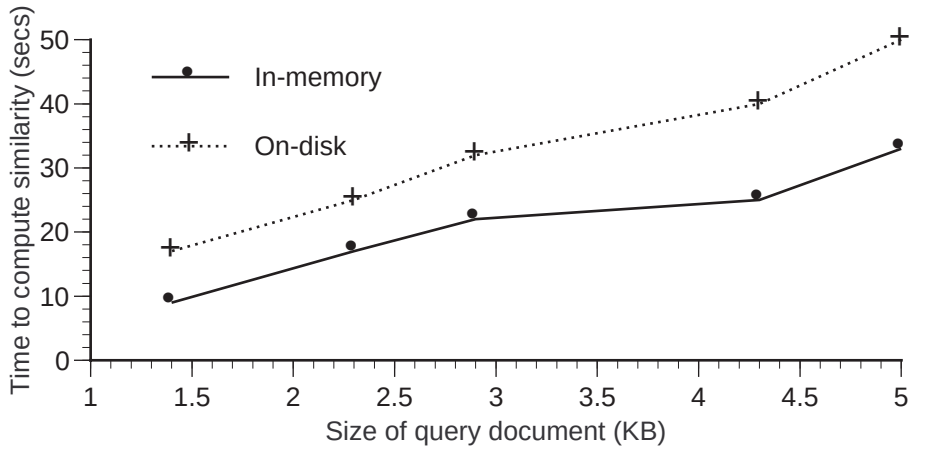

Figure 10. Performance comparison of in-memory and on-disk version with gamma compression.

As the preceding experiments have shown, gamma compression performs well for n-gram-based IR just as it does for word-based schemes. However, we had to adjust the gamma compression vectors to best compress the postings lists generated for n-grams because document gaps and, particularly, occurrence count distributions are differ between words and n-grams. Using these techniques, we expanded TELLTALE's capability from around 10 MB to over 1 GB while maintaining good query performance.

4 Exploring different similarity mechanisms

Because this is the first n-gram-based information retrieval system capable of handling a gigabyte of text, we were able to perform several experiments on using different document similarity schemes that were previously done only on relatively small corpora [19]. While our experiments were not extensive, they showed the effects of eliminating common n-grams from queries and the resulting performance gains. We also conducted some basic experiments on the effectiveness of TF/IDF similarity using n-grams rather than words.

4.1 Incremental similarity calculations

Incremental similarity is based on the idea that a n-gram which occurs in most documents is not important for retrieving a similar documents. If every document contains this n-gram, it must be a common term and not a key term to distinguish those documents. At the same time, these n-grams have a long posting list, requiring TELLTALE to spend a relatively long time reading and uncompressing the posting list for them. To test the effectiveness of ignoring common n-grams, we modified TELLTALE to include a threshold t $(0 \le t \le 1)$ above which an n-gram is ignored for similarity computations. TELLTALE then ignores all n-grams that occur in more than $t \times numDocs$ documents, where *numDocs* is the total number of documents in the corpus.

Because the n-grams that occur in a high percentage of all documents are not likely to be important, they should not affect the accuracy of a query. A TELLTALE user can set the threshold with a larger value resulting in better accuracy at the expense of longer similarity computation time. On the other hand, reducing the threshold speeds up similarity computation but reduces accuracy. The speedup from reducing the threshold is illustrated in Figure 11. As expected, lower thresholds require less computation time, but the improvement is not dramatic. Even for a threshold of 50%, the maximum improvement time is from 33 seconds to 28 seconds, or 85% of the original time. Because there are many more unique n-grams than words, there are fewer n-grams that occur in most of the documents. Thus, omitting the most common ones does not result in very large performance gains.

We next produced a simple precision-recall graph for reduced n-gram frequency thresholds. Since we did not have "official" relevance judgments for queries on our corpus, however, we used approximate judgments. Nonetheless, the graph in Figure 12 shows that, as expected, lower thresholds result in lower precision and recall. Given the relatively small improvement in performance, we believe that eliminating common terms from similarity computations may not be as effective for n-grams as it is for words.

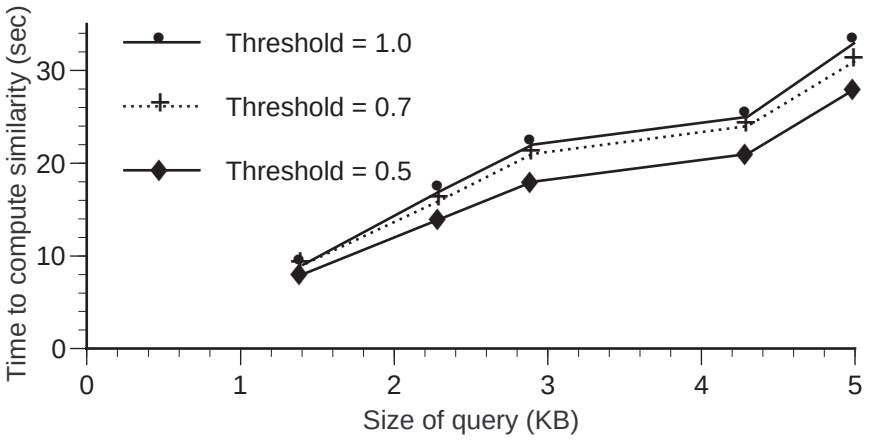

Figure 11. Performance for incremental similarity with different thresholds.

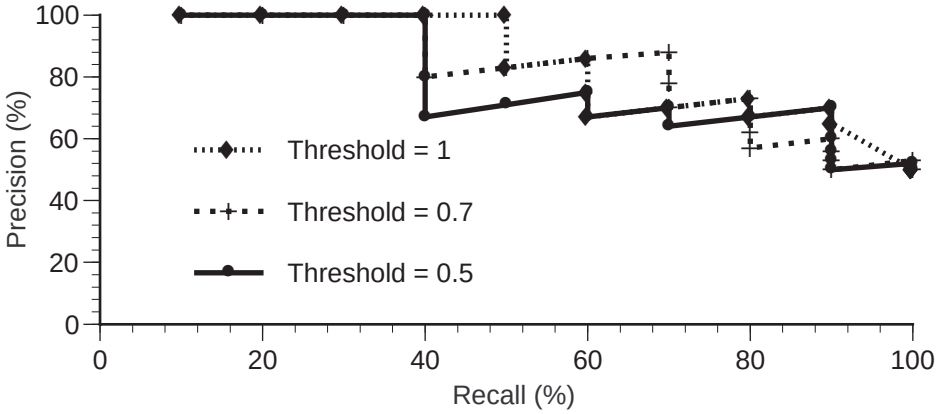

Figure 12. Recall-precision curve for different thresholds.

4.2 TF/IDF similarity

Another standard weighting system used in word-based systems is TF/IDF (Term Frequency/ Inverse Document Frequency) [19]. It uses the term frequency within a document times the log of the total number of documents over the number of documents containing the term. TELLTALE includes support for TF/IDF weighting, but we found it very difficult to test precision and recall for a different weighting system without a corpus with scored queries. However, our experiments showed that TF/IDF similarity can be calculated at about the same speed as the original TELLTALE similarity. Thus, if future work shows that TF/IDF yields more accurate similarity measures when using n-grams, as suggested by [19], TELLTALE will be able to support it with little loss in performance

5 Conclusions and future work

Though we greatly expanded TELLTALE's capacity and improved its performance, we did not affect its ability to handle multilingual or slightly garbled documents. It is these advantages, combined with an ability to perform retrieval using full documents rather than relatively short queries,

that make TELLTALE a useful tool. However, there is still much work to be done with it. We hope to perform a more complete study of the tradeoffs between different similarity measures using n-grams rather than words. Because TELLTALE is the first system using n-grams that can handle a gigabyte of text, we hope to be able to show that n-grams are equal to or better than words as indexing terms. We are also currently performing experiments in using TELLTALE to index collections in non-English languages ranging from European languages such as French and Spanish to ideogram-based languages such as Chinese. Our preliminary results are promising, but more investigation needs to be done.

We have demonstrated that it is possible to build a text information retrieval engine using n-grams rather than words as terms that can handle gigabyte-sized corpora. The TELLTALE IR engine adapts techniques that have been used for word-based systems to n-gram-based information retrieval, making adjustments as necessary to account for the different term distributions exhibited by n-grams. Because there are many more unique n-grams than words in a document, TELLTALE must cope with 1-2 orders of magnitude more unique terms and at least an order of magnitude more postings to allow indexing of a text corpus. By modifying standard techniques, we demonstrated a system that provides good performance on the large corpora that computers will be called upon to index. These techniques can also be used on other systems where performance and scalability are critical to better use of system resources and larger scale and faster processing.

Acknowledgments

The authors are grateful to the many people who contributed to this work by giving us feedback and suggestions. These include Claudia Pearce and Bill Rye at the Department of Defense and David Ebert at UMBC.

References

[1] Steve Lawrence and C. Lee Giles, “Searching the World Wide Web,” *Science* **280**(3), 3 April 1998, pages 98 - 100.

[2] Claudia Pearce and Ethan Miller, “The TELLTALE Dynamic Hypertext Environment: Approaches to Scalability,” in *Advances in Intelligent Hypertext*, J. Mayfield and C. Nicholas, eds. Lecture Notes in Computer Science, Springer-Verlag, October 1997, pages 109 - 130.

[3] Robert R. Korfhage, *Information Storage and Retrieval*, John Wiley & Sons, 1997.

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

[5] James P. Callan, W.Bruce Croft, and John Broglio, “TREC and Tipster experiments with INQERY,” *Information Processing and Management* **31**(3), 1995, pages 327 - 343.

[6] D. Heckerman, “A tutorial on learning with Bayesian networks,” Technical Report MSR-TR-95-06, Microsoft Research, March 1995 (revised November, 1996).

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

[8] G. Salton and M.J. McGill, "The SMART and SIRE Experimental Retrieval Systems," in *Readings in Information Retrieval*, Karen Sparck Jones and Peter Willett, eds., Morgan Kaufmann, 1997, pages 381 - 399.

[9] Claudia Pearce and Charles Nicholas, "TELLTALE: Experiments in a Dynamic Hypertext Environment for Degraded and Multilingual Data," *Journal of the American Society for Information Science*, April 1996, pages 263 - 275.

[10] Marc Damashek, "Gauging Similarity with n-grams: Language-Independent Categorization of Text," *Science* **267**, 10 February 1995, pages 843 - 848.

[11] C. E. Shannon, "Prediction and entropy of printed English," *Bell System Technical Journal* **30**, pages 50 - 64.

[12] F. Cuna Ekmekcioglu, Michael F. Lynch, and Peter Willett, "Stemming and N-gram Matching For Term Conflation In Turkish Texts," available at http:// www.shef.ac.uk/uni/academic/I-M/is/lecturer/paper13.html#lovi68.

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

[14] Brent B. Welch, *Practical Programming in Tcl and Tk*, 2nd edition, Prentice Hall, 1997.

[15] Donna Harman, "The DARPA TIPSTER project," *ACM SIGIR Forum* **26**(2), Fall 1992, pages 26 - 28.

[16] The Text Retrieval Conference. Information available at `http://trec.nist.gov`.

[17] R. L. Rivest, "The MD5 Message-Digest Algorithm," RFC 1321, Network Working Group, April 1992.

[18] *Secure Hash Standard*, FIPS-180-1, National Institute of Standards and Technologies, U.S. Department of Commerce, April 1995.

[19] James Mayfield and Paul McName. "N-gram vs. Words as Indexing Terms," available at URL http://www.cs.umbc.edu/~mayfield/.