Norwegian University of Science and Technology
Technical Report IDI-TR-6/2004

The TDocGen temporal document generator

Kjetil Nørvåg* and Albert Overskeid Nybø
Department of Computer and Information Science
Norwegian University of Science and Technology
7491 Trondheim, Norway

ISSN: 1503-416X

*Email of contact author: Kjetil.Norvag@idi.ntnu.no

Abstract

In research in temporal document databases, large temporal document collections are necessary in order to be able to compare and evaluate new strategies and algorithms. Large temporal document collections are not easily available, and an alternative is to create synthetic document collections. In this paper we will describe how to generate synthetic temporal document collections, how this is realized in the *TDocGen* temporal document generator, and we will also present a study of the quality of the document collections created by TDocGen.

Keywords: Temporal databases, document databases, data generators, benchmarking

1 Introduction

In this paper we will describe how to make document collections to be used in development and benchmarking of temporal document databases (for example a company document repository or a web archive), and how this is realized in the *TDocGen* temporal document generator. In order to motivate this work, we will first give an overview of temporal document databases and why a temporal document generator is needed.

1.1 Motivation

In a temporal document database system, it is possible to keep previous versions of documents after they are updated or deleted. All document versions are timestamped, and time together with document name is used to uniquely identify a particular version. In traditional document databases it is usually possible to store and retrieve particular documents, as well as perform text-based queries to find all documents containing a particular word (or a particular set of words). In a temporal document database a particular document version can be also retrieved, and should also be possible to search for all documents containing a particular word at a particular time$t$ (temporal text-containment query).

Aspects of temporal document databases are now desired in a number of application areas, for example web databases and more general document repositories:

- The amount of information made available on the web is increasing very fast, and an increasing amount of this information is made available *only* on the web. While this makes the information readily available to the community, it also results in a low persistence of the information, compared to when it is stored in traditional paper-based media. This is clearly a serious problem, and during the last years many projects have been initiated with the purpose of archiving this information for the future. This essentially means crawling the web and storing snapshots of the pages, or making it possible for users to "deposit" their pages. In contrasts to most search engines that only store the most recent version of the retrieved pages, in these archiving projects all (or at least many) versions are kept, so that it should also be possible to retrieve the contents of certain pages as they were at a certain time in the past. The most famous project is this category is probably the Internet Archive Wayback Machine [7], but in many countries similar projects also at the national level, typically initiated by national libraries or similar organizations.

- An increasing amount of documents in companies and other organizations is now only available electronically. These documents can be in a number of formats like plain text, HTML, XML, Microsoft Word, Adobe PDF, etc. Many organizations already has searchable repositories or intranet search engines that can be used to retrieve documents based on keywords search, and possibly also other searchable parameters like creation time.

Support for temporal document management is not yet widespread. Important reasons for that are issues related to:

- Space usage of document version storage.
- Performance of storage and retrieval.
- Efficiency of temporal text indexing.

More research is needed in order to resolve these issues, and for this purpose test data is needed in order to make it easier to compare existing techniques and study possible improvements of new techniques. In the case of document databases test data means document collections. In our previous work [10], we have employed versions of web pages to build a temporal document collection. However, by using only one collection we only study the performance of one document creation/update pattern. In order to have more confidence in results, as well as study characteristics of techniques under different conditions, we need test collections with different characteristics.

Acquiring large document collections with different characteristics is a problem in itself, and acquiring *temporal* document collections close to impossible. In order to provide us with a variety of temporal document collections, we have developed the TDocGen temporal document generator. TDocGen creates a temporal document collection whose characteristics are decided by a number of parameters. For example, probability of update, average number of new documents in each generation, etc., can be configured. A synthetic data generator is in general useful even when test data from real world applications exists, because the it is o very useful to be able to control the characteristics of the test data in order to do measurements with data sets with different statistical properties.

Creating synthetic data is not a trivial task, even for "simple" data like relational data. A general document generator is even more difficult, because not only the contents (words) is important, but also the structure and order is important. However, in our context the task can be simplified: our context is not the information retrieval part of the problem, only storage related issues as described above, and in our own research, indexes to support word-containment queries in particular. As a result, we have not taken into account structure and order of words in the documents. However, because one of our application areas of the created document collections is study of text-indexing techniques, the occurrence of words, size of words, etc., have to be according to what is expected in the real world. This is a non-trivial issue that will be explained in more detail later in the paper. In order to make temporal collections, the TDocGen document generator essentially simulates the document operation by users during a specific period, i.e., creations, updates, and deletes of documents. The generator can also be used to create non-temporal document collections when collections with particular characteristics are needed.

1.2 Organization

The organization of the rest of this paper is as follows. In Section 2 we give an overview of related work. In Section 3 we define the data and time models we base our work on. In Section 4 we give requirements for a good temporal document generator. In Section 5 we describe how to create a temporal document collection. In Section 6 we describe TDocGen in practice. In Section 7 we evaluate how TDocGen fulfill the requirements. Finally, in Section 8, we conclude the paper.

2 Related work

For measuring various aspects of performance in text-related contexts, a number of document collections exist. The most well-know example is probably the TREC collections [15], which includes text from newspapers as well as web pages. Other examples are the INEX collection [8] which contains 12,000 articles from IEEE transaction and magazines in XML format, and documents in Project Gutenberg [14], which is a collection of approximately 10,000 books.

A number of other collections are also publicly available, some of them can be retrieved from the UCI Knowledge Discovery in Databases Archive [16] and the Glasgow IR Resources pages [5]. We

are not aware any temporal document collections suitable for our purpose.

Several synthetics document generators have been developed in order to provide data to be used by XML benchmarks. Examples are ToXgene [1], which creates XML documents based on a template specification language, and the data generators used for the Michigan benchmark [11] and XMark [12]. Another example of generator is the change simulator used to study the performance of the XML Diff algorithm proposed in [3], which takes an XML document as input, do random modification on the document, and outputs a new version. Since the purpose of that generator was to test the XML Diff algorithm it does no take into account word distribution and related aspects, thus making it less suitable for our purpose.

In the context of web warehouses, studies of evolution of web pages like those presented in [2, 4] can give us some hints on useful parameters to use for creating collections reflecting that area.

3 Document and time models

In our work we use the same data and time model as is used in the V2 document database system [10].

A document version V is in our context seen as a list of words, i.e., $V = [w_0, w_1, ..., w_k]$. A word w_i is an element in the vocabulary set W, i.e., $w_i \in W$. There can be more than one occurrence of a particular word in a document version, i.e., it is possible that $w_i = w_j$. The total number of words n_w in the collection is $n_w = \sum_{i=0}^{n} |V_i|$.

In our data model we distinguish between documents and document versions. A temporal document collection is a set of document versions $V_0...V_n$, where each document version V_i is one particular version of a document D_i. Each document version was created at a particular time T (more than one version of different documents can have been created at T), and we denote the time of creation of document version V_i as T_i. A particular document version is identified by the combination of document name N_j and T_i. Simply using document name without time denotes the most recent document version.

A document version exists (is valid) from the time it is created (either by creation of a new document or update of the previous version of the document) and until it is updated (a new version of the document is created) or the document is deleted. The collection of all document versions is denoted C, and the collection of all document versions valid at time T (a snapshot collection) is denoted C_T. A temporal document collection is a collection containing document versions not valid at the same time.

The time model is a linear (non-branching) time model, time advances from the past to the future in an ordered step-by-step fashion. The time axis can be viewed as an ordered sequence of instants, where at some instants an event happens. In the context of this paper, an event is a document creation, deletion, or update. More than one event can happen at a particular time instant.

4 Requirements for a temporal document generator

A good temporal document generator should produce documents with characteristics similar to real documents. The generated documents have to satisfy a number of properties:

- Document contents:
  - Number of unique words (size of vocabulary) should be the same as for real documents, both inside a document and at the document collection level.

  - Size and distribution of word size should be the same as for real documents.
  - Average document size as well as distribution of sizes should be similar to real documents.

- Update pattern:
  - A certain number of document in the start, i.e., when the database is first loaded.
  - A certain number of documents created and deleted at each time instant.
  - A certain number of documents updated at each time instant.
  - Different documents have different probabilities of being updated, i.e., dynamic versus relatively static documents.
  - The amount of updates to a document, including inserting and deleting words.

Many parameters of documents depend on application areas. The document generator should be used to simulate different application areas, and has to be easily reconfigurable. We will now in detail describe some of the important parameters and characteristics.

4.1 Contents of individual documents and a document collection

Documents containing text will in general satisfy some statistical properties based on empirical laws, for example size of vocabulary will typically follow Heaps' law, distribution of words are according to Zipf's law, and have a particular average length of words.

4.1.1 Size of vocabulary

According to Heaps' law [6], the number of *unique* words $n_u = |W|$ (number of elements in vocabulary) in a document collection is given as a function of the total number of words n_w in the collection: $|W| = Kn_w^\beta$, where K and β are determined empirically. In English texts typical values are $10 < K < 100$ and $0.4 < \beta < 0.6$. Note that Heaps' law is valid for a *snapshot collection*, and not necessarily valid for a complete temporal collection. The reason is that a temporal collection in general will contain many versions of the same documents, contributing to the total amount of words, but not many new words to the vocabulary.

4.1.2 Distribution of words

The distribution of the words in natural languages and typical texts is Zipfian, i.e., the frequency of use of the n^{th}-most-frequently-used word is inversely proportional to n: $P_n = \frac{P_1}{n^a}$, where P_n is the frequency of occurrence of the n^{th} ranked item, a is close to 1, and $P_1 \approx 0.1$ [18, 17].[1]

4.1.3 Word length

Average word length can be different for different languages, and we have also two different measures: 1) average length of words in vocabulary, and 2) average length of words occurring in documents. Because the most frequent words are short words, the latter measure will have a lower value. The average word length for the words in the documents we used from the Project Gutenberg collection was 4.3.

[1] For our test collections we have measured $P_1 = 0.0$.

4.2 Temporal characteristics

The characteristics of a snapshot collection as described above is well studied during the years, and a typical document collection will obey these empirical laws. Temporal characteristics, on the other hand, are likely to be more diverse, and very dependent of application area. For example, a in document database containing newspaper articles the articles themselves are seldom updated after publication. On the other hand, a database storing web pages will be very dynamic.

It will also usually be the case that some documents are very dynamic and frequently updated, while some documents are relatively static and seldom or never updated after they have been created.

Because these characteristics are very application area dependent, a document generator should be able to create documents based on specified parameters, i.e., update ratio, amount of change in each document, etc., as listed earlier in this section.

5 Creating a temporal document collection

In this section we describe how to create a temporal document collection. We describe first the basis of creating non-temporal documents, before we describe how to use this for creating a temporal document collection.

Each snapshot collection C_T, or "generation", should satisfy properties as described in the previous section. The basis for creating the first generation as well as new texts to be inserted into updated documents is the same as if creating a non-temporal collection.

5.1 Creating synthetic non-temporal documents

Several methods exists for creating synthetic documents, we will here describe the methods we considered in our research, which we call the *naive*, *random-text*, *Zipf-distributed/random words*, and the *Zipf-distributed/real words* methods.

Naive: The easiest method is probably to simply create a document from a random number of randomly created words. Although this could be sufficient for benchmarking when only data amount is considered, it would for example not be appropriate for benchmarking text indexing. Two problems are that occurrence distribution of words and vocabulary size is not easily controllable with this method. Although the method can be improved so that these problems are reduced, there is also the problem that because words in real life are not created by random, and frequent words are not necessarily uniformly distributed in the vocabulary, some of them can be close to each other. One example is some frequently occurring words starting with common prefixes, or different forms of the same word (for example "program" and "programs"), especially the case when stemming (where only the root form of a words is stored in the index) is not employed.[2]

Random-text: If a randomly generated sequence of symbols taken from an alphabet S where one of the symbols are blank (*white space*), and the symbols between two blank spaces are considered as a word, the frequency of words can be approximated by a Zipf distribution [9]. The average word size will be determined by the number of symbols in S. Such sequences can be used to create synthetic documents. However, the problem is that if the average length of words should be comparable to natural languages like English, the number of symbols in S have to be low. Another problem is that

[2] This is typically the case for web search engines/web warehouses.

the distribution is only an approximation to Zipf: it is stepwise distribution, all words with same length has same probability of occurrence. Both problems can be fixed by introducing bias among different symbols. By giving a sufficient high probability for blanks the average length of words even with a larger number of symbols (for example, 26 in the case of the English language) can be reduced to average length of English words, and by giving different probabilities for the other symbols a smoother distribution is achieved. It is also possible to introduce cut-off for long words. The advantage with this methods is that an unlimited vocabulary can be created, but the problem with lexicographically closer words as described above remain.

Zipf-distributed/random-words: A method that will create a document collection that follow Heaps' law and has a Zipfian distribution, is to first create n_u random words with an average word length L. The number of n can be determined based on Heaps' law with appropriate parameters. Then, each word is assigned an occurrence probability bases on Zipfian distribution. This can be done as follows: As described in Section 4.1.2, the Zipfian distribution can be approximated to $P_n = \frac{P_1}{n}$. The sum of probabilities should be 1, so that:

$$\sum_{i=1}^{n_u} P_i = 1 \Rightarrow \sum_{i=1}^{n_u} \frac{P_1}{i} = 1 \Rightarrow P_1 \sum_{i=1}^{n_u} \frac{1}{i} = 1 \Rightarrow nP_n \sum_{i=1}^{n_u} \frac{1}{i} = 1 \Rightarrow P_n = \frac{1}{n \sum_{i=1}^{n_u} \frac{1}{i}}$$

In order to select a new word to include in a document, the result r from a random generator producing values $0 \leq r \leq 1$ are used to select the word ranked k that satisfies $\sum_{j=1}^{k-1} P_j \leq r < \sum_{j=1}^{k} P_j$
Using this method will create a collection with nice statistical properties, but still have the problem of not including the aspect of lexicographically close words as described above.

Zipf-distributed/real words: This actually the approach we use in TDocGen, and is an extension of the Zipf-distributed/random-words approach. Here a *real-world vocabulary* is used instead of randomly created words. In order to make any improvement, these words need to have the same properties as in real documents, including occurrence probability and ranking. This is achieved by first making a histogram of word frequency based on real texts and rank words according to this. The result will be documents that include the aspect of lexicographically close words as well as following Heaps' law and having a Zipfian distribution of words.

5.2 Creating temporal documents

The first event in a system containing the document collection, for example a document database, is to load the initial documents. The number of documents can be zero, but it can also be a larger number if an existing collection is stored in the system. The initial collection can be made from individual documents created as described above.

During later events, a random number of documents are deleted and a random number of new documents are inserted. The next step is to simulate operations to the document collection: inserting, deleting, and updating documents.

Inserting documents. New documents to be inserted into the collection are created in the same way as the initial documents.

Deleting documents. Documents to be deleted are selected from the documents existing at a particular time instant.

Parameters	Pattern I		Pattern II	
	Avg. or Fixed	Std. dev.	Avg. or Fixed	Std. dev.
Number of files that exist the first day	1000	-	10	-
Percentage of documents being dynamic	20	-	20	-
Percent of updates applied to dynamic documents	80	-	80	-
Number of new documents created/day	200	5	2	1
Number of deleted documents/day	100	2	1	1
Number of updated documents/day	500	20	5	2
Number of words in each line in document	10	-	10	-
Number of lines in new document	150	10	150	10
Number of new lines resulted from update	25	5	25	5
Number of deleted lines resulted from update	20	5	20	5

Table 1: Document generator parameters for two temporal patterns. Parameters that are not fixed are uniformly distributed with average and standard deviation as given in the table.

Updating documents. The first task is to decide *which* documents to be updated. In general, the probability of updates to files will also in general follow a Zipfian distribution. A commonly used approximation is to classify files into dynamic and static files, where the most updates will be to dynamic files, and the number of dynamic size is smaller than the number of static files. A general rule of thumb in databases is that 20% of the data is dynamic, but 80% of the updates are applied to this data. This can be assumed to be the case in the context of document databases as well, and in TDocGen documents are characterized as being static or dynamic, and which category a document belongs to is decided when it is created. When updates are to be performed, it is first decided whether the update should be to a dynamic or static file, and which document in the category that is actually updated, is chosen at random (i.e., uniform distribution).

After it is decided what documents to update, the task is to perform the actual update. Since we do not care about structure of text in the documents, we simply delete a random number of lines, and insert a random number of new lines. The text in the new lines are created in the same was as the text to be included in new documents.

One of the goals of TDocGen is that it should be able to create temporal document collections that can have characteristics for chosen application areas. This is achieved by having a number of parameters that can be changed in order to generate collections with different properties. Table 1 show the most important parameters. Some of them are given a fixed value, while other parameters are given as average value and standard deviation. The table also contains the values for two parameter sets in our experiments which are reported in Section 7.

6 Implementation and practical use of TDocGen

TDocGen has been implemented according to the previous description, and consists of two programs: one to create histograms from an existing text collection, and a second program to create the actual document collection. Creating histograms is a relatively time-consuming task, but by separating this into a separate task this only have to be performed once. Histograms are stored in separate histogram files that can also be distributed, so that it is it is not actually necessary for every user to retrieve a large

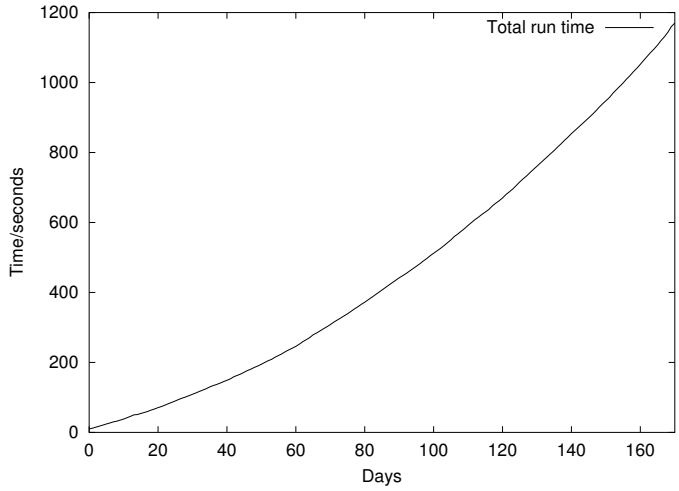


Figure 1: Total run time as a function of number of generations, using pattern I.

collection. This is a big saving, because a histogram file are much smaller than the text collection it is made from, for example, the compressed size of the text collection we use is 1.8 GB, while the compressed histogram file is only 10 MB.

The result of running TDocGen is a number of compressed archive files. There is one file for each day/generation, and the file contains all document versions that existed during that particular time instant. The words in the documents will follow Heaps' and Zipf's laws, but because the vocabulary/histogram has a fixed size, Heaps' law will only be obeyed as long as the size of a the documents in a particular generation is smaller than the data set which the vocabulary was created from.

7 Evaluation of TDocGen

The purpose of the output of a document generator is to be used to evaluate other algorithms or system, and it is therefore important that the created documents have the quality in terms of statistical properties as expected. It is also important that the document generator has sufficient performance, so that the process of creating test document does not in itself become a bottleneck in the development process. In this section, we will study the performance of TDocGen, and the quality of the created document collection.

TDocGen creates documents based on a histogram created from a base collection. In our measurements we have used several base collections. The performance and quality of results when using these is mostly the same, so we will here limit our discussion to the largest collection we used. This collection is based on a document collection that is available from Project Gutenberg [14]. The collection contains approximately 10,000 books, and our collection consists of most of the texts there, except some documents that contain contents we do not expect to be typical for document databases., e.g., files contains list of words (for crosswords), etc.

7.1 Cost

In order to be feasible in use, a dataset generator has to provide results within "reasonable time". Figure 1 illustrates the time it takes to create a temporal document collections using pattern I in

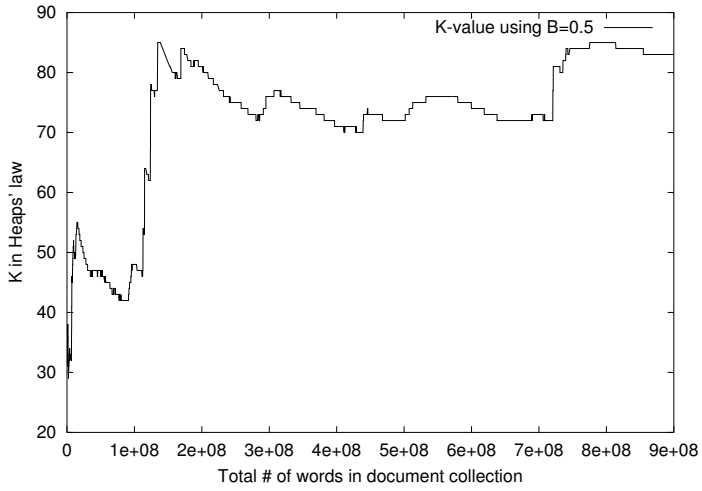


Figure 2: Value of K while processing the base collection.

Table 1.

In this case, 171 generations are created. As can be seen, the elapsed time using the actual parameters is polynomial, but close to linear. The reason is that the time to create one generation increases: the size of the documents in each generation is monotonically increasing, and so is the time required to store these (all documents in one generation is stored in the archive file).

The total collection created in this experiment is quite large, a total of 1.6 million files are created, containing 13.3 GB of text. The size of the last generation is 159MB of text in 18,000 files, and it is obvious that the main bottleneck here is disk access. However, the elapsed time is less than half an hour, which should be low for most uses of such a generator.

7.2 Quality of generated document collections

We will study the quality of the generated document collections with respect to word distribution and number of unique words. However, in order to be able to explain the results and compare them with real-world documents, we start we a study of the properties of the base collection we used, i.e., the Project Gutenberg Collection.

7.2.1 Base collection

The base collection was first studied with respect to Heaps' law. In the part of the collection used in out experiments, we have measured a calculated value of K given $\beta = 0.5$ as the collection is read and total number of words and vocabulary size is determined. As Figure 2 shows, $K \approx 35$ for the first amount of documents, then jumping up to higher values, finally stabilizing around $K \approx 80$. The reason for the jumps and the relatively large value of K is that the collection contains texts in different languages. On the figure we see the "jumps" each time documents in a new language are considered. The first major jump is when the first German languages are processed, the next when some French documents are processed, etc. This gives a K value larger than a single-language collection would give. However, we assume that in a typical document collection this would often be the case. For example, in a collection of documents in almost any company in Norway you can expect to find

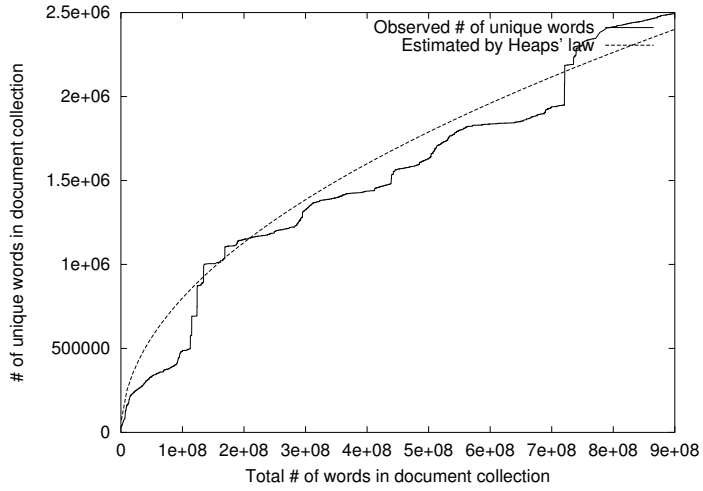


Figure 3: Comparing the size of vocabulary in the base collection with value calculated using Heaps' law with $K = 80$ and $\beta = 0.4$.

documents in English and Scandinavian languages in addition to documents written in Norwegian. Figure 2 illustrates well how the document collection follows Heaps' law with $K = 80$ and $\beta = 0.4$.

The study of word distribution can be done by creating histograms of the document collection. Figure 4 shows the number of occurrences for each word compared to the value calculated by Zipf's law using $P_1 = 0.05$. In order to make it easier to compare we have used logarithmic scale on both axes on the figure, and we see that the distribution is close to Zipf.

7.2.2 Generated collection

The first study is word distribution in the created collections, and we perform this study on the first and last snapshot collections created during the tests using the two patterns in Table 1. As Figure 5 show, words distribution is Zipfian in the created collections. It should also be mentioned that an inspection of the highest ranked words shows that the most frequently occurring words are "the", "of", "and", and "to". This is as expected in a text collection that is based on documents that are mostly in English.

The second study is to see whether the document collections is according to Heaps' law as is expected. Figure 6 shows the value K for the created collections, and we see that K is well within reasonable bounds.

8 Conclusions and further work

In research in temporal document databases, different algorithms and approaches are emerging, and in order to be able to compare these good test collections are important. In this paper we have described how to make temporal document collections, how this is realized in the TDocGen temporal document generator, and we have provided a study of the quality of the document collections that are created by TDocGen.

TDocGen have been shown to meet the requirements for a good temporal document collection generator, and is available free for download at the TDocGen Resource Site [13]. Also available are

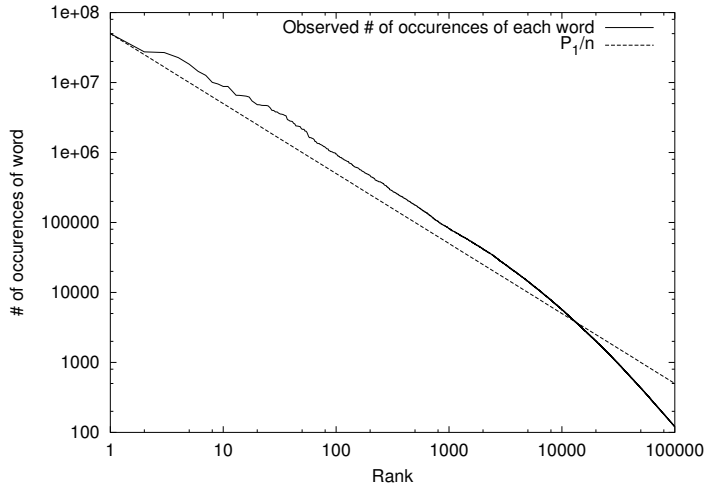


Figure 4: Number of occurrences for each word in the Project Gutenberg Collection, compared to the value calculated by Zipf's law.

ready-made histograms, including the one used for the experiments in this paper, based on 4 GB of text documents from Project Gutenberg [14]).

If users want to generate temporal document collections especially suited for their own domain, it is possible to use own existing documents o as basis for building the histograms used to generate the temporal document versions. It should also be noted that the generator can also be used to create non-temporal document collections when collections with particular characteristics are needed.

References

[1] D. Barbosa, A. O. Mendelzon, J. Keenleyside, and K. A. Lyons. ToXgene: a template-based data generator for XML. In *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data*, 2002.

[2] B. E. Brewington and G. Cybenko. How dynamic is the Web? *Computer Networks*, 33(1-6):257–276, 2000.

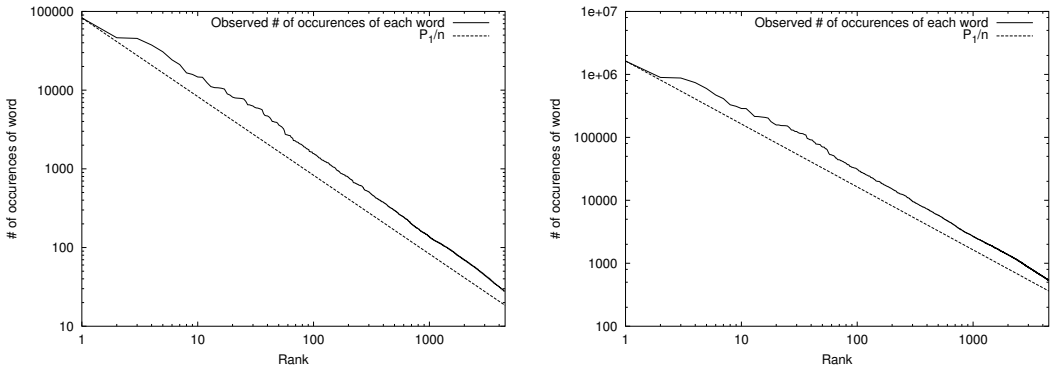
[3] G. Cobena, S. Abiteboul, and A. Marian. Detecting changes in XML documents. In *Proceedings of the 18th International Conference on Data Engineering*, 2002.

[4] D. Fetterly, M. Manasse, M. Najork, and J. L. Wiener. A large-scale study of the evolution of Web pages. *Software - Practice and Experience*, 34(2):213–237, 1996.

[5] Glasgow IR Resources, Publically available IR test collections, `http://www.dcs.gla.ac.uk/idom/ir_resources/test_collections/`.

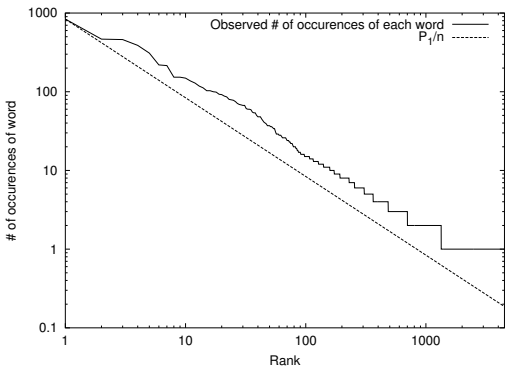
[6] H. S. Heaps. *Information Retrieval: Computational and Theoretical Aspects*. Academic Press, Inc., 1978.

[7] Internet Archive. `http://archive.org/`.

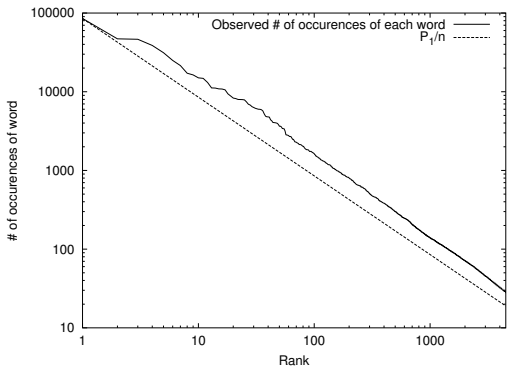


(a) Pattern I, first generation.

(b) Pattern I, last generation.

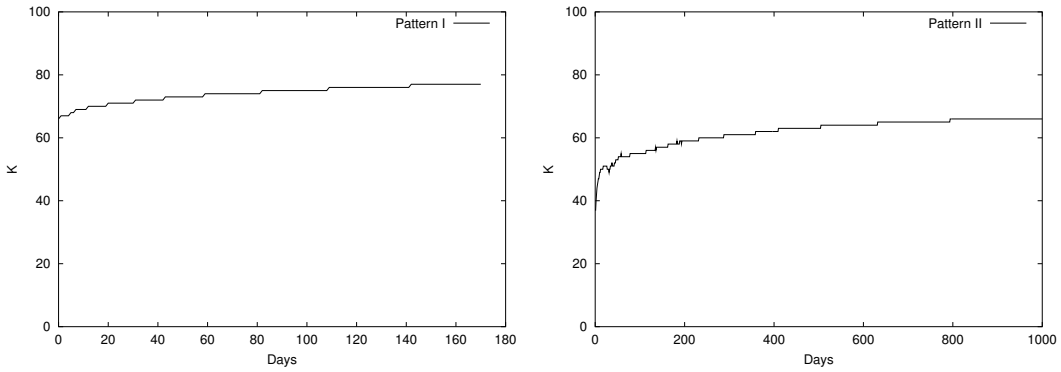


(c) Pattern II, first generation.



(d) Pattern II, last generation.

Figure 5: Comparison of ideal Zipf distribution and the words in the actual created document collections.



(a) Pattern I.

(b) Pattern II.

Figure 6: Value of K for different generations of created documents.

[8] G. Kazai, N. Gvert, M. Lalmas, and N. Fuhr. The INEX evaluation initiative. In *Intelligent Search on XML Data, Applications, Languages, Models, Implementations, and Benchmarks*, 2003.

[9] W. Li. Random texts exhibit Zipf's-law-like word frequency distribution. *IEEE Transactions on Information Theory*, 38(6), 1992.

[10] K. Nørvåg. The design, implementation, and performance of the V2 temporal document database system. *Journal of Information and Software Technology*, 46(9):557–574, 2004.

[11] K. Runapongsa, J. M. Patel, H. V. Jagadish, and S. Al-Khalifa. The Michigan Benchmark: A microbenchmark for XML query processing systems. In *Efficiency and Effectiveness of XML Tools and Techniques and Data Integration over the Web, VLDB 2002 Workshop EEXTT and CAiSE 2002 Workshop DTWeb. Revised Papers*, 2002.

[12] A. Schmidt, F. Waas, M. L. Kersten, M. J. Carey, I. Manolescu, and R. Busse. XMark: a benchmark for XML data management. In *Proceedings of VLDB'2002*, 2002.

[13] TDocGen Resource site, `http://www.norvag.com/tdocgen/`.

[14] Project Gutenberg, `http://www.gutenberg.net`.

[15] Text REtrieval Conference, `http://trec.nist.gov/`.

[16] UCI Knowledge Discovery in Databases Archive, `http://kdd.ics.uci.edu/`.

[17] G. K. Zipf. *Human Behaviour and the Principle of Least Effort: an Introduction to Human Ecology*. Addison-Wesley, 1949.

[18] Zipf's law, `http://www.nist.gov/dads/HTML/zipfslaw.html`.