Reducing E-Discovery Cost by Filtering Included Emails

Tsuen-Wan "Johnny" Ngan
Core Research Group
Symantec Research Labs
900 Corporate Pointe
Culver City, CA 90230

Abstract

As businesses become more reliance on information technology, electronic information is often produced as vital evidence during civil litigation. The process of discovering electronic information as evidence is getting increasingly expensive as the volume of data explodes. This surging demand calls for a solution to reduce the cost associated with the discovery process. In this paper, we propose filtering included emails as a means to reduce review data volume, and describe efficient algorithms to identify those emails. Experiments show that this can reduce the number of emails to be reviewed by 20% in corporate email corpse.

1 Introduction

Email has become an indispensable part of communication for enterprises of any size, both for within the enterprise and with outside parties. It has long been considered as official means of communication, replacing postal mails and facsimiles as the most common form. Nevertheless, its speed and convenience come with a hefty price of management and storage overhead.

The problem is highlighted with the introduction of legislation such as the Health Insurance Portability and Accountability Act (HIPAA) in 1996 and the Sarbanes-Oxley Act (SOx) in 2002, both of which have significantly increased the importance of managing and storing all information within enterprise. Email, in particular, has become one of the most important content types that needs to be retained. This is further complicated by the fact that retention rules could vary based on users, locations, file types, and whatnot. In fact, for larger enterprises, the number of items to be retained could easily be multi-billions.

The management of this sheer volume of data becomes a formidable task for large enterprises. Off-the-shelf storage products and solutions often perform some basic processing on the data. For instance, many have the feature of whole-item de-duplication (or single-instance storage), where the same item retrieved from different locations and received by different users is stored only once. Other common processing is usually based on email headers, typically allowing users to perform boolean queries to select and filter emails. However, little is done on email content, other than perhaps building an inverted index to expedite searching. Analysis of content data is therefore done manually and inefficiently, particularly under the recent surging demand for *e-discovery*.

E-discovery, or electronic discovery, refers to discovery of evidence from electronic data in civil litigation. Unlike traditional discovery from paper information, the uniqueness of electronic information stems from its intangible form, volume, transience, and persistence. Furthermore, electronic information is usually accompanied by metadata, which is rarely present in paper information.

E-discovery is often time-consuming and expensive. Litigations sometimes require humongous amount of emails to be reviewed to extract evidence. This process is usually done manually by attorneys that charge by the hours or data size. A survey conducted in 2006 [Fulbright, 2006] on senior corporate counsels showed that companies are facing growing numbers of lawsuits, and fewer are escaping them, even as arbitration numbers also grow. The cost of this active environment for dispute is very high: Companies with $1 billion or more in annual revenues spent an average of $31.5 million on all their legal matters, and 40% of the study samples had at least one lawsuit filed against them in which $20 million or more was at issue. While there are several products in the market aiming to assist attorneys and their customers to facilitate and streamline this process [Symantec, 2008, EMC, 2008, ZANTAZ, 2008], little is done on analyzing email content to reduce the amount of emails to be reviewed. Machine-learning techniques like classification and clustering [Huang et al., 2004, Surendran et al., 2005] are often unsuitable, as enterprises cannot risk missing crucial evi-

dence from false negatives of these techniques.

This paper positions itself in a practical perspective towards alleviating the burden on attorneys to manually reviewing every single email. The key insight is that during e-discovery, any email that is completely included by another email in the repository can be safely ignored, without any risk of losing evidence. To that end, we propose an efficient paragraph-based algorithm for statically and dynamically finding all email inclusions in an repository. We also experimented the algorithms against the Enron corpus and an mailing list trace.

The rest of this paper is organized as follows. Section 2 describes the process of e-discovery and explains the advantages for filtering included emails. Section 3 depicts our paragraph-based comparison algorithm to quickly find inclusions. The algorithm is tested on two data sources in Section 4. Finally, Section 5 covers related work and Section 6 concludes.

2 The E-discovery Process

Due to the potential of litigations, companies are required to preserve electronic information. In fact, e-discovery is the subject of amendments to the Federal Rules of Civil Procedure [Supreme Court, 2006], effective since December 2006. The amendment concerns itself with a company's duty to preserve and produce electronically stored information in the face of litigation or pending litigation. Parties involved in civil court proceedings must provide a list of all electronically stored information that might be relevant to the case. Involved parties must also discuss the forms in which this information should be produced, and the party who requests the information may specify the form or forms in which electronically stored information is to be produced.

Since e-discovery is relatively new, there is no well-established standards and procedures for enterprises to follow. E-discovery typically involves a number of phases, and various industry bodies have proposed standards as to how each phase should be satisfied. One of those industry bodies, called the Electronic Discovery Reference Model project (EDRM) [EDRM Group, 2008], is backed by many technology and service vendors in the discovery market. This model, laying down e-discovery standards and guidelines, contains the following steps:

1. **Identification.** The scope, breadth, and depth of electronically stored information that might be pursued during discovery is first assessed and determined.
2. **Preservation.** The stored information is protected against destruction and alternations.
3. **Collection.** The stored information is gathered from various sources, including backup tapes, drivers, and portable storage devices.
4. **Processing.** The overall set of data is reduced by setting aside duplications and data that are irrelevant due to their type, origin, or date.
5. **Review.** The collected information is evaluated for relevance.
6. **Analysis.** The discovery materials are analyzed to determine relevant summary information.
7. **Production.** The information is delivered to various recipients, for use of various systems, and on various media.
8. **Presentation.** The information is finally presented at depositions, hearings, and trials.

As the volume of data decreases down the steps, the relevance remains in the volume rises.

Out of all steps, step 5 (review) and 6 (analysis) are usually the most expensive ones, since a large amount of data still needs to be manually processed by attorneys. Moreover, each review and analysis may be specific to the current litigation. This means that separate review and analysis steps are needed for each new litigation. Hence, any optimization should be done on minimizing the volume of data involved in these steps.

Consider an email discussion thread with possibly many emails. When an email in the thread is replied or forwarded, it is common practice for many to quote the content of the original email fully in verbatim. From a reviewing perspective, it suffices to review only the last email in the thread, since it already contains the text of all emails in the thread; all other emails in the thread can be filtered without any loss of evidence.

Of course, a response email may or may not quote its parent email in entirety, and an email thread may contain many branches that resemble a tree rather than a line. It is therefore necessary to compare email content to determine which emails can be safely filtered.

There are two advantages of this inclusion comparisons. First and foremost, it reduces the number of emails to be manually reviewed. Second, it helps to group emails in the same discussion thread together, even when other heuristics [Yeh & Harnly, 2006] fail. A reviewer can then read these related emails together, allowing the review process to be more efficient.

3 Finding Email Inclusions

This section describes our proposed algorithms for finding inclusions.

3.1 Defining Inclusions

Conceptually, an email is "included" when its content is completely present in other emails. However, in practice, it is tricky to define inclusion exactly for e-discovery. In one extreme, inclusion may require a verbatim copy of an entire email; in the other extreme, an email can be considered included when all its sentences are eventually reviewed. While cases may be made for either case, these definitions are too extreme and are not particularly useful for e-discovery.

Recall that the purpose of finding inclusion is to allow reviewers to skip certain emails, with the confidence that in doing so they would not miss out any information. To that end, classification must be done conservatively. A sentence or paragraph that is present in multiple emails may carry different meanings based on its location. For instance, it would be incorrect to conclude that a reply of "Yes" can be ignored if there are some unrelated emails containing a "Yes." Thus, we require an email to be completely contained by another email before it can be filtered.

Another decision is on the smallest unit for comparisons. Since we are considering the meaning of emails, the minimal basic unit with a complete meaning should be sentences. However, notice that paragraphs are usually unmodified after quotation. By expanding the basic unit to paragraphs, it can greatly reduce the number of items for comparisons without affecting much on accuracy.

Of course, this paragraph-based definition may result in false negatives (failure to find an inclusion). For example, if a paragraph is broken into two in a reply, it would end up with two new paragraphs instead of the original one. However, for the purpose of reducing volume of review data, these false negatives would only prevent the filtering of some emails, but would not remove potential evidence. We consider this as an acceptable tradeoff for faster searching.

3.2 Paragraph-Based Inclusion Check

We now present a high-level design of our paragraph-based inclusion check. Each email is represented by a set of digests, where each digest is computed from a paragraph in the email.

Fundamentally, inclusion check can be done using two fundamental building blocks:

- *Subset test*: Given email E, determine if E is contained by any member of a set of emails.
- *Superset test*: Given email E, determine if E contains any members of a set of emails.

With a large set of emails, the superset test is much more expensive than the subset test. Assume an inverted index is built that maps paragraphs to emails that contain them. For the subset test, it suffices to use *any* paragraph in E to pick out candidate emails. However, for the superset test, it is necessary to consider *all* paragraphs in E.

```
-----Original Message-----
From:  Allen, Phillip K.
Sent:  Friday, December 07, 2001 5:14 AM
To:  Dunton, Heather
Subject:  RE: West Position
```

```
> At 04:18 PM 10/1/2001 -0500, you wrote:
```

Figure 1: Two examples of software-generated quotation text in email content.

Because of the difference in performance, the two tests enable different use cases. For an existing email archiving system, all emails are present at the very beginning. To find inclusions, it suffices to iterate over all emails and perform the subset test on each one. However, for a live email archiving system, when a new email arrives, it is necessary to check inclusion in both directions.[1] Therefore, a tool that is tightly integrated to a live email archiving system would need to run both tests on each new email. This paper solves the latter, more computational-intensive problem.

3.3 Implementation Details

The rest of this section describes details of our proposed algorithm. The algorithm consists of preprocessing individual emails and then finding inclusions between them.

3.3.1 Preprocessing Individual Emails

The preprocessing converts each email into a set of digests. It involves three steps: (1) Removing extraneous text; (2) dividing email into paragraphs; and (3) hashing each paragraph. Since some of these steps depend on the language and its usage practice, for simplicity we assume all emails are in English.

The first step is to remove extraneous text. Extraneous text is defined as parts in an email that are not inputed by human users. This includes email headers and quotation text in content that are automatically generated by email client software. Examples of the latter are shown in Figure 1.

The remaining email content is then divided into paragraphs by locating paragraph separators. For text emails, a paragraph separator could be a line with no alphanumeric characters or a paragraph separator character defined in Unicode. For html emails, it could be a paragraph tag

[1] Note that while new emails are unlikely to be included by existing emails in normal circumstances, emails may not arrive in chronological order. For example, they may be collected from different sources at different time.

(<p/>) or consecutive line break tags (
).

The last step is to compute a digest for each paragraph. Since we are only interested in text and not its format, only alphanumeric characters are extracted from each paragraph and hashed. This ensures that automatic alternations like insertion of quoting symbols (e.g., >) and line breaks would not affect the generated digests. A universal hash function [Carter & Wegman, 1979] like UMAC [Black et al., 1999] is preferred due to the use of Bloom filters below, but any commonly used ones like MD5 or SHA-1 would be sufficient.

3.3.2 Finding Inclusions

Since each email is already converted to a set of digests, the problem is reduced to running both subset test and superset test efficiently.

If we perform the superset test by iterating over all paragraphs and check all emails containing them, the process would be very slow, as some paragraphs appear in a large number of emails (see Section 4.1.4), and every time when a new email containing a popular paragraph arrives, it becomes necessary to iterate over all emails that contain that popular paragraph. To expedite the process, we need to handle popular paragraphs separately.

Popular Paragraphs

The algorithm divides paragraphs into two sets: *popular* and *unpopular*, based on a threshold of occurrences. With this partitioning of paragraphs, we can categorize emails into two sets: (1) Emails with only popular paragraphs (called $\mathcal{P}$); and (2) Emails with at least one unpopular paragraph (called $\mathcal{Q}$). Two separated inverted indices are built, mapping paragraphs to emails in $\mathcal{P}$ and $\mathcal{Q}$.

Note that for a paragraph to be popular, it must appear in a large number of emails, many of which may be unrelated to each other. Examples of these include signatures and greetings gratitude. Since emails usually contain other pieces of useful information, very few emails should fall into $\mathcal{P}$.

To search for inclusion candidates, the algorithm iterates over all paragraphs in the email, but handles popular and unpopular paragraphs differently. For each unpopular paragraph, since they are only present in $\mathcal{Q}$, all emails that contain it in $\mathcal{Q}$ are added to the candidate set. However, for each popular paragraph, it suffices to add only emails in $\mathcal{P}$. This is because any email in $\mathcal{Q}$ would also contain unpopular paragraph, so if it is an inclusion, it would be selected when we consider that unpopular paragraph.

The algorithm avoids the worst case, namely, iterate over all emails that contain a popular paragraph whenever it processes an email with that popular paragraph.

Once the set of candidate emails are selected, it is still necessary to determine if there are inclusions. The set of candidate emails can still be very large, since any email that share any common paragraph is in the set. Thus, we use Bloom filters to filter out most false positives.

Bloom Filter

Bloom filter [Bloom, 1970] is a space-efficient probabilistic data structure used to test set membership. It is basically a bit vector initialized to all 0. Each element added to the filter will set one or more bits to 1. A set membership test is done by checking if all the bits corresponding to the given element are set to 1. Since there are more elements than the number of bits available, Bloom filters may have false positives but not false negatives. We extend this data structure to test for subset relation between emails.

Note that while most emails have only a few paragraphs, some emails contain a large number of paragraphs (see Section 4.1.2). To avoid many emails from having their Bloom filters entirely set to 1 and rendering the Bloom filters ineffective, we correspond each paragraph to only one bit.

Once we have the Bloom filters for two emails, checking for inclusion can be done very quickly. If email A contains all the paragraphs in email B, all the bits that are set in B's Bloom filter would also be set in that of A. Hence, the bitwise AND of the two Bloom filters should be the same as B's Bloom filter.

Of course, similar to using Bloom filters for set membership test, this inclusion test can still have false positives. Thus, after this test is passed, it is imperative to compare the sets of digests from the two emails to conclude with certainty that they indeed have an inclusion relation.

Blacklisting Uninteresting Paragraphs

A possible optimization is to ignore popular paragraphs that do not contribute much in content. Examples of those paragraphs include company names, greetings, gratitude, signatures, advertisements (from webmails or eletronic devices), boilerplates, and subscription information for mailing lists. While technically their presence can affect inclusion decisions, these paragraphs can be ignored without affecting the quality of filtering.

4 Experiments

In this section, we examine the proposed algorithm against two datasets. We also investigate the effectiveness of removing included emails and the performance improvement from using Bloom filters.

4.1 Datasets

To understand the performance of the algorithm under different input, we evaluated it using two datasets: the *Enron*

email corpus and a *mailing list dataset* we created. This subsection describes these two datasets and performs some comparisons and analyses.

The Enron email corpus contains data from about 150 users, mostly senior management of the now-defunct company Enron Corporation [Klimt & Yang, 2004]. It was originally made available by the Federal Energy Regulatory Commission during its investigation. The data does not include attachments, and some emails have been deleted "as part of a redaction effort due to requests from affected employees." This serves as a corporate email trace.

The mailing lists dataset is collected from four mailing lists at December 2007, consisting of mails from their respective archive websites: The Cygwin project mailing list archive [Cygwin, 2007], general Gentoo user support and discussion mailing list [Gentoo, 2007], MySQL general discussion [MySQL, 2007], and qmail mailing list [qmail, 2007]. This serves as a general email discussion trace.

A basic summary of the two datasets is shown in Table 1. Since both datasets contain duplications, a de-duplication process is done by computing MD5 digests on the alphanumeric characters of each email. The bottom half of the table shows the data after de-duplication. Note that less than half of the emails remained after de-duplication in the Enron corpus, but only less than 5% are removed in the mailing list dataset. This is mostly because the Enron corpus includes the "Sent" folder of each user, and some emails are sent to multiple users within the corpus.

Even though the two datasets are very different, they turn out to possess similar properties. Below we compare some of these properties.

4.1.1 Paragraph Length

Figure 2 shows the length distribution of distinct paragraphs, measured in the number of alphanumeric characters. It shows that both datasets have similar fraction of shorter paragraphs. However, the mailing list dataset has more middle-sized paragraphs, whereas the Enron corpus has more longer paragraphs.

The former is probably because all the mailing lists are related to programming and software and thus the mails often contain moderately long code snippets. The latter can be attributed to two reasons: (1) Some emails in the corpus were converted from other formats to text, but during the conversion paragraph margins were not properly retained; and (2) some emails contain very long XML documents that are stored in one large paragraph.

4.1.2 Paragraphs Per Email

Figure 3 shows the number of paragraphs per email. In both datasets, most of the mails contain a small number of 10

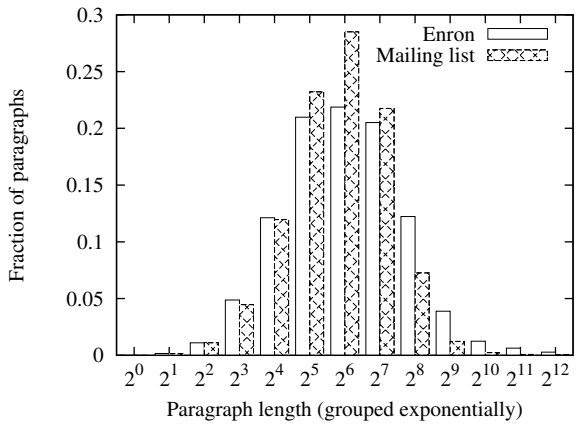

Figure 2: The length distribution of paragraphs, measured in alphanumeric characters, grouped exponentially. The two data sets have similar fraction of shorter paragraphs, but the mailing list dataset has more middle-sized paragraphs while the Enron corpus has more longer paragraphs.

paragraphs. The main difference between the two datasets is that the Enron corpus has more shorter emails, likely due to short personal notes and sending attachments between colleagues. On the other hand, postings on mailing lists usually contain at least a few paragraphs to describe or answer the questions, or to include code snippets.

4.1.3 Correlation Between Size and Likelihood of Inclusion

The size of an email, measured in either alphanumeric characters or number of paragraphs, has a negative correlation to the probability that it is included by another email. Figure 4 and 5 shows that most included emails are relatively short (less than 100 alphanumeric characters or 10 paragraphs).

4.1.4 Paragraph Occurrences

Figure 6 shows the number of occurrences of distinct paragraphs over each entire repository. Note that for both datasets, the frequency of appearance goes down logarithmically even as each group is growing exponentially in size. Also, both curves have very long tails: The most common paragraph appeared in the Enron corpse 23,052 times, whereas that for the mailing list dataset is 54,316.

Figure 7 shows the data in cumulative distribution curves. Note that most paragraphs appeared only for a few times. For example, for both datasets, less than 1% of the paragraphs appeared more than 10 times. Thus, a small popular threshold can be used to separate a small fraction of popular paragraphs from unpopular paragraphs.

Since many paragraphs appear many times across many emails, a possible application of paragraph-based analysis

Table 1: The two datasets and their detailed properties. The bottom half shows the data after removing duplications.

	Enron	Mailing list
Data type	Corporate emails	Mailing list discussions
Number of emails	517,431	486,869
Total size (excluding headers)	961MB	680MB
Average size per email	1,858 bytes	1,397 bytes
Average number of paragraphs per email	8.32	9.62
Number of distinct paragraphs	1,020,319	1,916,744
Number of emails after removing duplications	248,517	464,766
Fraction of remaining emails	48.03%	95.46%
Total size of remaining emails	473MB	656MB

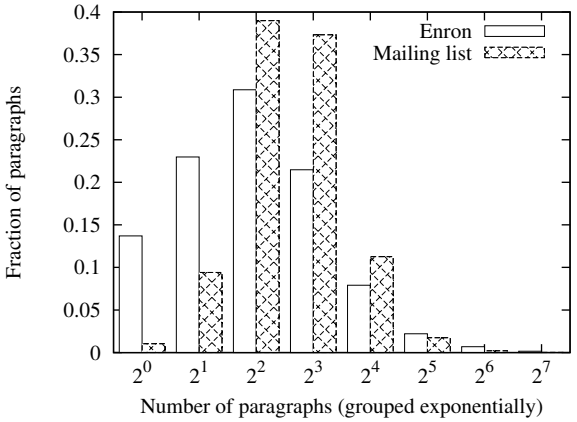

Figure 3: The distribution of the number of paragraphs per email, grouped exponentially. The main The Enron corpus has more shorter emails, likely due to short personal notes and sending attachments between colleagues.

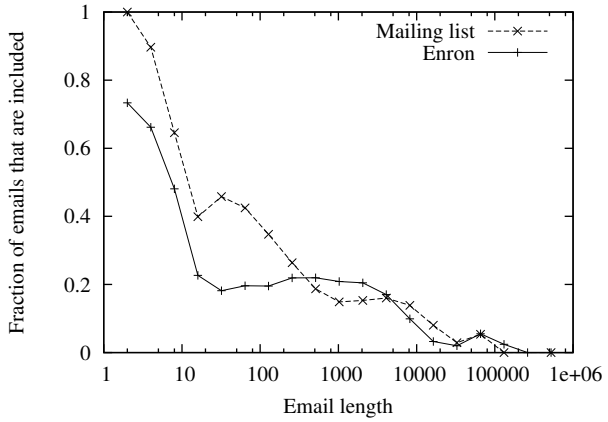

Figure 4: The correlation between number of paragraphs and the probability that it is included.

Table 2: Fraction of included emails that can be removed before reviewing.

	Enron	Mailing list
No. of nonidentical items	248,517	464,766
Included items	50,787	87,544
Fraction of included items	20.4%	18.8%

Table 3: Average running speed of the prototype algorithm. These numbers do not include the disk I/O time for reading the emails from disk.

	Enron	Mailing list
Without blacklist	2.01 MB/s	2.48 MB/s
With blacklist	2.64 MB/s	3.99 MB/s

is to perform de-duplication at the paragraph level. Specifically, paragraphs that are identical can be stored only once, with duplicated paragraphs pointing to a common copy. We performed an informal measurement on this feature. Even after whole-item de-duplication and without compression, this feature can reduce storage space by an additional 22–28%.

4.2 Effectiveness of Filtering Included Emails

The goal of finding included emails is to reduce the volume of data to be reviewed. We first determine how well this idea may work in practice by measure the amount of emails that are completely included and can thus be ignored during e-discovery. Table 2 shows the fraction of included emails. In both datasets, even after de-duplication, this amounts to around 20% of the emails, all of which can be removed without affecting the accuracy of reviewing.

4.3 Algorithm Performance

We built a prototype of the algorithm to get a handle of its performance. The prototype is ran on a Windows XP desktop machine with a Pentium-4 3.4 GHz CPU. The prototype uses Berkeley DB version 4.6.19 as a large hash table for building inverted indices. The program and the database

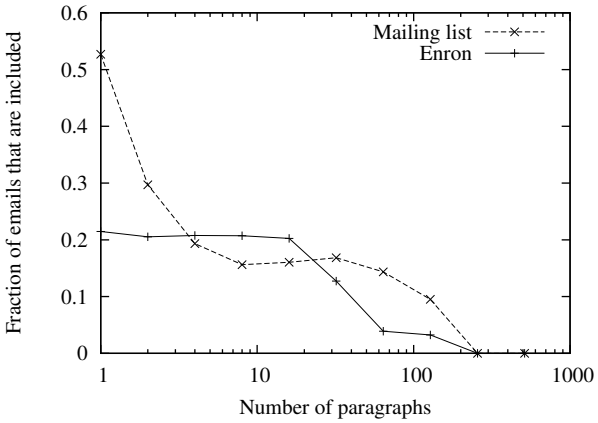

Figure 5: The correlation between email length and the probability that it is included.

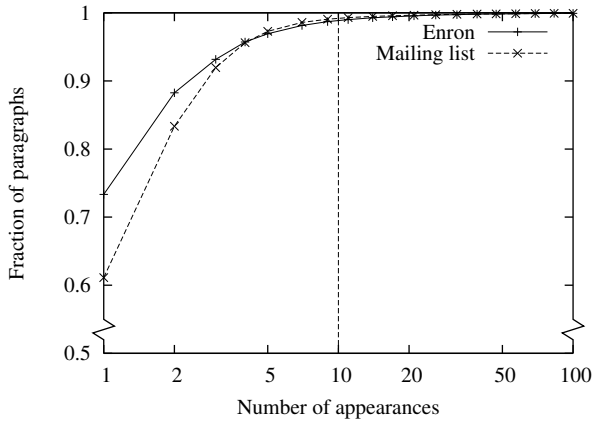

Figure 7: The cumulative distribution of occurrences of paragraphs. The dotted line shows a cutoff threshold of 10.

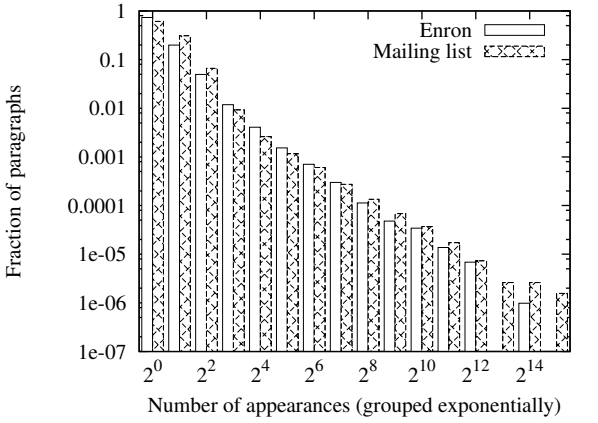

Figure 6: The distribution of occurrences of paragraphs, grouped exponentially. For both datasets, the frequency goes down logarithmically even as each group is expanded exponentially.

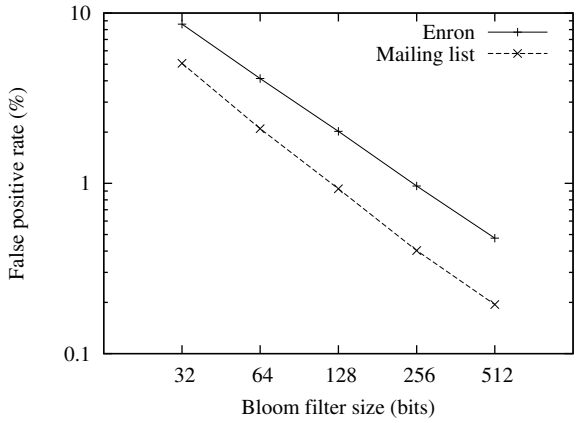

Figure 8: Effectiveness of Bloom filter for subset comparisons at different filter sizes. Similar to the original use of Bloom filters, doubling the size of the filter reduces the false positives by a factor.

is given enough memory so that it is always running entirely in memory. The program is ran with and without a blacklist, which is manually built to include 41 popular but unimportant paragraphs.

Table 3 shows the running speed of the prototype averaged over the entire datasets. Note that these numbers do not including the time spent on reading emails from disk. The table shows that the algorithm can process multi-MB per second, fast enough to run in a real-time system. Also, when employing a blacklist to ignore unimportant paragraphs, the algorithm can run 31–61% faster.

Of course, the actual running speed for each email depends on the size of the indices and whether the paragraphs in the email are common. However, the prototype seems to scale reasonably well over the datasets tested, as the last 1% of the emails are only 40–50% slower than the first 1%.

The rest of this section shows two more micro-benchmarks on two parameters used.

4.3.1 Bloom Filter

Bloom filter is used as a front layer to filter out email candidates that are not inclusions. In our prototype that used a Bloom filter size of 16 bytes (128 bits), its false positive rate is measured to 2% for the Enron corpus and 0.9% for the mailing list trace. This shows that a Bloom filter can be very effective even at a size much smaller than an average email. It also allows us to filter out most candidates without even looking at any of their digests.

While the effectiveness of Bloom filter is transparent, it is unclear how large it should be, as it is difficult to analytically derive the false positive rate and optimal Bloom filter size. The traditional analy-

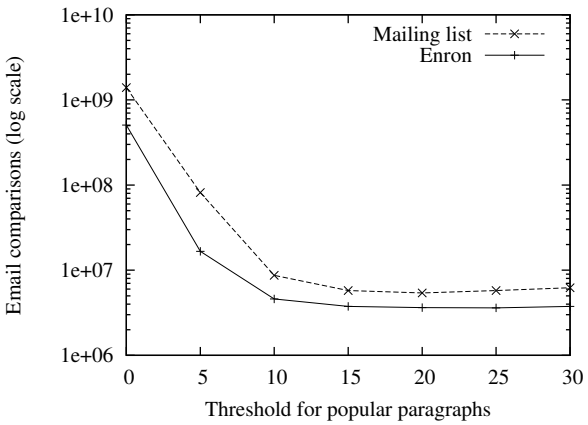

Figure 9: Filtering cost with different popular thresholds. A zero threshold represents the case when there is no separate popular paragraph set. Observe that a reasonable threshold reduces the filtering cost by a factor of 200.

sis [Broder & Mitzenmacher, 2002] is for set membership test and is not applicable to our subset test. Moreover, as unrelated emails often share common paragraphs, the conditional probabilities in the analysis cannot be assumed to be independent. Instead, we study the tradeoff by experimentally measuring the false positive rate at different Bloom filter sizes.

The result is shown in Figure 8. Clearly, for both datasets, the false positive rate is reduced by roughly the same factor every time the Bloom filter size is doubled. This result is similar to using Bloom filters for set membership test.

4.3.2 Popularity Threshold

Lastly, Figure 9 shows the cost of finding inclusions using different popular thresholds. The cost is measured in terms of the number of full set comparisons (i.e., the number of times the Bloom filter test is passed) needed. For both datasets, the number of comparisons reduced by two orders of magnitude when a popular threshold is used and is increased to 10, but remained roughly at the same level when the threshold is further increased.

This shows the effectiveness of partitioning paragraphs into popular and unpopular. When a popular threshold is used, popular paragraphs are usually not used to find inclusion candidates. This eliminates the worst case of iterating thousands of emails just because they share a common popular paragraph. However, when the threshold is too small, e.g., 5, too many emails are classified into $\mathcal{P}$ (emails with only popular paragraphs). Since popular paragraphs still cause emails in $\mathcal{P}$ to be iterated, this would require a large number of comparisons.

5 Related Work

An alternative to finding inclusion is to group emails into email threads, perhaps through header information or other heuristics [Yeh & Harnly, 2006]. While this would provide reviewers the same benefit of reviewing the same discussion at a time, further comparisons are still needed to filter out included emails. Moreover, these heuristic-based algorithms may not be accurate, and are ineffective for emails that are inherently not in the same thread, for example when text is copied from one email and pasted into another.

Email signatures do not contain useful information, but they may appear in many unrelated emails and cause slowdown. A simple optimization is to remove these signatures before running our algorithms using a signature extraction algorithm [Carvalho & Cohen, 2004].

Fingerprinting techniques, e.g., Rabin fingerprinting [Rabin, 1981], coupled with fingerprint selection algorithms like Winnowing [Schleimer et al., 2003], can similarly generate digests for finding emails that share common portions. The same comparison algorithm can then be applied for finding inclusions. Our paragraph-based generation is more suitable for this application since it exploits the fact that paragraphs are usually unmodified in responses. Our algorithm guarantees to generate a digest for each paragraph, no matter how short it is, and only generates one digest for very long paragraph, which reduces the number of comparisons but increases accuracy.

There are other machine-learning research that work on email content. These includes automatic clustering [Huang et al., 2004] and classification [Surendran et al., 2005] and other techniques for filtering spams [Joachims, 1999, Sahami et al., 1998].

6 Conclusions

In this paper, we used inclusions to filter emails for e-discovery. We proposed a paragraph-based comparison to find inclusions, and algorithms for performing comparisons efficiently. Our experiment showed a 20% reduction of number of emails to be reviewed in an e-discovery process.

While our algorithms for finding inclusions are independent of the language used by the emails, the preprocessing needs to be aware of the language, as it has to know how to divide email content into paragraphs and remove formatting characters. Also, to remove extraneous text, it may require the preprocessor to recognize quotation text in the language used. Extending the preprocessor to other languages is left as future work.

References

[Black et al., 1999] Black, J., Halevi, S., Krawczyk, H., Krovetz, T., & Rogaway, P. (1999). UMAC: Fast and secure message authentication. *Advances in Cryptology — CRYPTO '99*. Santa Barbara, CA.

[Bloom, 1970] Bloom, B. H. (1970). Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, *13*, 422–426.

[Broder & Mitzenmacher, 2002] Broder, A., & Mitzenmacher, M. (2002). Network applications of Bloom filters: A survey. *Proceedings of the 40th Annual Allerton Conference on Communication, Control, and Computing*. Urbana, IL.

[Carter & Wegman, 1979] Carter, J. L., & Wegman, M. N. (1979). Universal classes of hash functions. *Journal of Computer and System Sciences*, *18*, 143–154.

[Carvalho & Cohen, 2004] Carvalho, V. R., & Cohen, W. W. (2004). Learning to extract signature and reply lines from email. *Proceedings of the 1st Conference on Email and Anti-Spam (CEAS)*. Mountain View, CA.

[Cygwin, 2007] Cygwin (2007). The Cygwin project mailing list archives. Retrieved from `http://cygwin.com/ml/cygwin/`.

[EDRM Group, 2008] EDRM Group (2008). The electronic discovery reference model (EDRM) projects. `http://edrm.net/`.

[EMC, 2008] EMC (2008). EMC EmailXtender family. `http://www.emc.com/products/family/email-xtender-family.htm`.

[Fulbright, 2006] Fulbright (2006). *Third annual litigation trends survey findings*. Fulbright & Jaworski L.L.P.

[Gentoo, 2007] Gentoo (2007). General Gentoo user support and discussion mailing list. Retrieved from `http://archives.gentoo.org/gentoo-user/`.

[Huang et al., 2004] Huang, Y., Govindaraju, D., Mitchell, T., de Carvalho, V. R., & Cohen, W. (2004). Inferring ongoing activities of workstation users by clustering email. *Proceedings of the 1st Conference on Email and Anti-Spam (CEAS)*. Mountain View, CA.

[Joachims, 1999] Joachims, T. (1999). Transductive inference for text classification using support vector machines. *Proceedings of 16th International Conference on Machine Learning*. Bled, Slovenia.

[Klimt & Yang, 2004] Klimt, B., & Yang, Y. (2004). Introducing the Enron corpus. *Proceedings of the 1st Conference on Email and Anti-Spam (CEAS)*. Mountain View, CA. Data available at `http://www.cs.cmu.edu/~enron/`.

[MySQL, 2007] MySQL (2007). MySQL general discussion. Retrieved from `http://lists.mysql.com/mysql/`.

[qmail, 2007] qmail (2007). qmail mailing list. Retrieved from `http://www.ornl.gov/lists/mailing-lists/qmail/`.

[Rabin, 1981] Rabin, M. O. (1981). *Fingerprinting by random polynomials* (Technical Report TR-15-81). Center for Research in Computing Technology, Harvard University.

[Sahami et al., 1998] Sahami, M., Dumais, S., Heckerman, D., & Horvitz, E. (1998). A Bayesian approach to filtering junk e-mail. *Proceedings of the AAAI Workshop on Learning for Text Categorization*. Madison, Wisconsin.

[Schleimer et al., 2003] Schleimer, S., Wilkerson, D. S., & Aiken, A. (2003). Winnowing: Local algorithms for document fingerprinting. *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*. San Diego, CA.

[Supreme Court, 2006] Supreme Court (2006). *Amendments to the Federal rules of civil procedure*.

[Surendran et al., 2005] Surendran, A. C., Platt, J. C., & Renshaw, E. (2005). Automatic discovery of personal topics to organize email. *Proceedings of the 2nd Conference on Email and Anti-Spam (CEAS)*. Palo Alto, CA.

[Symantec, 2008] Symantec (2008). Symantec Enterprise Vault. `http://www.symantec.com/ev/`.

[Yeh & Harnly, 2006] Yeh, J.-Y., & Harnly, A. (2006). Email thread reassembly using similarity matching. *Proceedings of the 3rd Conference on Email and Anti-Spam (CEAS)*. Mountain View, CA.

[ZANTAZ, 2008] ZANTAZ (2008). ZANTAZ Enterprise Archive Solution. `http://www.zantaz.com/products/eas.php`.