Generative models for simulating text collections

David Hawking Bodo Billerbeck Nick Craswell Paul Thomas

Microsoft

ABSTRACT

Simulated test collections find application in situations where real datasets cannot easily be accessed due to confidentiality concerns or practical inconvenience. They can potentially support experimentation, tuning, validation, performance prediction and hardware sizing. Simulation depends upon models and the fidelity and generality of the models determines the value of the simulation and the range of uses to which it can be put.

We present and analyse a range of models for each of document length distribution, term frequency distribution (for independent and non-independent cases), term length and textual representation, generation of corpus-compatible queries, and corpus growth. We present results of emulating existing corpora and for scaling up corpora by two orders of magnitude. We show that simulated collections generated with relatively simple methods are suitable for some purposes and can be generated very quickly. Indeed we show that a simple lightweight corpus generator can be embedded into an indexer for the purpose of efficiency studies.

Limitations: Here we restrict ourselves to the simulation of an unstructured corpus of plain text documents. Ideally, a generator would also model document structure, hyperlinks, authorship, creation dates, attachments, and file system organization. A more basic limitation is that our methods do not generate real words or natural language structures. Although LSTM methods are capable of generating more plausible language, none of the methods we can envisage are able to generate text realistic enough to support work in natural language understanding.

1. INTORDUCTION

There are a number of real world applications in which it is useful to generate an artificial text corpus with a corresponding set of test queries. A prime example is when a cloud hosting company builds search systems for hosted clients whose data is private and/or confidential. The hosting company wishes to measure and improve indexing performance and query processing latency and throughput, to tune a ranking function, and to experiment with add-on features such as query suggestion and spelling correction. If privacy and confidentiality constraints allow an automated agent to extract sufficient properties of a client's data then the hosting company can build a simulated collection and study it.

For academic research, simulated corpora can permit exact reproducibility of efficiency experiments and meaningful study of the scalability of retrieval systems. A corpus can be shared exactly with other researchers by communicating at most a few kilobytes of parameters, while completely eliminating confounds due to differences in tokenization and other lexical issues. Furthermore, a parameterized collection generator allows data sets to be engineered to specification, permitting focused study of specific efficiency or effectiveness factors. Finally, a comprehensive generative model also models the growth of a corpus, allowing retrieval systems to be tested on data sets as they might become in the future.

In this work we report a quite thorough study of many aspects of the collection simulation problem. We study the properties of nine different corpora and discuss alternative generative models for: document length distribution; term frequency distribution assuming independence; term dependence; and generation of the textual representation of terms. We have built a system capable of generating corpora according to multiple alternative models and we compare the properties of corpora emulated with its aid with those of the corresponding originals.

In order to build models of corpus growth we study samples of the nine corpora and plot changes in properties as the samples grow from 1% to 100% of the parent. Then, starting with a static model of the smallest sample and modifying it according to a growth model, we generate a corpus 100 times larger and show that its properties are quite similar to the original. The growth in vocabulary as corpus size increases, as described by Herdan and Heaps, prevents us from using the original corpus vocabulary when scaling up, even when the original data is public.

Azzopardi et al. propose methods for generating known item queries from a collection. In order to permit the study of query processing over artificially genertated corpora, we implement Azzopardi's best-performing method and augment it with a method for generating queries comprising frequently-occurring words which never co-occur.

It is not exactly known which properties of a corpus have the greatest influence on efficiency and observed effectiveness of a retrieval system. Indeed, the relative importance of the properties likely depends on both the algorithm and the characteristics of the hardware architecture. A thorough study of these dependencies is beyond the scope of the present work but for illustrative purposes we compare indexing and query processing rates of Indri over a tokenized form of WT10g with those for an emulated version.

A real corpus can be emulated with different degrees of fidelity. It may be impractical to faithfully emulate a corpus either due to the cost of generation or due to the potential to leak information in a privacy scenario. Our study sheds light on the range of degrees of fidelity and the costs of achieving a specified level. If the degree of fidelity required is not high it may be possible to build the generation model into the retrieval system under study and to generate words, word representations and document boundaries on the fly without ever instantiating the generated corpus. Successful on-the-fly generation for efficiency studies may require that generation of sampled terms and document boundaries is no slower than reading them from a disk, and that the generation algorithms have no greater interference on CPUs and memory caches. We demonstrate that, for certain levels of fidelity, this is actually possible.

1.1 A simple-minded algorithm for text generation

Language models used in IR are generative models but they are typically used to assess the likelihood that a document and a query were generated from the same model, rather than to actually generate text. They could, however, be used for synthetic corpus generation, as illustrated in the following simple-minded algorithm BASELINE:

1. Extract a unigram language model (LM) from a real corpus $\mathcal{C}$, representing the language model as a cumulative probability histogram, in which observed terms (x-axis) are ranked by descending probability. (Zipfian model. [95])
2. Repeat the following until the desired amount of text has been generated:
   (a) Generate a pseudo random document length l (in words) from a length distribution derived from $\mathcal{C}$.
   (b) Generate l uniformly distributed random numbers in the range $0 \ldots 1$. Use the cumulative probability histogram to map each of them to a term and emit that term.
   (c) Emit a document boundary marker.

The result $\mathcal{E}$ should closely match the modeled properties of the original corpus but the method suffers from a number of limitations:

1. The vocabulary of the synthetic corpus cannot be larger than that of the emulated one. Terms not found in the base corpus cannot be emitted. That means that scaling $\mathcal{E}$ beyond the size of $\mathcal{C}$ will result in decreasing fidelity since laws due to Herdan [49] and Heaps [46] tell us that the vocabulary of a corpus should grow with the size of the text. Williams and Zobel [93] have confirmed this growth in much larger datasets than available to earlier authors.
2. Because we are sampling "with replacement" the expected vocabulary size of $\mathcal{E}$ will be less than that of $\mathcal{C}$ since some terms will likely not be sampled.
3. Terms are generated independently. Phrase and sentence structure are not modeled and nor is the tendency of term occurences to cluster in documents.
4. The size of the model (histogram) may be very large. A very large corpus may contain hundreds of millions of distinct terms.

We present some experimental results for BASELINE in Section 3.0.1. Naturally, the range of applications for a synthetic corpus can be increased if more sophisticated modeling is undertaken. A number of more sophisticated generative models are available.

1.2 More sophisticated models

Topic modeling via Latent Dirichlet Allocation can be used to better model the tendency of subject words to cluster [18]. Separate LMs are generated for topics represented in a corpus and documents are generated as a mixture of a subset of topics. For instance, Wallach et al. [91] propose a method to generate held-out documents given the remainder of a collection.

N-gram language models can at least partly model the natural language structure of real text. A corpus may be modeled as a mixture of unigrams and higher-order "grams". While taking n-grams into account has been shown useful (see e.g. Bendersky et al. [15]), unsurprisingly there is little on generating documents or even whole collections using n-grams alone.

Markov chains. Sequences of words in natural language text could in theory be modeled using a Markov chain [54] but with vocabulary sizes in the hundreds of millions, the size of the transition probability matrix and the effort required to calculate it would be prohibitively large.

Recurrent neural nets (RNNs) have been used to model language structure even more accurately, by modeling longer range dependencies. Sutskever et al. [83] have demonstrated how this technique can, with appropriate training data, generate fragments of text which convincingly match the style of e.g. Shakespeare or a technical report. The resulting text conveys no real meaning but presents groups of words in plausible order, and with plausible punctuation.

Natural language generation from real world semantics, that is, the generation of artificial text which conveys real meaning, seems at this point in time to be restricted to narrow semantic domains such as sportscasting, e.g. [30].

Most applications in the "experimenting with private data" area would require sophisticated models like these, but are constrained by the need to maintain security of private and confidential information. A very accurate model might enable the deduction of confidential information. For example a sophisticated model may allow an attacker to determine that the sentence, "Our company will acquire company X" has higher probability than the sentence, "Our company will not acquire company X". It remains to be seen whether a useful balance can be struck between representational fidelity and preservation of privacy.

The more sophisticated models listed above remove the need to assume term independence but still require large models, are still limited to a fixed vocabulary, may require more CPU time to generate terms, and generally require more effort in model generation.

For efficiency and scalability experimentation there are practical advantages in overcoming these limitations, even if the modeling is less faithful to natural text. Overcoming the limit of fixed vocabulary seems critical to scalability experimentation; smaller models allow easier sharing of experimental data with colleagues; and efficient generation of text enables faster experimental turn-around times.

1.3 Efficiency and scalability

Studies of the efficiency of text indexing algorithms have often reported time and space results for the indexing of a single corpus (such as GOV2[32]), e.g. [60], [37] and many referenced in [96]. Empirically assessing the scalability of algorithms is difficult because of heterogeneity between corpora of different sizes. The data size of one corpus may be x times larger than that of another but the ratio of the magnitude of the indexing tasks may be very different from x due to different vocabulary sizes, different term frequency distributions, and different proportions of indexable text.

Another limitation of indexing efficiency studies is that published results are difficult to reproduce because of differences in details of how each indexer scans text. Indexers differ widely in the following aspects: recognition and conversion of character sets; stopword handling; word-break characters; stemming; parsing of markup; recognition and handling of text in "fields"; handling of comments and no-index sections, obedience to 'robots' metatags; indexing of element attributes; recognition of XML entities; and handling of mark-up which isn't well-formed.

There is now considerable interest in the study of memory-resident indexing and retrieval algorithms for NUMA (Non-Uniform Memory Access) architectures. Since 2010 or so, it has been possible to purchase off-the-shelf servers configured with multiple terabytes of RAM. Such servers are characterized by great variations in memory access speed, due to multiple levels of cache, to the need to maintain cache coherency, and to the closer association of banks of RAM to just one of several CPU chips. For example, the best access latency in clock cycles for different levels of memory in the Intel Core i7 Xeon 5500 series is 4 for L1 cache, 10 for L2 cache, 40 for L3 cache, 60 for local DRAM and 100 for remote DRAM.[1]

Meaningful empirical comparison of retrieval algorithms in NUMA environments requires large corpora, which are difficult to share with others – test collections like ClueWeb09 have been shared by shipping arrays of hard disks.

Logically, a synthetic corpus derived from a real one may be considered to be a highly compressed version of the original. Instead of shipping hard disks of text data we can ship a much smaller model plus a small set of parameters, a random seed, and the generator code.

The compression factor can increased significantly if we generate the term representations, and mathematically model the term probability distribution rather than recording a full histogram. In the algorithm presented earlier, if the random number generator picks, say term 13507, then we look up the term table in the language model and find that that corresponds to "adumbrate". If we instead generate the term representations, then an algorithm maps term number 13507 to a unique sequence of characters rather than to a known word. For example, it may generate "xx$p2". Note that a term representation generator is needed to solve a limitation mentioned above – that scaling up a corpus should not assume a fixed vocabulary.

[1]`https://software.intel.com/en-us/forums/intel-manycore-testing-lab/topic/287236` accessed 07 Jan 2016.

A generator for term representations opens another exciting possibility. With such a generator we may never need to physically instantiate the corpus. Instead, the term generation could be done internally to the indexer, avoiding the need to store text on disk and to read it during indexing. We call this *internal generation.* For internal generation to be practical, the random selection of the term number and the generation of its character representation, must require minimal memory to avoid interfering too much with use of memory caches by the indexing algorithm being studied. It should also take no more CPU time than would be used by reading and parsing a term from the real collection.

Synthetic corpora, whether instantiated or not, can be used for studies of efficiency and scalability of indexing algorithms. Combined with simulated known-item queries generated by methods similar to those described by Azzopardi et al. [6][7], synthetic corpora can potentially also be used to study the efficiency of query processing algorithms. Indeed, since known item queries provide for limited evaluation without the need for human judgments, it is possible to simulate test collections and to use them to compare and tune retrieval methods.

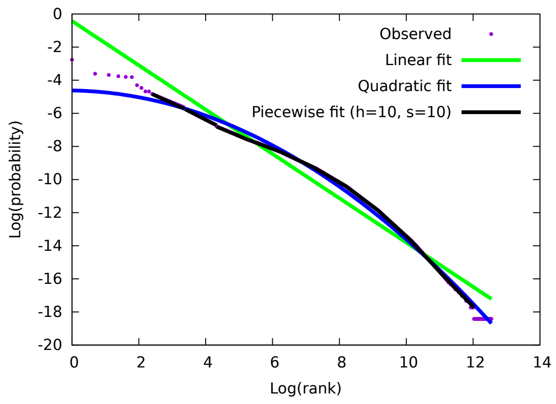
1.4 Outline

In this paper we study the nine different base corpora documented in Table 1. The selection of these corpora reflects an interest on our part in indexing and querying large numbers of short texts such as tweets. However, TREC-AP is a TREC corpus comprising the full text of the AP newswire articles, and ClueWeb12 bodies comprises the extracted full text of 20 million ClueWeb12 documents. A third collection of longer documents is Indri-WT10g. It comprises the document boundaries and indexable tokens extracted by the Indri[2] retrieval system.

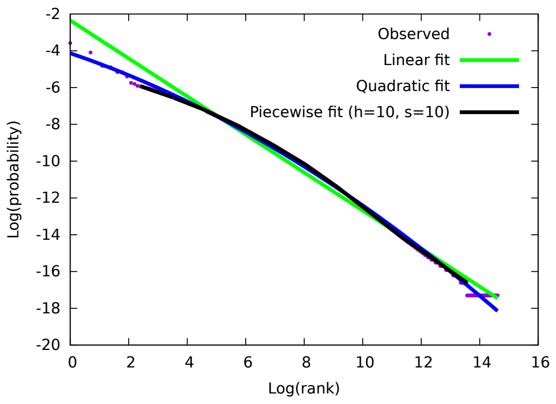
We focus on emulating the following corpus characteristics:

1. Number of documents.
2. Document length distribution.
3. Total number of term occurrences (postings) P. This is the main determinant of the scale of an indexing task.
4. Vocabulary size $|V|$. This determines the size of the term dictionary and influences the time taken by term insertions and lookups.
5. Term probability distribution. This determines the distribution of postings list lengths and the length of the longest postings list. The best choice of index data structures and algorithms may depend upon this.
6. Term dependence. We try to capture the tendency of certain pairs of words to associate within documents, the tendency of certain words to appear more often in particular documents than would be expected by chance, and the tendency of certain word n-grams to appear more often than would be expected by chance.
7. Characteristics of term representations. The length of terms and the patterns of characters in the representation may affect hash table collision rates or the efficiency of tree structures.

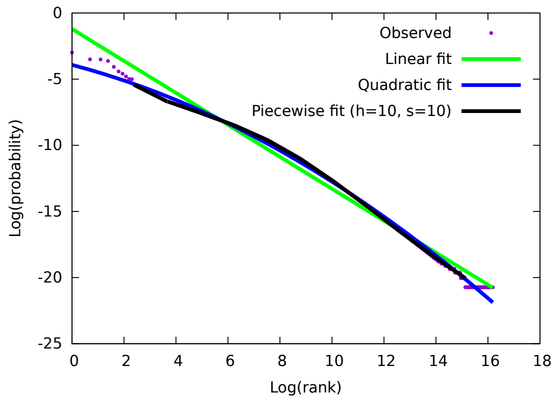
[2]`http://www.lemurproject.org/indri/`



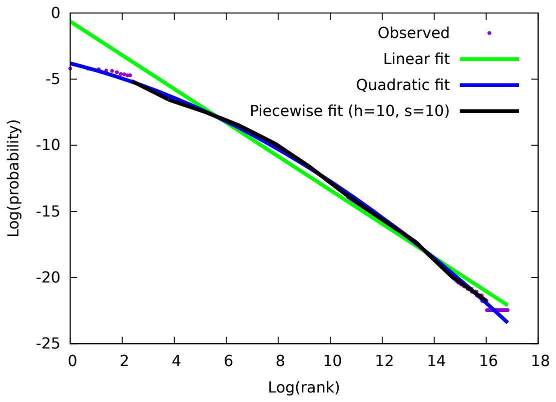
(a) TREC-AP



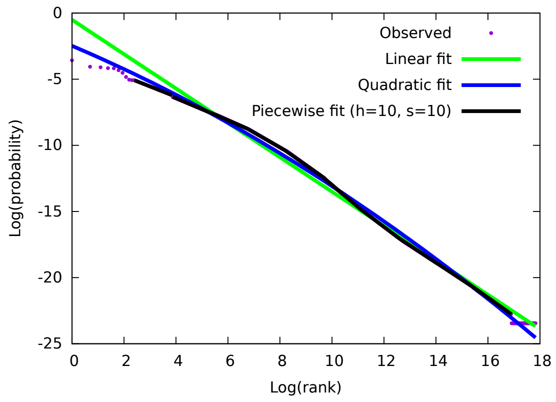
(b) Wikipedia titles



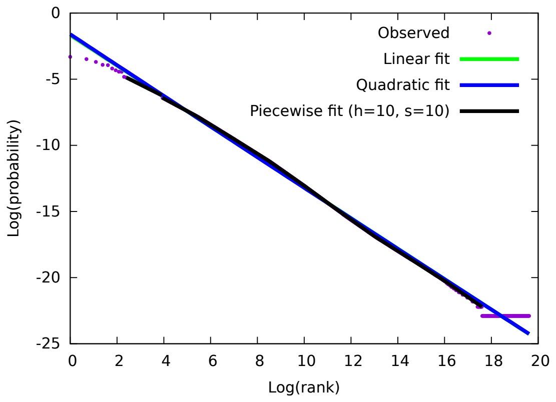
(c) Academic paper titles



(d) ClueWeb12 titles



(e) ClueWeb12 bodies



(f) Tweets

Figure 1: Term probability distributions in log-log space for six of the nine real corpora – others omitted for reasons of legibility. Lines of linear, quadratic and piecewise linear fit are also shown. Note that the piecewise fit has ten segments and covers neither the 10 head terms nor the singletons.

Name	No. Docs	Vocab. Size	No. Term occurrences	Remarks
TREC-AP	0.24M	0.29M	100.1M	TREC newswire covering 24 months
Popular queries	100.00M	6.41M	381.0M	popular Web queries
Song lyrics	27.92M	0.91M	153.0M	song lyric lines
Wikipedia titles	11.06M	2.32M	32.7M	Wikipedia titles (inc. redirects)
Academic paper titles	89.53M	11.98M	1003.6M	collection of paper titles
ClueWeb12 titles	728.88M	21.78M	5687.3M	ClueWeb12 document titles
ClueWeb12 bodies	23.30M	61.11M	5340.1M	subset of ClueWeb12 bodies
Tweets	710.62M	340.75M	8843.7M	daily Twitter samples covering 18 months
Indri-WT10g	1.94M	5.65M	1040.1M	TREC WT10g words extracted by Indri

Table 1: Details of the corpora used in sampling experiments.

In Section 2 we consider a range of models for document length distribution. In Section 3 we explore models of the term probability distribution, assuming independence. In Section 4 we look at dependence models, and the important question of how to measure whether the associations between terms have been properly modeled. Section 5 discusses the generation of sequences of characters to represent terms, when it is not possible to use the lexicon of a corpus being emulated. Section 6 discusses methods for generating corpus-compatible queries and judgments. In Section 7 we show how the characteristics of samples of each corpus change as we increase the size of the samples from 1% up to 100%. From these observations we derive growth models.

In Section 8 we attempt to emulate each corpus $\mathcal{C}$, by modeling the properties listed above, and using models, including Baseline to control the generation of an emulated corpus $\mathcal{E}$. We compare the properties of $\mathcal{E}$ and $\mathcal{C}$ and conclude that, for the best type of model, they are sufficiently similar to permit their use in studies of indexing efficiency. Section 9 describes scaling-up experiments which attempt to emulate $\mathcal{C}$ from the average properties of a number of 1% samples of $\mathcal{C}$. Fidelity is less than in the direct emulation case, but the results are still quite promising.

Section 10 discusses practical issues and trade-offs in test collection simulation, and the possibilities and limitations of on-the-fly generation. Section 12 reviews related work.

2. MODELING OF DOCUMENT LENGTHS

In many applications, approximate modeling of the distribution of document lengths will suffice. However, there are scenarios in which accurate modeling is desirable. For example, the efficiency of some indexing or query processing algorithms may be affected by very long or very short documents. Furthermore, distributions of term cooccurrences, distributions of TFs, and topic mixtures are all affected by the distribution of document lengths.

We have undertaken a study of a number of different document length models. In the following, the length of a document is defined as the number of indexable words it contains.

2.1 Left-truncated Gaussian length model

Mean and standard deviation of the primary corpus are easily calculated. We can then sample lengths from a normal distribution with those parameters using an appropriate function such as ceil() to convert real values to integer lengths. If a zero or negative length is sampled it is rejected, leading to a shift in the actual mean of the truncated distribution. We applied simple multiplicative correction factors to both mean and standard deviation to more accurately model the primary corpus.

Efficient generation of normally distributed random numbers is possible using the method of Marsaglia and Bray [66].

We found that for many corpora the truncated normal distribution significantly under-generates longer documents found in the original distribution. Furthermore, we see no theoretical reason to expect that document lengths should be normally distributed. Thus, apart from implementation convenience, there is little to recommend this model.

2.2 Negative binomial length model

If we added a symbol representing end-of-document to our term distribution and emitted it with fixed probability p, then each act of sampling would constitute a Bernoulli trial in which the emission of an end-of-document symbol represents the "failure". In this scheme, the number of successes (i.e. length of document in words) before a failure would be distributed according to a negative binomial.

Unfortunately, the form of negative binomial which fits actual document length distributions best tends to be one for multiple failures. Even if we interpret the additional symbol as a possible-end-of-document, it is hard to suggest a good reason why document authors would always stop at the k-th of k possible stopping points.

Although the negative binomial has some relation to a plausible author model, it's ability to fit observed data is less good than that of the gamma distribution.

2.3 Gamma distribution length model

A gamma distribution can be specified using two real parameters, shape and scale. Depending upon the values of these parameters the form of the distribution can approximate normal, exponential, uniform and other forms. This flexibility allows gamma to model observed length distributions for many real corpora with greater fidelity than was the case with the two preceding models. Unlike the Gaussian, gamma can emulate the heavy tail found in many observed document length distributions.

We estimated the parameters for a gamma distribution using the method and code due to Minka [72]. We implemented sampling from a specified Gamma distribution using the efficient method of Marsaglia and Tsang [67].

The gamma distribution can achieve good fidelity on some corpora, requires only two parameters, and can easily control the generation of a corpus much larger than the original. Unfortunately, the gamma distribution, like the Gaussian and negative binomial, is unable to model actual distributions with multiple points of inflection. Such distributions seem to be reasonably common and may correspond to corpora consisting of a mixture of document types. E.g. a mixture of abstracts and articles. Indeed, the distribution of document lengths in a corpus seems to be essentially arbitrary.

For this reason, we turned toward a piecewise linear model.

2.4 Piecewise linear length model

Initial experiments with piecewise segments defined as either fixed intervals on the horizontal axis or fixed areas under the curve were unsatisfactory. Fine segments were needed to model perturbations in the curve but led to high numbers of segments.

An adaptive method was tried which started with a single segment and split it at the point of maximum deviation of actual from expected, provided that deviation was larger than a threshold. This method was capable of modeling multi-modal distributions but again encountered problems with certain corpora. We were unable to find a combination of bucket size and split threshold which achieved sufficient modeling accuracy on all the corpora while using only a modest number of segments. Small bucket sizes were needed to model narrow peaks but led to jitter and large numbers of segments.

Given that piecewise linear provides no explanatory power, relies on heuristics, and requires large numbers of parameters to achieve good results across corpora, it seemed sensible to abandon the attempt to provide a model, and to instead use the actual histogram of observed lengths.

2.5 Length models relying on histograms

A histogram of observed document lengths for an original corpus is reasonably compact, even with a bucket size of 1. When emulating the corpus we could generate document lengths by traversing the histogram, first generating all the documents of length 1, then all those of length 2, and so on up to the maximum observed length. This would be perfectly acceptable if we were able to assume that the order of occurrence of document lengths was arbitrary and had no effect on anything we wanted to measure or observe on the emulated data.

If this were not the case, we could use the histogram as a distribution from which to sample. Sampling with replacement is very straightforward to implement, but the resulting length distribution only approximately matches the original.

Sampling without replacement would be very inefficient if the cumulative frequency distribution were adjusted after every sample. An alternative approach samples according to the unaltered original distribution, decreases the count in the selected bucket and rejects the sample if the bucket is empty. This sampling process slows down as buckets empty due to the increasing probability of a rejection, but the time cost may be small as buckets tend to empty at around the same time.

If the number of distinct document lengths is high then it may be necessary to use a larger bucket size. When the sampling process picks a bucket, the length chosen could be uniformly sampled from the range of lengths represented by the bucket or perhaps a fixed representative length could be emitted. There would be advantage to using a dynamic bucket sizing scheme which uses narrow buckets for the shorter lengths where accurate modeling may be more important and increases them in arithmetic, geometric or other sequence.

Scaling up a histogram in order to generate a corpus larger (or smaller) than the original is easily achieved by multiplying the bucket counts by a constant factor.

Perfect fidelity can be achieved when emulation is based on the length histogram from the original corpus. On the other side of the ledger, the histogram may be quite large for some corpora, which may result in adverse memory requirement consequences. Furthermore, particularly when scaling up, smoothing would be needed to allow the possibility of generating lengths which were not observed in the original corpus.

2.6 Length models: which to choose?

[[TODO: *Tabulate the model sizes for each of the methods.*]] [[TODO: *Paul: âĂć Each of the models in Âğ3.1, 3.2, âĂę could be illustrated with examples of the resulting fit, i.e. a pointer into the table weâĂŹve been building. That table could easily be extended to include the total storage needed for each model (or model/seed-corpus pair). âĂć A plot or two would also illustrate the alternatives nicely, and the problem with e.g. bimodal distributions in the seed corpus. Again we can produce those easily.*]]

We recommend choosing the gamma model if mathematical simplicity and a small number of parameters is desired, unless the observed length distribution for the primary corpus is clearly multi-modal. It would be possible to model a distribution as a mixture of gamma distributions, but the complexity of doing this would reduce the advantage and argue for a histogram-based approach.

In other circumstances, the use of a histogram-based method is preferred despite being a highly-overfitted model. (Can it really be called a model?) The decision on whether to scan the histogram or to sample by one of the methods discussed above should be made based on the requirements of the intended application for the synthesized corpus. The optimal bucketing scheme and whether to implement some form of smoothing should be determined after analysis of the data.

3. TERM PROBABILITY DISTRIBUTION, ASSUMING INDEPENDENCE

We start by exploring the use of the full language model and the BASELINE algorithm and move on to a model which is considerably more sophisticated, though it still assumes term independence.

3.0.1 *Generating text with the* BASELINE *algorithm*

Corpus	P	$\|V\|$	singletons
$\mathcal{C}$	1004M	11.98M	7.187M
$\mathcal{E}_{10\%}$	100.4M	2.715M	1.676M
$\mathcal{E}_{100\%}$	1004M	9.053M	3.256M
$\mathcal{E}_{1000\%}$	10040M	11.98M	0.003239M

Table 2: Emulation of Academic paper titles corpus using the BASELINE algorithm.

As noted in the introduction we can model the term probability distribution with a cumulative term probability histogram. We have done this for the Academic paper titles corpus described in Table 1. We used the BASELINE algorithm to generate simulated corpora of 10%, 100% and 1000% of the number of postings in the original corpus. The cumulative term probability histogram was derived from the full Academic paper titles corpus. To test that our algorithm and the underlying pseudo random number generator (Mersenne Twister [68]) were working correctly we mathematically predicted what percentage of singleton terms in the language model would fail to be selected in $P = 1004$ million draws – the number of postings in Academic paper titles. Since each draw has a $(P-1)/P$ probability of missing a particular singleton and there are P draws, we can raise $(P-1)/P$ to the power of P to get the probability of missing a singleton entirely. The predicted and observed proportions of missed singletons were 0.36785 and 0.36788 respectively, showing agreement to 4 decimal places. BASELINE misses not only singletons but also terms with higher frequency by through the same process, leading to significant overall loss of vocabulary for $\mathcal{E}_{100\%}$.

The results for BASELINE are tabulated in Table 2. When simulating a corpus of the same size as $\mathcal{C}$ the vocabulary size is only 76% as large as the original, largely due to the failure to select a large percentage of the singleton terms. That is to be expected. Note that another method of creating $\mathcal{E}_{100\%}$ which would include all words and exactly match the original term probability distribution would be to randomly shuffle the terms in the corpus. However, shuffling is not capable of creating corpora of different size to the original.

When the number of samples drawn from the language model is increased by a factor of ten ($\mathcal{E}_{1000\%}$) virtually all the terms are selected. This confirms that the random number generator has sufficient resolution. However, the vocabulary size can not and does not exceed the size of the model and the number of singleton terms reduces to almost zero (because former singletons are drawn more than once), confirming the unsuitability of this algorithm for modeling growth.

The model sizes for the BASELINE algorithm are large. In the case of the Academic paper titles corpus, the size of the model is approximately 12 million elements, each comprising a text string and a high-precision probability value. For the Tweets corpus the corresponding figure would be more than 340 million.

3.0.2 Mathematical modeling of term distributions.

Many believe that the term probability distribution in a text corpus follows Zipf's law [95]. That law states that the probability $Pr(t_r)$ of the r-th most frequent term is proportional to a negative power α of the rank r. We can relate this to a continuous probability function in which r is real:

$$Pr(t_r) \propto r^{\alpha}$$

If this model applied perfectly, a log-log plot of probability against rank would show a negatively sloping straight line. However, it is frequently observed that real-world frequency distributions conform to Zipf's law less well at both high and the low frequencies. Baeza-Yates [9] reports that singleton queries are far more heavily represented in query frequency distributions than would be predicted by a power law which fits the upper part of the distribution.

A considerable literature has accumulated describing approaches to achieving better modeling of such "Zipfian" distributions. One of them [62] proposes a "parabolic fractal" model which would lead to a quadratic fit in log-log space. Figure 1 shows log-log plots of term probability against rank along with both linear and quadratic lines of best fit.

Parameter(s)	Explanation
P	Total of word occurrences.
$\lvert V\rvert$	Vocab. size.
w_1	The proportion of distinct words which occur only once.
$H_1 \ldots H_h$	The percentages of all term occurrences represented by each of the most frequent h words.
$S_1 \ldots S_s$	The 4-tuples of the s segments used in piecewise linear approximation to the middle part of the curve.

Table 3: Parameters of our static corpus model.

Note that only a subset of points are shown – the full number of points is too large to display. Least squares fitting is carried out on the plotted subset of points. The points in the subset were chosen such that each point in the subset was more than a small distance ϵ (in log-log space) away from the others. The number of points was thus reduced to the order of a thousand rather than tens or hundreds of millions. One effect of this procedure is to reduce the bias of fitting toward the tail due to its huge preponderance of points.

In general the quadratic fit is more accurate than the linear, but neither fits perfectly. In particular, the pattern of the first few head points varies substantially across corpora. Given this, and given the potential importance of the frequency of the most common terms to indexing efficiency, we chose to record the actual probabilities of the first h terms as parameters of the model. By inspection of the nine corpora, we concluded that $h = 10$ would adequately cover the head deviations from linear or quadratic models. We have used $h = 10$ in our experiments but it could be that a smaller value would suffice, or a collection-dependent value would be better. Reducing h would reduce the size and complexity of the model, but the effect would be small.

We also decided to model the singleton terms separately. The relevant model parameter is just the proportion of the vocabulary which has a frequency of one.

Finally, we chose to model the middle part of the curve using a piecewise linear model with s segments. The segment ends are equally spaced in the horizontal ($\log(rank)$) dimension. This model is mathematically simpler than the parabolic fractal model, and – given a large enough s – can accurately model departures from the quadratic fit. Each segment is described by a tuple of four values: first rank, last rank, slope in log-log space, and sum of term probabilities within the domain of the segment. From inspection we were confident that ten segments would be sufficient to fit the data well enough. Accordingly we chose $s = 10$. As with h, reducing s would reduce the size and complexity of the model, but the effect would be small.

Table 3 shows all the parameters of our static model. For $h = 10, s = 10$ the size of the model is 53 numeric quantities. That is five or six orders of magnitude smaller than a full language model. Note that when $s = 1, h = 0$ and $w_1 = 0$ the model reduces to a straightforward Zipfian one.

We assign term-ids $1 \ldots h$ to the head, $h+1 \ldots m$ to the middle and $m+1 \ldots |V|$ to the tail, where $m = |V| - w_1$.

3.0.3 Generating term-ids

The starting point for generation of integers representing term-ids is a uniform pseudo-random number generator. Selection of this primary random number generator is critical. It must have a very long period, have sufficient resolution to generate billions of distinct values, and, for in-situ generation, it must require minimal state memory, and respond with speed comparable to the speed of the get-next-term function in a conventional indexer.

The Mersenne Twister [68] algorithm has a reputation for fast generation of high quality pseudorandom numbers. We chose to use the TinyMT64 version[3] to keep the memory footprint small. It has a period of $2^{127} - 1$ which is adequate for our purposes.

A random number R in $0 \ldots 1$ can be used to select head, middle or tail fraction, based on H and w_1/P. If R falls into the head section, R is compared with each of the cumulative H_i values until the id of the correct head term is found and emitted. If R falls into the tail section, we simply emit the next term-id in sequence from a dedicated range of term-ids whose lowest value is determined from $|V|, P,$ and w_1.

For the remaining middle terms we select the appropriate piecewise segment by comparing R agaist the cumulative probabilities in S_i. We then transform R into the range $0 \ldots 1$ and map it to a term-id by treating it as an area under the "Zipf" line represented by the segment and solving for the term rank, i.e. numerical term-id.

The integral with respect to rank r of the continuous Zipf function is

$$\int \frac{r^{\alpha+1}}{\alpha+1} + c$$

for $\alpha \neq -1$. If the first and last ranks of the selected segment are f and l respectively, then the area under the segment is given by:

$$A_{f \ldots l} = \frac{l^{\alpha+1}}{\alpha+1} - \frac{f^{\alpha+1}}{\alpha+1}$$

We want to scale up this area to 1, because R has been transformed into $0 \ldots 1$. The scale factor is thus $F = 1/A_{f \ldots l}$ and the scaled area under the curve from $r = 0$ to $r = f - 1$ is thus $A_{0-(f-1)} = F \times \frac{(f-1)^{\alpha+1}}{\alpha+1}$.

To assign a random number R to a term, we convert it to $R' = R + A_{0 \ldots (f-1)}$ and solve $R' = F \times \frac{r^{\alpha+1}}{\alpha+1}$ for r. We then convert fractional r to the smallest integer t which is larger than r:

$$t = \lceil \exp(\log((R') \times (\alpha + 1))/(\alpha + 1)) \rceil$$

Once a term instance has been identified as "the term at rank r", we need to produce the character string corresponding to that term. Methods for doing that are discussed in Section 5.

4. TERM DEPENDENCE

Our discussion of term dependence assumes that the documents in a corpus form an unordered set. This allows us to avoid consideration of possible term associations across document boundaries.

We consider three types of within-document term association:

[3]`http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/TINYMT/index.html`

1. **Term repetitions.** The tendency of a term T to occur more frequently in certain documents than would be expected if occurrences of T were randomly distributed. For example, the word "economic" may be repeated many times in a financial article.

2. **Phrases.** The tendency of terms T and U (and possibly more terms) to occur in sequence more often than would be expected if T and U were independently distributed. For example, the sequence "economic policy" may occur often in financial or political articles.

3. **Other co-occurrences.** Pairs of terms with related meanings may tend to co-occur (other than as a phrase) more often than would be expected by chance. For example, the words "economic" and "deficit" may tend to co-occur.

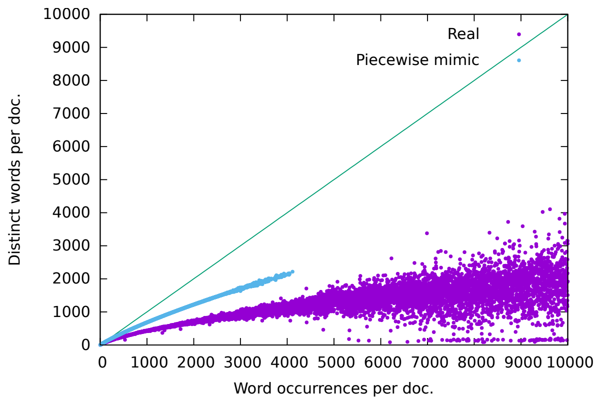


Figure 2: The average number of distinct words per document plotted against the total number of words per document for Indri-WT10g and a term-independent emulation.

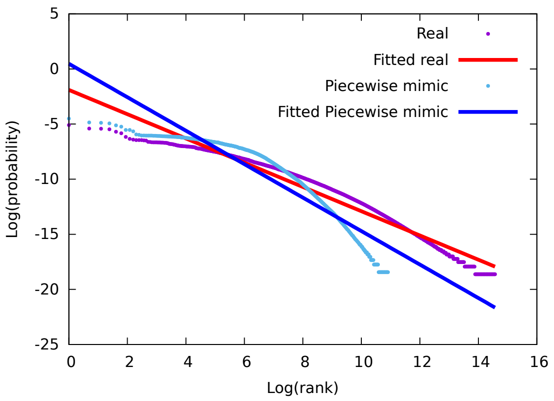


Figure 3: Zipf-like plots of the probability of occurrence of repeated terms for all terms that are repeated for Indri-WT10g and a term-independent emulation.

An important question arises: How to measure the degree to which term associations in a real corpus $\mathcal{C}$ are modeled in an emulated version $\mathcal{E}$? Figure 2 plots the relationship between numbers of distinct and total words per document for Indri-WT10g and an emulated version in which terms were generated independently of each other. As can be seen, the two plots diverge substantially. The only explanation is that, in $\mathcal{C}$, certain words tend to be repeated more often than by chance, reducing the number of distinct terms observed per document. A different view of the same effect can be seen in Figure 3. In it, the distributions of repeated terms are very different between $\mathcal{C}$ and $\mathcal{E}$. Zipf-like plots of bigram frequencies and cooccurrence frequencies for various corpora also show differences between $\mathcal{C}$ and term-independent $\mathcal{E}$.

To achieve more realistic emulation of corpora, we propose a mixture model for term generation in which we generate terms by sampling, in the observed proportions, from a distribution of repeated terms, a distribution of n-grams, a distribution of co-occurring terms not accounted for by repetitions and n-grams, and finally the residual distribution of unigrams (whose probabilities have been adjusted by removing word occurrences accounted for in one of the other distributions).

[[TODO: *A significant problem yet to be solved is how to avoid great increase in model size when representing the pairs of terms which co-occur.*]]

We recommend that the distributions representing repetitions, n-grams and co-occurrences should include only the terms or term pairs whose observed occurrence frequencies would be unlikely to have occurred by chance. In principle, we can achieve this using statistical significance testing.

If we were to repeatedly generate corpora using a term-independent, with-replacement model, we would observe a distribution of occurrence frequencies of the event of interest, e.g. term co-occurrence. Given a desired confidence level of say 95% we could then choose a criterion frequency CF such that 95% of the cases show an occurrence frequency less than CF.

Calculating CF for all pairs of terms presents some challenges. We can approach it either using occurrence probabilities or using probability estimates based on dfs. In the following discussion we consider only the cooccurrence-within-a-document relation. The procedure is essentially the same for the other relations of interest.

4.1 Identifying term associations based on overall occurrence probabilities

What is the probability that independent terms T and U co-occur in a document of length w words? Let p be the probability of occurrence of term T and q be the probability of occurrence of term U.

The probability that T occurs in the document at least once is given by $1-(1-p)^w$. The probability that U occurs in the same document as T is given by approximately

$$(1-(1-p)^w) \times (1-(1-q)^{w-1})$$

. This is an overestimate, because when T occurs more than once the number of possible positions for U is reduced. However, except for cases where p is high or w is low, the accuracy of approximation should be high enough for our purposes.

Note that the above calculation is based on a with-replacement assumption. If a term is observed to occur only once in a corpus of a million term occurrences, its probability of occurrence is estimated as 10^{-6} and that probability is maintained even after we have already sampled it one or more times.

Accurately computing the probability that T and U will co-occur in a collection is complicated by the distribution of document lengths. One would need to sum weighted probabilities for each length. An easy approximation would be to assume that all documents have the same length $\bar{w}$, giving a probability estimate of $(1-(1-p)^{\bar{w}}) \times (1-q)^{\bar{w}-1})$.

Another approximation approach would be to use probabilities based on dfs rather than on overall occurrence frequencies.

[[TODO: *On which probabilities should we base our recommendation?*]]

4.2 Identifying term associations based on probabilities derived from dfs.

If the probability of term T occurring in a doc is p (ie. $DF(T)/N$, where N is the number of documents) and the probability of term U occurring is q, then the probability of them co-occurring in any particular document is $P = p.q$.

Continuing to assume replacement, then the number of co-occurrences observed over an infinite number of trials would follow a binomial distribution. Without replacement the distribution would be hypergeometric, but a binomial approximation is considered good enough. The cumulative binomial we need in order to decide how likely it would be to observe a co-occurrence frequency of k would be incredibly time consuming to compute so, as per common practice, we can in turn approximate the binomial by a normal distribution with mean NP, and variance $NP(1P)$.

In a normal distribution 95% of all observations fall below a Z score of +1.65 (found by looking up a table of standard scores). Therefore our criterion is

$$CF = NP + 1.65 \times \sqrt{NP(1-P)}$$

. We can say with 95% confidence that any pair of terms co-occurring more than this criterion are dependent.

Table 4 gives some example CF values for different values of p_1 and p_2. If for example we found that âĂŸmadâĂŹ with $df = 1000$ and âĂŸmeataxeâĂŹ with $df = 1000$ co-occurred 6 times we would conclude that mad and meataxe exhibited a dependence, because the criterion is 5.65.

4.3 Modeling term burstiness

[[TODO: *Check whether this section is integrated properly with the rest of the discussion.*]] Term burstiness is the tendency for occurrences of a term to cluster within documents. One way to measure a term's burstiness is to relate its overall frequency of occurrence to its observed df.

If occurrences of term T (with frequency of occurrence f) showed no burstiness, we could calculate its expected df on the following basis:

Assuming that each of the N documents has the same length w words, the occurrence probability of T is $q = f/(Nw)$; The probability P that T occurs in a particular document is thus:

$$P = (1-(1-q)^w)$$

We can then the corpus as a sequence of N Bernoulli trials with success probability P, giving us a binomial distribution for the number of successes. The expected number of successes (expected df) is NP. As in Section 4.2 we can approximate the binomial distribution using a normal and use a criterion df of

$$CF = NP - 1.65 \times \sqrt{NP(1-P)}$$

N (no. docs)	df_1	df_2	p_1	p_2	P (joint prob)	NP	$1-P$	Criterion
250000	1	1000	0.000004	0.004	0.000000016	0.004	1	1.654
250000	10	1000	0.00004	0.004	0.00000016	0.04	1	1.69
250000	100	1000	0.0004	0.004	0.0000016	0.4	0.999998	2.049999
250000	1000	1000	0.004	0.004	0.000016	4	0.999984	5.649987
250000	10000	1000	0.04	0.004	0.00016	40	0.99984	41.64987
250000	100000	1000	0.4	0.004	0.0016	400	0.9984	401.6487
250000	1	1	0.000004	0.000004	1.6E-11	0.000004 1	1.650004	
250000	10000	10000	0.04	0.04	0.0016	400	0.9984	401.6487
250000	100000	100000	0.4	0.4	0.16	40000	0.84	40001.51
250000	0	0	0	0	0	0	1	1.65
250000	250000	250000	1	1	1	250000	0	250000
250000	1	250000	0.000004	1	0.000004	1	0.999996	2.649997

Table 4: Criterion values for various pairs of df values in a corpus of 250,000 documents

Any term whose df is less than CF exhibits burstiness. We could perhaps use standard scores to estimate the burstiness:

$$B(T) = (df - NP)/\sqrt{NP(1-P)}$$

[[TODO: *If we new a term was bursty, how would we modify generation?*]]

4.4 Information theoretic approaches

[[TODO: *Do we need to discuss KLD contribution?*]] I'm happy for us to do so, but we would need to:

1. Decide what to use for the base distribution. Is it just $P = p.q$?
2. Justify using the divergence contribution of individual term pairs rather than the divergence of the whole distribution.
3. Justify why using $e \times \log \frac{e}{o}$, where o is the observed "probability" of occurrence and e is the expected probability is better than comparing o to a distribution of e.
4. Work out a principled way of setting a threshold on the Divergence scores

To give a measure of how good our emulation was (with respect to term dependence) we could calculate an overall divergence between co-occurrence probabilities observed in an emulated collection and the $P = p.q$ values for all possible term pairs in the real collection.

Two problems: Our emulation is random, so we'd need to do many emulations and average them. This idea would work best when we were working with the same vocab in $\mathcal{E}$ as in $\mathcal{C}$. If we were using a generated vocab we would have to work out how to map term pairs in $\mathcal{E}$ with pairs in $\mathcal{C}$.

5. GENERATION OF TERM REPRESENTATIONS

Our models of term generation (Sections 3 and 4) output a series of integers R representing the ranks of the terms in a Zipf-style ordering. If we are given a lexicon (for example the lexicon of a corpus being emulated), we can convert each R to a textual representation by simple look-up. If we have no lexicon and no interest in the actual textual representations, we can emit strings such as "T27", representing the 27th most frequenly occurring term.

However, in many simulation scenarios, we will need to generate textual representations of the term-ids output from the term generator. For example, the lengths of terms and the actual character strings which represent them may influence the performance of the vocabulary structure used by an indexer – e.g. the collision rate of a hashing method or the balance of a B-tree. Sigurd et al. [81] modeled the distribution of word lengths in English and Swedish using the formula:

$$f_{exp} = a \times L^b \times c^L$$

where f_{exp} is the expected total frequency of words of length L, and a, b, and c are empirically determined constants, $0 < c < 1$. Sigurd et al. note that this formula corresponds to a Gamma distribution.

Unfortunately, a model of the combined frequency of words of a particular length helps little in modeling how many letters there should be in the representation of a particular term T (identified by its rank R in the term frequency distribution), and helps even less in modeling what those letters should be. We therefore seek a method with the following characteristics:

One-to-one Each term rank R should map to a different simulated term, and no term should map to multiple ranks.

Term length The distribution of term lengths should approximate that expected for a real corpus.

Character frequency Term strings should have an appropriate distribution of characters.

Data structure friendliness Ideally, the synthesized lexicon should stress vocabulary structures, such as trees, tries, or hash tables to a similar extent to that of a real corpus. This is difficult to assess in general, but can be tested with reference to a specific implementation.

Resource usage To support experiments with large corpora, the method should make modest demands on memory and CPU. Cache-friendliness and speed are critical if adopting on-the-fly term generation.

5.1 Methods for generating term representations

As noted above, the simplest way to generate terms for a synthetic collection is to use the lexicon from a real collection. This approach can be used when there are no privacy/confidentiality concerns and is to be the same size as $\mathcal{C}$,

Bucket	Rank Range	Mean length	Standard dev.
0	1–9	2.222	0.629
1	10–99	3.844	1.831
2	100–999	5.600	2.123
3	1000–9999	6.744	2.425
4	10000–99999	7.204	2.586
5	100000–999999	8.265	2.759
6	1000000–9999999	9.415	2.802
7	10000000–99999999	9.736	2.568

Table 5: Means and standard deviations of lengths of terms in ClueWeb12 based on term ranks grouped into log10 buckets. There were a total of approximately 89 million terms.

Length	Cumulative frequency	S
1	26	1
2	702	1
3	18278	1
4	449647	1
5	3082702	5
6	9621775	47
7	19857953	785
8	32701943	16259

Table 6: The first eight rows of the Base26-sparsity multiplier table for ClueWeb12 data.

but is unable to realistically model the lexicon of a scaled-up of terms if the synthetic collection size grows.

More general methods for generating terms include various forms of hashing into a restricted alphabet, as well as more complex methods explored recently by Sutskever et al. [83] and the Markovian models pioneered by Shannon [80].

5.1.1 Word-length model

It is well established [81, 94] that there is a relationship between term length and term frequency: frequently occurring words tend to be shorter. Unfortunately, there is no simple function to map a term's rank to its length.

To generate the term lengths required in some of the term generation methods below, we used an existing corpus as a model. We divided the vocabulary into logarithmically-sized buckets and learned a simple truncated-normal model of term lengths in each bucket. The bucket means and standard deviations for a vocabulary extracted from the ClueWeb12 corpus[4] are shown in Table 5.

5.1.2 Methods for generating term text

Let us consider a range of models for generating the text of words in a simulated corpus. Models vary in complexity, memory requirements, degree of fidelity, amount of training data needed, and speed of generation.

Base26. A fast, very simple method for generating words from an alphabet $\mathcal{A}$ is to represent the rank R as a number in base $|\mathcal{A}|$ using letters of the alphabet as digits. With an English alphabet ($|\mathcal{A}| = 26$) the first 26 ranks are all the one-letter words, then the next 676 are all the two-letter words, and so on. Unfortunately, we cannot expect a realistic distribution of letters or of term lengths. Two variants of Base26 are proposed to partially address these limitations.

Base26-Sparsity. This approach rests on the observation that the sparsity of terms of a particular length increases with the length, as illustrated in Table 6. We estimate the average "skip" S between successive terms of length L, as the ratio of the number of possible terms of length L (i.e. $|\mathcal{A}|^L$) to the number of observed terms of that length. We then calculate $R' = S \times R$ and express R' in base $|\mathcal{A}|$. Provided that S increases monotonically with L, there is no possibility that a term will be generated more than once, and the only additional memory requirement is a table similar to Table 6, with size $O(max\_term\_length)$.

[4]http://www.lemurproject.org/clueweb12.php/

FNV-Base26. Here, we apply the 64-bit Fowler-Noll-Vo hashing algorithm[5] to rank R, and represent FNV(R) as a base-$|\mathcal{A}|$ number. We then use the length model to generate a length L for the term, and select the L least significant base-$|\mathcal{A}|$ "digits".

It is possible that the term generated this way will have been generated before. In this case, we increment L and take a longer term. Note that this requires a large lookup table (of size $O(|V|)$), to record terms have already been generated.

recursive neural network. A better approach is to use observations from some real corpus to guide the construction of synthetic terms. We trained a character-level language model using a Long Short-Term Memory (LSTM) recurrent neural network (RNN).[50, 83] The model is trained on the lexicon file of an existing corpus, predicting one character at a time including an end-of-word character, therefore word length is determined entirely by the RNN. Our RNN has two hidden layers of size 512. During generation, at each step the RNN has a probability distribution (via softmax) over the possible outputs. We sample a character from this distribution, which becomes the input for the next step. It is possible to make the RNN more conservative, by skewing the sample towards high-probability characters, which can make the output look more like correctly-spelled words. However, for our purposes we use the probability distribution as-is, favoring more variation in output. We draw characters one at a time including end-of-word, to generate words.

Markov chains. Inspired by Shannon [80], our final method generates terms from order-k Markov chains. From a sample corpus we learn transition probabilities from every length-k prefix to every letter in our alphabet: for example, for $k = 2$ we learn, for each possible bigram, the probability of any single subsequent letter. We represent the start of a word by k special START symbols, but do not encode word endings. Smoothing is implemented for the higher order models by accessing the order-zero matrix rather than the order-k one, with probability λ. Smoothing increases the range of terms which can be generated and speeds up processing. Results presented here are unsmoothed.

Given a rank R, we draw a length L from the length model described above; then, starting with k START symbols, we randomly walk the Markov chain to generate exactly L letters. If the resulting term was already generated for some other

[5]http://www.isthe.com/chongo/tech/comp/fnv/. We used variant 1a – FNV1A_64.

R, we try again up to an arbitrary maximum of $|\mathcal{A}|$ times. If that fails, we increment L and repeat the process.

Our implementation used a straightforward matrix representation with probabilities represented in double precision. With an alphabet size of 26, the maximum memory required is less than 3GB (for order-5). Memory requirements could be reduced using sparse representations, storing probabilities with less precision, or filtering and smoothing.

5.1.3 Experimental validation

We are concerned with three criteria: the quality of the terms generated by each method, and the memory and runtime cost of the method itself. All our validation experiments involve taking the vocabulary of a real (seed) corpus, generating a synthetic vocabulary of the same size, and comparing the properties of the two. We report results for Clueweb12 [87]; space doesn't permit presentation of broadly similar results for the TREC Associated Press documents and a collection of academic paper titles.

Term quality. For most experimental purposes, we want synthetic terms to "look like" real terms from (e.g.) English web pages. To evaluate our methods on these grounds we use three criteria. First, **term lengths** should be distributed similarly to the seed corpus. Second and third, **character distribution** and **bigram distribution** should match. In each case we measure the Jensen-Shannon divergence [63] between the seed and synthetic distributions: this measures the difference between two multinomial distributions and ranges from 0 (the two distributions are identical) to 1 (they are completely dissimilar).[6]

We must also consider the way generated terms behave in typical components of an IR system. A wide range of vocabulary accumulation structures are reviewed by Heinz et al. [47]. Comparing the suitability of synthetically generated terms against all of these methods is beyond the scope of the present work, but as a sanity check, we consider the rate of hash collisions—that is, how many terms share hashes? This clearly depends crucially on the choice of hash function and table size, but serves to illustrate the sorts of downstream processing needed in a real IR system. The goal is not, of course, to minimise collisions; rather it is to generate terms which behave similarly to those in the seed corpus. We used the FNV-1a hashing method and set the table size to the smallest power of two which was at least 20% larger than the vocabulary size.

Finally, we report the number of generated terms which were also seen in our seed corpus: that is, the number of times each method coincidentally generated a "real" word.

Runtime requirements. Synthetic corpora are likely to be used in performance and scalability testing. If synthetic term generation uses a lot of memory or computation, then it may interfere with the test; on the other hand, if it can be done fast and with minimal memory then it would be possible to embed the generator into the indexer or other component being studied.

We report memory use and CPU time, based on a single indicative run, but note that optimisation has not been a focus and these timings should be interpreted as indicative only. Tests were run single-threaded on a server using Intel Xeon(R) CPU E5-2643 CPUs running at 3.30GHz,and with 64GB of RAM and RAID storage. Reading the seed vocabulary and building all the transition matrices and other structures took a little over 3 minutes elapsed.

We note that the Base26 and Base26-sparsity methods do not rely on shared read-write structures and could easily be multi-threaded. However, methods (all the others) which can potentially generate the same term at different ranks, would require locking of the hashtable used to detect and avoid this.

Results. Results are summarised in Table 7, which shows quality scores for 89,536,745 terms generated with Clueweb12 as seed. The methods which use the length model (Markov-$*$ and FNV-Base26) closely emulate the length distribution of the seed corpus; Base26 performs very poorly while Base26-sparsity may be accurate enough. Perhaps surprisingly, all methods perform fairly well on the unigram distribution. Unsurprisingly the higher order Markov models perform best on the bigram distribution. Choice of method has almost no effect on hash collision rate. The Base26 methods are much faster than the Markov ones, and the higher the Markov order, the slower the method. Much of the slow down is due to the need to generate and test more candidates for uniqueness. The slowest method is more than three hundred times slower than the fastest.

Figure 4 shows the term length distributions of the methods – some of the Markov methods are omitted for clarity. The inherent deficiency of the Base26 method shows clearly in the spike at length 6, while the peak at length 13 for the Base26-sparsity method is believed to be due to measures taken to avoid overflowing the 64 bit integer from which the base 26 string is generated.

5.2 Discussion of term representations

The term lists produced by the higher order Markov methods we implemented show quite good distributions of length, unigram frequency and bigram frequency. Particularly with smoothing enabled, they are capable of generating vocabularies far larger than the one used for training. Subjectively, there is a plausibility to many of the terms they generate: e.g. "bayesiant", "biopestive", "relaxationalco". However, our method cuts off the letter chains at a pre-selected length, resulting in many unnatural word endings. This potential problem could be addressed by avoiding the length model and including an END symbol in the alphabet. To model the dependence of term length on term rank, we propose using multiple START symbols based on the rank buckets in Table 5.

Unfortunately, the Markov methods require far too much memory for use in embedded efficiency experiments. This is because the need for large, randomly accessed data structures causes flushing of CPU caches and consequent serious interference with algorithms being studied. In other off-line generation scenarios they may be preferred.

In embedded applications the Base26-sparsity method seems the best choice. It requires only a small table of skip values (15 in our case), generates better terms than base26, and is capable of generating almost 5 million terms per second, single threaded. However, generating a scaled-up vocabulary would require a model of how the skip table entries change with vocabulary size. Speed of generation can easily be improved by multi-threading and probably by code

[6]Note that we used logarithms with base 2.

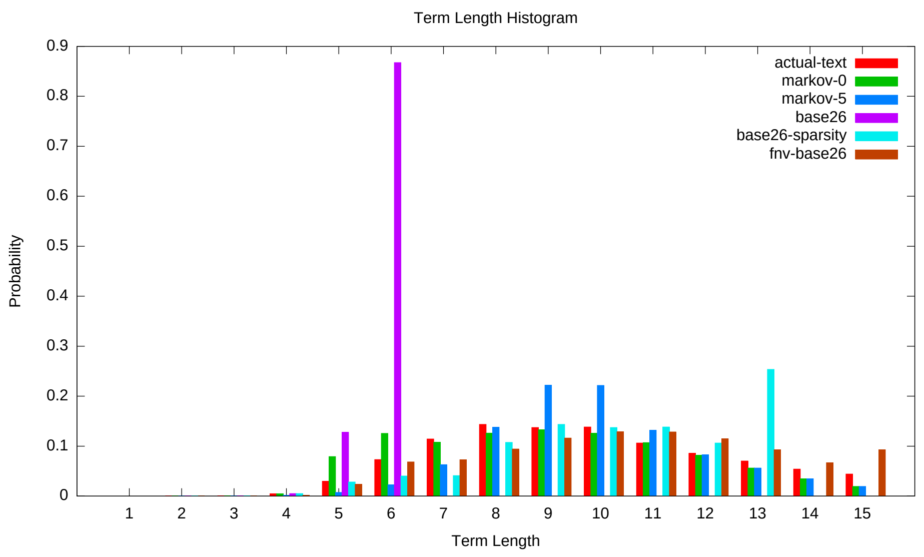


Figure 4: Term length histogram for the ClueWeb12 corpus and various simulated methods.

	JS divergence			Terms	Hash	State	Time/word	
	Lengths	Unigrams	Bigrams	in seed (M)	collisions	memory	μs	Tries
Base26	0.7551	0.0852	0.2388	4.9	1.0013	0	0.12	1.0
Base26-sparsity	0.1095	0.0906	0.2460	1.0	0.9998	16-word array	0.22	1.0
FNV-Base26	0.0192	0.0852	0.2387	0.8	0.9996	HT	0.86	1.1
Markov $k = 0$	0.0200	0.0085	0.0906	4.0	0.9999	HT+$\ll$1MB	3.72	5.8
$k = 1$	0.0176	0.0118	0.0558	5.9	1.0002	HT+$\ll$1MB	5.81	8.6
$k = 2$	0.0182	0.0099	0.0448	6.1	1.0001	HT+<1MB	9.14	12.3
$k = 3$	0.0171	0.0047	0.0230	5.9	0.9996	HT+4MB	15.63	18.1
$k = 4$	0.0229	0.0025	0.0127	6.2	0.9997	HT+105MB	24.17	23.3
$k = 5$	0.0422	0.0019	0.0102	5.9	0.9994	HT+2846MB	43.78	34.2

Table 7: Summary characteristics of the term generation methods. k is the order of the Markov model. Lengths and character distributions are measured with Jensen-Shannon divergence, which ranges from 0 (identical) to 1 (completely dissimilar). Using a FNV-1A hash reduced to 27 bits, 27.038% of terms in the original vocabulary hashed to a slot already occupied by another time. Hash collisions record the result of dividing the corresponding percentage for each method by 27.038%. Memory and runtimes are to generate 89,536,745 terms and are approximate."HT" is the size of the hashtable used to detect duplicates. In our implementation, HT was 2560MB. The number of tries per word is the average number of candidates generated before finding one which had not been used before. The time/word includes all the unsuccessful tries.

optimisation. A disadvantage of the method is that terms sorted by rank are also sorted by length, i.e. all the n-letter terms have lower ranks than any of the terms with $n + 1$ letters. A solution to this might lie in using the length model to choose the length L of a term and then using the skip value for L to compute a term number. An extra counter for each length and a fallback mechanism would be needed to ensure that repeated terms were not generated but total memory requirement would still be only around 30 numbers.

Reproducibility of experiments involving synthetic term generation is an important desideratum. Base26 for an agreed alphabet is inherently reproducible and base26-sparsity requires only the additional sharing of a small table of skip values. Reproducible term generation by the stochastic methods is more challenging. It requires communication of the random seed, and the table of parameters to the length model. A reproducible Markov generator would additionally require communication of the transition matrix.

Sample size	1%	2%	5%	10%	20%	50%	100%
No. samples	7	6	5	4	3	2	1

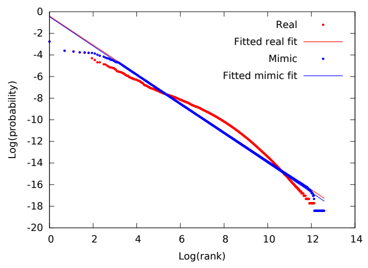
Table 8: Details of sample sizes used in studying collection growth.

6. GENERATING COMPATIBLE QUERIES

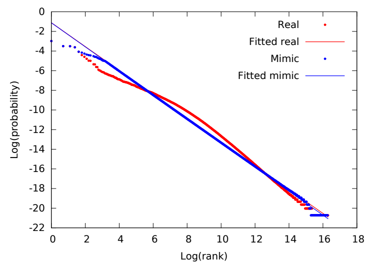
7. MODELS OF CORPUS GROWTH

We would like to provide a valid basis on which studies of algorithm scalability could include synthetic corpora many times larger than a base corpus. In other words, to extract the model parameter values from a base corpus and use a growth model to predict what the values of those parameters would be if the corpus were scaled up by a factor of k.

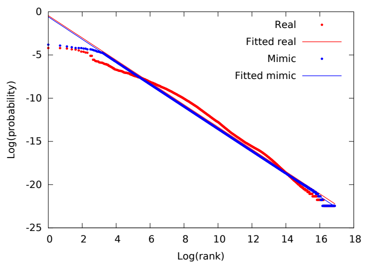
In the absence of a better alternative, we assume that the base corpus is an unbiased sample of an infinite parent



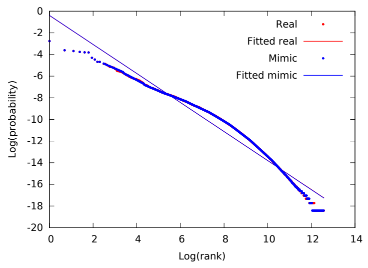
(a) TREC-AP



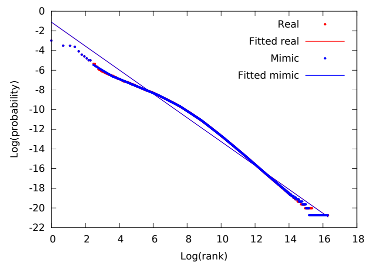
(b) Academic paper titles



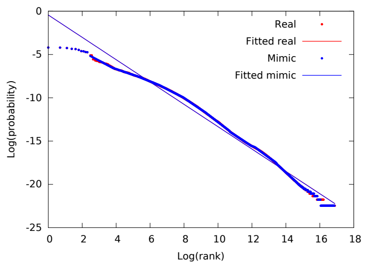
(c) ClueWeb12 titles



(d) TREC-AP

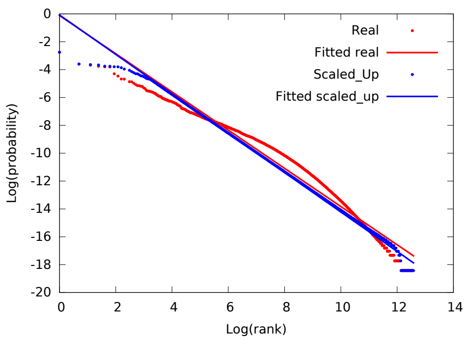


(e) Academic paper titles

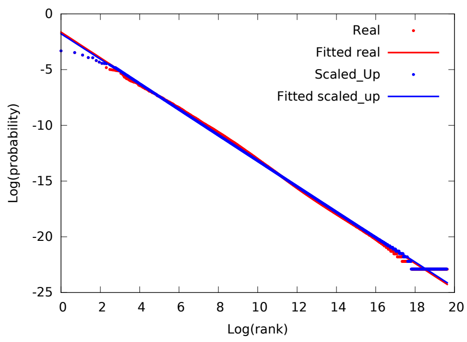


(f) ClueWeb12 titles

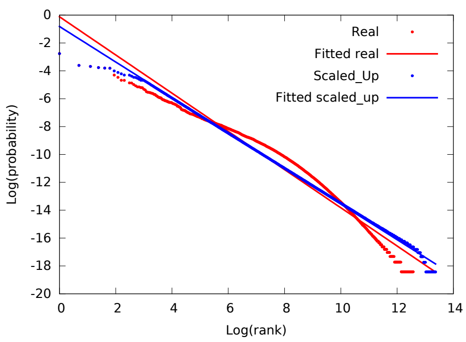
Figure 5: Term probability distributions of base corpora $\mathcal{C}$ and corresponding emulated corpora $\mathcal{E}$. Top row shows linear emulation ($h = 10, s = 1$) while bottom row shows piecewise linear emulation ($h = 10, s = 10$) for the same corpora.



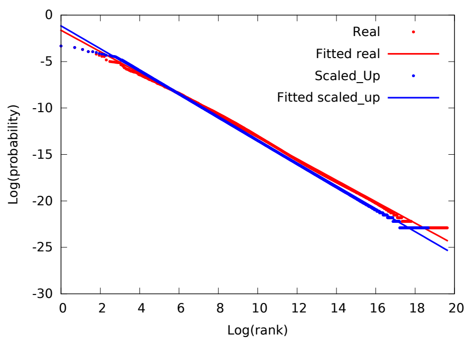
(a) Corpus specific: TREC-AP



(b) Corpus specific: Tweets



(c) Generic: TREC-AP



(d) Generic: Tweets

Figure 6: Emulating a corpus from a 1% sample using growth models. The top row shows term probability distributions for corpus-specific growth models and the bottom row shows those for a generic model. Corpus Tweets was selected to illustrate a very close to linear model while TREC-AP is a corpus with very noticeable non-linearity.

Parameter	Growth function	Additional information
$H_1 \ldots H_h$	constant	
s	$s \times G^{\beta_1}$	β_1: mean = 0.155 st. dev. = 0.0938.
α	$\alpha \times G^{\beta_2}$	β_2: mean = 0.0266 st. dev. = 0.0134.
P	linear	
$\|V\|$	$\|V\| \times G^{\beta_3}$	β_3: mean = 0.620 st. dev. = 0.125.
dl mean	constant	
dl st. dev.	constant	

Table 9: Corpus growth model. The parameters are the same as for the static model with $s = 1, h = 10$ but here we show how they vary as the corpus sample size increases. G is the factor by which the sample size is increased.

corpus. Scaling up the base corpus by a factor of k can be seen as drawing a sample k times as large from the parent.

With this in mind, we derived growth models by randomly sampling records from each base corpus to create samples ranging from 1% to 100% of the original. We drew multiple samples of each size and took the average of the parameter values for each size. The smaller the sample size, the more samples we averaged. We then plotted the change in value of each parameter for each base corpus against sample size and determined, by inspection, the nature of the relationship. These relationships are tabulated in Table 9. Unsurprisingly, the mean and standard deviation of document length don't change with change in sample size. More interestingly, the proportion of term occurrences accounted for by the ten highest frequency terms also remains more or less constant. Unsurprisingly, the number of postings grows linearly with sample size. Three parameters related to the vocabulary size grow in proportion to a fractional power of the scale up factor. Power law growth in vocabulary size is consistent with laws due to Herdan and Heaps.

When simulating the growth of a base corpus, it would be most convenient to avoid the need to derive a corpus-specific growth model from samples. Accordingly, we carried out scale-up experiments using both corpus-specific models and a generic model averaged over the nine corpora. Our growth model was developed using $h = 10, s = 1$. Time has not permitted growth modeling of S, the 4-tuples describing each piecewise segment. This remains for future work.

The scaling-up results are presented in Figure 6. They show that when $\mathcal{E}$ is derived from $\mathcal{C}$ using the generic model the emulation is reasonable, though limited by the accuracy of the linear static model. Results for the Tweets corpus, whose term probability distribution is close to linear, are much better than for TREC-AP. If it proves feasible to derive a good growth model for the piecewise case, we expect the results to be much better for the corpora whose term probability distribution deviates substantially from linear.

Note that the results in Figure 6 represent extrapolation by two orders of magnitude, and that the base for extrapolation in some cases is quite small. We expect that fidelity would be better if the scale-ups were based on larger samples, say 10%.

8. EMULATION EXPERIMENTS

For each of the corpora we extracted values of the model parameters listed in Table 3 and used our generator code to produce a synthetic version.

Figure 5 presents the results of emulation experiments for the piecewise linear model with $h = 10, s = 10$ and the simpler linear model ($h = 10, s = 1$). Piecewise emulation shows almost exact emulation and is clearly superior to the simpler model.

9. SCALING UP EXPERIMENTS

10. PRACTICAL MATTERS

10.1 Preliminary experiences with internal generation

We have modified an indexer to support internal term and document length generation using the three-section, linear and piecewise models described above. We have used it to emulate all nine corpora from Table 1. Preliminary observations suggest that emulation of vocabulary size, term probability distribution, and document length distribution are accurate enough to be useful.

Two attributes of indexing behaviour in this scenario show variations we haven't fully explained. They will be the subject of future study. The effect of emulation on collision rates for the vocabulary hash table in the indexer vary considerably from one corpus to another – sometimes the value for $\mathcal{E}$ is much higher than that for $\mathcal{C}$, sometimes it is much lower. Observed indexing rates (postings / sec.) also fluctuate but are more dependent on hardware than on the corpus. On one machine, the indexing rates for $\mathcal{E}$ tend to be slower than for $\mathcal{C}$; on another machine with more recent CPU/memory architecture the opposite is the case.

Until these anecdotal observations are confirmed and understood, it is too early to conclude whether internal generation is practical.

11. LEAKING OF PRIVACY

In discussion of collection emulation thus far we have focused on achieving fidelity of emulation, and shown that achieving high fidelity requires complex models with many parameters. Another trade-off which we now examine is the increasing loss of privacy occasioned by increasing fidelity of emulation. The most faithful emulation would be an exact copy but this would entail complete loss of privacy. Less faithful emulations also cause privacy leakage but to a lesser degree.

How can we quantify the leakage of privacy occasioned by an emulation method? How can we represent the trade-off between fidelity and privacy leakage in such a way that the custodian of a private corpus can choose an emulation method which achieves useful fidelity while limiting the risk of damage due to leakage of private information.

[[TODO: *Who knows about privacy literature?*]]

12. RELATED WORK

There is a long history of simulation in the field of Information Retrieval. For example, in 1996 Blunt et al. [20] wrote a report for the U.S. Office of Naval Research, proposing a general simulation model of an entire information retrieval system, including personnel and equipment, with the aim of being able to eliminate delays and bottlenecks in the retrieval of important information.

In 1973, Cooper [35] published the earliest system for artificially generating documents of which we are aware. Cooper's system first created an artificial block of text by generating words according to an observed word frequency distribution and assigning them to simulated word classes. It then created a thesaurus by examining word associations in the simulated text. The thesaurus was used in the process of generating both pseudo-documents and pseudo-queries. A pseudo-document consisted of compressed representations such as abstract and index terms. For each representation, a random length was calculated, some starter words were generated and then additional related words were drawn from the thesaurus. As can be appreciated, the simulation model is quite sophisticated but the work was severely limited by the computer hardware of the time. Experiments used a 200 word vocabulary, 150 documents, and a maximum of 20 words per docuement representation.

In 1980, Tague et al. [85] described a system for generating a document-term matrix using a Poisson distribution for document lengths and Zipf-like term generation. Tague et al. review a number of multivariate probability functions for the distribution of index terms over documents, including a model of term dependence due to van Rijsbergen [90]. Again due to the limitations of the era, empirical validation was limited to very small data sets.

Kanungo [53] describes a system for generating artificial corpora of degraded text, suitable for experimentation with OCR systems. He also describes a method for validating degradation models.

We have made mention elsewhere in the text of some of the many studies of term frequency distributions and of the work by Azzopardi et al. on simulated query sets [6] [7]. Other work such as that by Berendsen et al. [16] [17] has developed pseudo test collections for the purpose of providing training data to learning-to-rank systems.

13. DISCUSSION

Our approach to generating a synthetic corpus is essentially a language-model one, though we attempt a more abstract model than a term probability histogram. We attempt to model some types of term dependency, and we are able to model growth in vocabulary.

It would be possible to more explicitly model the process of creating documents and their text content. Considering the Associated Press collection, each article would be generated by an author, possibly commissioned by an editor, and edited by a sub-editor. The author would start with an event and a set of related facts. They would have in mind a target length. They would write a story starting with a heading and then a summary first paragraph, and comprising prose written according to the Associated Press style. The editor might have specified the target length and drawn attention to the event. The sub-editor might change the heading, correct spelling errors, edit the text, and trim the article to length.

Modeling this might start by randomly generating events and pseudo-facts around them. Some attempts have been made by others to automatically generate text from this type of starting point. A good example is the automated sportscaster of Chen, Kim and Mooney [30]. One could generate a sports corpus by repeatedly generating sportscasts – Each time generating facts by picking two cities, choosing animal or other names for their teams, then inventing facts, score lines and player actions.

This more sophisticated approach would generate more believable text but it wouldn't be as useful for the task of enabling efficiency and other measurements on emulated versions of private corpora.

14. CONCLUSIONS, AND FUTURE WORK

[[TODO: *This section came out of the original full-paper submission to SIGIR 2016. It needs to be broadened.*]] We have outlined the potential value of simulated corpora for facilitating reproducible indexing efficiency and scalability studies. From a study of nine real corpora with diverse properties we have formulated a three-section, piecewise linear model accurate enough to allow close emulation of properties which are important determinants of indexing efficiency. With our model it is possible for researchers to distribute terabytes of simulated indexing data by sharing less than a kilobyte of parameters. It is also possible for researchers to engineer corpora with specific sizes and properties to enable systematic study of the effects of varying parameters such as vocabulary size, or the shape of the term probability distribution.

We have developed a term generator whose C code will be made available as open source.[7] It is able to operate internally to an indexer with relatively small impact on CPU cache and runtime, thus avoiding the need to instantiate the simulated corpus on a storage medium. We have used it to emulate all nine of our experimental corpora. However, further work is needed to confirm the usefulness of internal generation. It is hard to see how the assumption of term independence could be removed while retaining the cache-friendly properties necessary for internal generation.

Further work is also needed to determine how to generate term representations (character strings) which match expected length and letter distributions. Improving the accuracy of term representation modeling in a Unicode environment is likely to require additional memory, further reducing the likely feasibility of internal generation.

Assuming that corpus growth can be modeled as a process of additional sampling from an infinite corpus, we developed a growth model which enabled us to simulate datasets much larger than the base corpus from which the static model was extracted. We have shown that corpus properties can be reasonably closely emulated from the properties of 1% samples, even using a generic, rather than corpus-specific, growth model.

Unlike a unigram language model, our growth model allows the vocabulary size to increase as would be expected by Herdan and Heaps. We hope that the ability to simulate believable homogeneous corpora of arbitrary size will facilitate meaningful scalability experiments.

We have shown that our static model overcomes some major limitations of a unigram language model. However, our current model assumes term independence and that limits its usefulness, even in indexing efficiency experimentation, where term burstiness may affect the relative efficiency of indexer structures. Modeling mixtures of unigrams and n-grams is one way forward, and topic modeling is another.

[7] Open source implementations of TinyMT and the Marsaglia polar method for normal random number generation are already available from others.

Extending the simulation model in either of these directions is a challenging problem, particularly in conjunction with the need to generate realistic term representations. Such extensions are only necessary if planned experimentation depends on such properties.

Acknowledgments

We gratefully acknowledge Katja Hoffman, and Walter Sun for their useful suggestions on modelling.

References

[1] Anonymous for blind review.

[2] G. Amati and C. J. Van Rijsbergen. Probabilistic models of information retrieval based on measuring the divergence from randomness. *ACM TOIS*, 20(4):357–389, 2002. URL `http://doi.acm.org/10.1145/582415.582416`.

[3] M. D. Araújo, G. Navarro, and N. Ziviani. Large text searching allowing errors. In *Proc. Fourth South American Conference on Web String Processing, WSP97*, pages 2–20. Carleton University Press, 1997.

[4] N. Asadi, D. Metzler, T. Elsayed, and J. Lin. Pseudo test collections for learning Web search ranking functions. In *Proc. SIGIR 2011*, pages 1073–1082, 2011. URL `http://doi.acm.org/10.1145/2009916.2010058`.

[5] L. Azzopardi. Query side evaluation: An empirical analysis of effectiveness and effort. In *Proc. SIGIR 2009*, pages 556–563, 2009. URL `http://doi.acm.org/10.1145/1571941.1572037`.

[6] L. Azzopardi and M. de Rijke. Automatic construction of known-item finding test beds. In *Proc. SIGIR 2006*, pages 603–604, 2006. URL `http://doi.acm.org/10.1145/1148170.1148276`.

[7] L. Azzopardi, M. de Rijke, and K. Balog. Building simulated queries for known-item topics: An analysis using six European languages. In *Proc. SIGIR 2007*, pages 455–462, 2007. URL `http://doi.acm.org/10.1145/1277741.1277820`.

[8] L. Azzopardi, K. Järvelin, J. Kamps, and M. D. Smucker. Report on the sigir 2010 workshop on simulation of interaction, 2010. `http://sigir.org/files/forum/2010D/sigirwksp/2010d_sigirforum_azzopardi.pdf`.

[9] R. Baeza-Yates. Incremental sampling of query logs. In *Proc. SIGIR 2015*, pages 1093–1096, 2015. URL `http://doi.acm.org/10.1145/2766462.2776780`.

[10] R. Baeza-Yates and G. Navarro. Block-addressing indices for approximate text retrieval. In *Proc. CIKM 1997*, pages 1–8, 1997.

[11] R. Baeza-Yates and G. Navarro. *Modeling Text Databases*, pages 1–25. Springer US, Boston, MA, 2005. URL `\url{http://dx.doi.org/10.1007/0-387-23394-6_1}`.

[12] N. Baker and R. Nance. The use of simulation in studying information storage and retrieval systems. *American Documentation*, November 1968. `notyet`.

[13] F. Baskaya. *Simulating Search Sessions in Interactive Information Retrieval Evaluation*. PhD thesis, School of Information Sciences, University of Tampere, 2014.

[14] F. Baskaya, H. Keskustalo, and K. Järvelin. Modeling behavioral factors in interactive information retrieval. In *Proc. CIKM 2013*, pages 2297–2302, 2013. `http://dx.doi.org/10.1145/2505515.2505660`.

[15] M. Bendersky, D. Metzler, and W. B. Croft. Learning concept importance using a weighted dependence model. In *Proc. WSDM 2010*, pages 31–40. ACM, 2010.

[16] R. Berendsen, M. Tsagkias, M. de Rijke, and E. Meij. Generating pseudo test collections for learning to rank scientific articles. In *Information Access Evaluation. Multilinguality, Multimodality, and Visual Analytics*, volume LNCS 7488 of *Lecture Notes in Computer Science*, pages 42–53. Springer Berlin Heidelberg, 2012. URL `http://dx.doi.org/10.1007/978-3-642-33247-0_6`.

[17] R. Berendsen, M. Tsagkias, W. Weerkamp, and M. de Rijke. Pseudo test collections for training and tuning microblog rankers. In *Proc. SIGIR 2013*, pages 53–62, 2013. URL `http://doi.acm.org/10.1145/2484028.2484063`.

[18] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent Dirichlet allocation. *The Journal of Machine Learning Research*, 3:993–1022, 2003.

[19] C. R. Blunt. An information retrieval system model. HRB-Singer inc.,Report of Contract Nonr. 3818(00) Information Systems Branch, Office of Naval Research, October 1965. `http://www.dtic.mil/dtic/tr/fulltext/u2/623590.pdf`.

[20] C. R. Blunt, R. T. Duquet, and P. T. Luckie. A general model for simulating information storage and retrieval systems. Report of Contract Nonr. 3818(00) Information Systems Branch, Office of Naval Research, April 1966. `http://www.dtic.mil/dtic/tr/fulltext/u2/636435.pdf`.

[21] P. Borlund and P. Ingwersen. The development of a method for the evaluation of interactive information retrieval systems. *Journal of Documentation*, 53(3): 225–250, 1997.

[22] H. Bota, K. Zhou, and J. M. Jose. Playing your cards right: The effect of entity cards on search behaviour and workload. In *Proc. 2016 ACM on Conference on Human Information Interaction and Retrieval (CHIIR)*, pages 131–140, 2016. URL `http://doi.acm.org/10.1145/2854946.2854967`.

[23] C. Bourne and D. Ford. Cost analysis and simulation procedures for evaluation of large information systems. *American Documentation*, April 1964. `notyet`.

[24] G. E. P. Box and M. E. Muller. A note on the generation of random normal deviates. *Ann. Math. Statist.*, 29 (2):610–611, 1958. URL `http://dx.doi.org/10.1214/aoms/1177706645`.

[25] P. F. Brown, J. Cocke, S. A. D. Pietra, V. J. D. Pietra, F. Jelinek, J. D. Lafferty, R. L. Mercer, and P. S. Roossin. A statistical approach to machine translation. *Computational Linguistics*, 16(2):79–85, 1990. URL `http://dl.acm.org/citation.cfm?id=92858.92860`.

[26] P. F. Brown, P. V. deSouza, R. L. Mercer, V. J. D. Pietra, and J. C. Lai. Class-based n-gram models of natural language. *Comput. Linguist.*, 18(4):467–479, 1992. URL `http://dl.acm.org/citation.cfm?id=176313.176316`.

[27] P. F. Brown, S. A. D. Pietra, V. J. D. Pietra, and R. L. Mercer. The mathematics of statistical machine translation: Parameter estimation. *Computational Linguistics*, 19(2):263–311, 1993.

[28] B. Cahoon and K. McKinley. An architecture for distributed information retrieval. In *Proc. SIGIR 1996*, pages 110–118, 1996. `https://ciir-publications.cs.umass.edu/pub/web/getpdf.php?id=70`.

[29] B. Carterette, E. Kanoulas, and E. Yilmaz. Simulating simple user behavior for system effectiveness evaluation. In *Proc. CIKM 2011*, pages 611–620, 2011. URL `http://doi.acm.org/10.1145/2063576.2063668`.

[30] D. L. Chen, J. Kim, and R. J. Mooney. Training a multilingual sportscaster: Using perceptual context to learn language. *Journal of Artificial Intelligence Research*, 37:397–435, 2010. URL `\url{http://www.cs.utexas.edu/users/ai-lab/?chen:jair10}`.

[31] K. W. Church and W. A. Gale. Poisson mixtures. *Natural Language Engineering*, 1:163–190, 1995.

[32] C. Clarke, N. Craswell, and I. Soboroff. Overview of the TREC 2004 Terabyte Track. In *Proc. TREC 2004*. NIST, 2004.

[33] A. Clauset, C. R. Shalizi, and M. E. Newman. Power-law distributions in empirical data. *SIAM review*, 51(4): 661–703, 2009.

[34] C. R. Conger. The simulation and evaluation of information retrieval systems. HRB-Singer inc.,Report of Contract Nonr. 3818(00) Information Systems Branch, Office of Naval Research, April 1965. `http://www.dtic.mil/dtic/tr/fulltext/u2/464619.pdf`.

[35] M. D. Cooper. A simulation model of a retrieval system. *Information Storage and Retrieval*, 9:13–32, 1973. `http://beachmat.berkeley.edu/~mike/Articles/IPMSimulation1973.pdf`.

[36] W. B. Croft, S. Harding, K. Taghva, and J. Borsack. An evaluation of information retrieval accuracy with simulated OCR output. In *Symposium of Document Analysis and Information Retrieval*, 1994. `https://ciir-publications.cs.umass.edu/pub/web/getpdf.php?id=109`.

[37] S. Ding, J. Attenberg, and T. Suel. Scalable techniques for document identifier assignment in inverted indexes. In *Proc. WWW 2010*, pages 311–320, 2010. URL `http://doi.acm.org/10.1145/1772690.1772723`.

[38] L. Egghe. Untangling Herdan's law and Heaps' law: Mathematical and infometric arguments. *JASIST*, 58 (5):702–709, 2007.

[39] T. Field, U. Harder, and P. Harrison. Network traffic behaviour in switched ethernet systems. *Performance Evaluation*, 58(2):243–260, 2004.

[40] W. N. Francis and H. Kucera. *Frequency Analysis of English Usage.* Houghton Mifflin, 1982. Available through Project Muse.

[41] G. Gonnet and R. Baeza-Yates. *Handbook of Algorithms and Data Structures in Pascal and C, 2nd edition.* Addison-Wesley, Wokingham, England, 1991.

[42] M. Greiner, M. Jobmann, and L. Lipsky. The importance of power-tail distributions for modeling queueing systems. *Operations Research*, 47(2):313–326, 1999.

[43] L. Q. Ha, P. Hanna, J. Ming, and F. Smith. Extending ZipfâĂŹs law to n-grams for large corpora. *Artificial Intelligence Review*, 32(1):101–113, 2009.

[44] W. A. Hall. A general simulation model of an on-line telephone directory assistance information retrieval system. In *Proc. the Third Conference on Applications of Simulation*, pages 434–441, 1969. URL `http://dl.acm.org/citation.cfm?id=800164.805229`.

[45] R. Hayes and K. D. Reilly. The effect of response time upon utilization of an information storage and retrieval system – a simulation. In *Proc. Annual meeting of the Operations Research Society of America*, June 1967. `http://files.eric.ed.gov/fulltext/ED033732.pdf`.

[46] H. S. Heaps. *Information retrieval: Computational and theoretical aspects*, chapter 7.5, pages 206–208. Academic Press, 1978.

[47] S. Heinz, J. Zobel, and H. E. Williams. Burst tries: A fast, efficient data structure for string keys. *ACM Transactions on Information Systems*, 20(2):192–223, 2002. URL `http://doi.acm.org/10.1145/506309.506312`.

[48] S. Heinz, J. Zobel, and H. E. Williams. Burst tries: A fast, efficient data structure for string keys. *ACM Transactions on Information Systems (TOIS)*, 20(2): 192–223, 2002.

[49] G. Herdan. *Type-token mathematics: A textbook of mathematical linguistics.* Mouton, The Hague, 1960.

[50] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[51] S. Johnson, A.Cervellino, and J. Wuttke. Libcerf, numeric library for complex error functions version 1.4, 2014. `http://apps.jcns.fz-juelich.de/libcerf`.

[52] K. S. Jones. Collection properties influencing automatic term classification performance. *Information Storage and Retrieval*, 9(9):499–513, 1973.

[53] T. Kanungo. *Document degradation models and a methodology for degradation model validation.* PhD thesis, University of Washington, 1996. `http://www.kanungo.com/pubs/phdthesis.pdf`.

[54] J. G. Kemeny and J. L. Snell. *Finite Markov Chains*. van Nostrand, Princeton NJ, 1960.

[55] H. Keskustalo, K. Järvelin, A. Pirkola, T. Sharma, and M. Lykke. Test collection-based IR evaluation needs extension toward sessions — a case of extremely short queries. In *Proc. AIRS 2009*, volume LNCS 5839, pages 63–74, Berlin, Heidelberg, 2009. Springer-Verlag.

[56] J. Kim and W. B. Croft. Retrieval experiments using pseudo-desktop collections. In *Proc. CIKM 2009*, 2009. URL `\url{https://people.cs.umass.edu/~jykim/papers/cikm1244_kim.pdf}`.

[57] J. Kim and W. B. Croft. Building pseudo-desktop collections. In *Proc. SIGIR 2009 workshop on the future of IR evaluation*, 2009. URL `\url{https://people.cs.umass.edu/~jykim/papers/desktop_collection_sigir_workshop_jykim.pdf}`.

[58] J. Kim and W. B. Croft. Retrieval experiments using pseudo-desktop collections. In *Proc. CIKM 2009*, pages 1297–1306, 2009.

[59] J. Kim and W. B. Croft. Building pseudo-desktop collections. In *Proc. SIGIR 2009 Workshop on the Future of IR Evaluation*, pages 39–40, 2009.

[60] R. Konow, G. Navarro, C. L. Clarke, and A. López-Ortíz. Faster and smaller inverted indices with treaps. In *Proc. SIGIR 2013*, pages 193–202, 2013. URL `http://doi.acm.org/10.1145/2484028.2484088`.

[61] A. Kornai. Zipf's law outside the middle range. In J. Rogers, editor, *Proc. Sixth meeting on the mathematics of language*. University of Central Florida, 1999.

[62] J. Laherrère. Distributions de type "fractal parabolique" dans la nature. *Comptes Rendus de l'Académie des sciences. Série 2a, Sciences de la terre et des planètes.*, 322:535–541, 1996. In French with English summary.

[63] J. Lin. Divergence measures based on the Shannon entropy. *IEEE transactions on Information Theory*, 37 (1):145–151, 1991.

[64] J. Lin. User simulations for evaluating answers to question series. *Information Processing and Management*, 43(3):717–729, 2007.

[65] J. Lin and M. D. Smucker. How do users find things in PubMed? Towards automatic utility evaluation with user simulations. In *Proc. SIGIR 2009*, pages 19–26, Boston, MA, 2009.

[66] G. Marsaglia and T. Bray. A convenient method for generating normal variables. *SIAM Review*, 6:260–264, 1964.

[67] G. Marsaglia and W. W. Tsang. A simple method for generating gamma variables. *ACM Transactions on Mathematical Software*, 26(3):363–372, 2000. URL `http://doi.acm.org/10.1145/358407.358414`.

[68] M. Matsumoto and T. Nishimura. Mersenne Twister: A 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation*, 8(1):3–30, 1998. URL `http://doi.acm.org/10.1145/272991.272995`.

[69] D. Maxwell and L. Azzopardi. Agents, simulated users and humans. an analysis of performance and behaviour. In *Proc. CIKM 2016*, 2016. `http://dx.doi.org/10.1145/2983323.2983805`.

[70] D. Maxwell, L. Azzopardi, K. Järvelin, and H. Keskustalo. Searching and stopping: An analysis of stopping rules and strategies. In *Proc. CIKM 2015*, 2015. `http://dx.doi.org/10.1145/2806416.2806476`.

[71] G. Miller, E. Newman, and E. Friedman. Length-frequency statistics for written English. *Information and Control*, 1:370–389, 1958.

[72] T. P. Minka. Estimating a gamma distribution, 2002. `http://research.microsoft.com/en-us/um/people/minka/papers/minka-gamma.pdf`.

[73] K. J. Mock. *Intelligent Information Filtering via Hybrid Techniques: Hill Climbing, Case-Based Reasoning, Index Patterns, and Genetic Algorithms*. PhD thesis, Computer Science, University of California at Davis, 1996. `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.2.4405&rep=rep1&type=pdf`.

[74] M. E. Newman. Power laws, Pareto distributions and Zipf's law. *Contemporary physics*, 46(5):323–351, 2005.

[75] J. Nichols, J. Mahmud, and C. Drews. Summarizing sporting events using twitter. In *Proc. IUI 2012*, pages 189–198, 2012. URL `http://doi.acm.org/10.1145/2166966.2166999`.

[76] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. Numerical recipes in C: the art of scientific computing. *Cambridge University Press*, 131:243–262, 1992.

[77] R. Reichman. *Getting computers to talk like you and me*. MIT Press, Cambridge, Massachusetts and London, England, 1985. ISBN 0-262-18118-5.

[78] K. D. Reilly. Outline for a simulation study of the california state library network. Technical report, Institute of Library Research. University of California at Los Angeles, July 1968. `http://files.eric.ed.gov/fulltext/ED031280.pdf`.

[79] K. D. Reilly. Digital computer simulation models of library-based information retrieval systems. In *Proc. 3rd Annual Computer Science and Statistics Symposium*. Institute of Library Research. University of California at Los Angeles, January 1969. `http://files.eric.ed.gov/fulltext/ED030451.pdf`.

[80] C. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27:379– 423, 1948.

[81] B. Sigurd, M. Eeg-Olofsson, and J. van der Weijer. Word length, sentence length and frequency – Zipf revisited. *Studia Linguistica*, 58(1):37–52, 2004.

[82] M. D. Smucker. A plan for making information retrieval evaluation synonymous with human performance prediction. In *Proc. SIGIR Workshop on the Future of IR Evaluation*, 2009. `http://mansci.uwaterloo.ca/~msmucker/publications/smucker-SIGIR09-FIRE-workshop.pdf`.

[83] I. Sutskever, J. Martens, and G. E. Hinton. Generating text with recurrent neural networks. In *Proc. ICML-11*, pages 1017–1024, 2011.

[84] L. Sweeney. k-Anonymity: A model for protecting privacy. *International journal of uncertainty, fuzziness and knowledge-based systems*, 10(5):557–570, 2002. doi: 10.1142/S0218488502001648.

[85] J. Tague, M. Nelson, and H. Wu. Problems in the simulation of bibliographic retrieval systems. In *Proc. SIGIR 1980*, pages 236–255, Kent, UK, 1981. Butterworth & Co. ISBN 0-408-10775-8. URL `http://dl.acm.org/citation.cfm?id=636669.636684`.

[86] J. M. Tague and M. J. Nelson. Simulation of user judgments in bibliographic retrieval systems. In *Proc. SIGIR 1981*, pages 66–71, 1981. URL `http://doi.acm.org/10.1145/511754.511764`.

[87] The Lemur Project. The ClueWeb12 Dataset, 2012. URL `www.lemurproject.org/clueweb12.php/`.

[88] J. Thom and J. Zobel. A model for word clustering. *Journal of American Society for Information Science*, 43(9):616–627, 1992.

[89] D. van Leijenhorst and T. van der Weide. A formal derivation of Heaps' Law. *Information Sciences*, 170 (2–4):263–272, 2005.

[90] C. van Rijsbergen. A theoretical basis for the use of co-occurrence data in information retrieval. *J. Doc.*, 33 (2):106–119, 1977.

[91] H. M. Wallach, I. Murray, R. Salakhutdinov, and D. Mimno. Evaluation methods for topic models. In *Proc. ICML 2009*, pages 1105–1112. ACM, 2009.

[92] R. White, J. Jose, K. van Rijsbergen, and I. Ruthven. A simulated study of implicit feedback models. In *Proc. ECIR 2004*, pages 311–326. Springer, LNCS 2997, 2004.

[93] H. E. Williams and J. Zobel. Searchable words on the Web. *International Journal on Digital Libraries*, 5(2): 99–105, 2005.

[94] G. K. Zipf. *The psycho-biology of language.* Houghton, Mifflin, Oxford, 1935.

[95] G. K. Zipf. *Human behavior and the principle of least effort.* Addison-Wesley, 1949.

[96] J. Zobel and A. Moffat. Inverted files for text search engines. *ACM Computing Surveys*, 38(2), 2006. URL `http://doi.acm.org/10.1145/1132956.1132959`.