Implementing a Similar Messages Search Function in an IMAP Mail Server

Arto Teräs <arto.teras@hip.fi>
Helsinki Institute of Physics Technology Program

September 30, 2002

Abstract

This paper presents a prototype implementation of a mail server featuring a content-based message search function. The user can choose an interesting email and ask the server to retrieve similar ones. Communication between the client and the server is implemented via an additional command in the IMAP protocol and requires minimal changes to the client software.

Contents

1 Introduction

People who receive a lot of email usually have large archives of messages distributed in several folders. Sometimes when receiving a new message the user vaguely remembers that he has saved some information about the same topic earlier but cannot remember the folder and/or author. In this case it would be useful to have the computer automatically retrieve similar documents to the current one in hand.

This document describes a "search for similar messages" extension to an IMAP mail server. All the analysis functionality is implemented at the server so the modifications required to the client software are small. Currently the system accepts only another email message as the search key, but it would be easy to add a function where the user could fill in keywords and the system would then fetch the best matching messages.

This research project was related to the Datagrid project in the Helsinki Institute of Physics Technology Program [1]. It is also submitted as the project work of the course `T-61.195 Information Science, special assignment I` at the Helsinki University of Technology. The supervisors of the project have been M.Sc. Matti Heikkurinen from Helsinki Institute of Physics and D.Sc.(Tech.) Mikko Kurimo from Helsinki University of Technology, Laboratory of Computer and Information Science.

2 Choosing the Approach

2.1 Architecture

When extending a widely used and standardized system such as Internet email, it is important that the new features don't break any existing functionality. Because many people are already very accustomed to their favorite email client it would also be nice that the new features could be easily added to several clients.

Currently there are mainly three methods to read email. The messages can either reside on the same computer as the client software[1] or on a server, which further can use either the POP [2] or IMAP [3] protocol[2]. The IMAP protocol, optionally secured by SSL, has become the most common method in use today. It allows keeping the messages on a server in a folder hierarchy chosen by the user, but many users also copy their email archives on their local workstation. Finally, the format of the messages themselves can be either strictly plain text [4] or use MIME extensions [5].

Three different system architectures were considered.

1. Both the classification and the messages reside on a server. The client program uses a simple command to ask for similar messages from the server.
2. The classification is done on a server but the similarity data is added as extra headers (so called X-headers) to the email message. In this case the messages can be left either at the server or downloaded at the client but the client is responsible for interpreting the data in the extra headers.
3. Both the classification functionality and the messages are at the client.

The second and third option don't require any extensions to protocols in use today. Unfortunately they both have practical disadvantages. If all the classification functionality is at the client (option 3), it easily becomes closely tied to one particular client software and porting it to others would be a big task. If the classification information is in the headers (option 2) the system becomes either rather unflexible (e.g. a fixed list of message numbers in one extra header) or a considerable amount of interpretation functionality needs to be implemented at the client.

Finally option 1 was chosen. The advantage of this approach is that few modifications are needed in the client software. Communication between the client and the server was decided to be implemented as an extension to the IMAP protocol as allowed in section 6.5.1 in the IMAP specification [3].

[1]The messages can be either in standard mailbox or maildir format or a proprietary format of the client software.

[2]Email services with a web interface can still be considered as another category. Some of them use IMAP internally, some have their own databases and protocols, but in all cases the implementation is completely at the server and the client software (web browser) does not need to be modified when new functionality is added.

2.2 Software Packages

To avoid license fees, to make development easier and to encourage further development it was decided to use existing open source [6] components as much as possible. The software packages chosen for the project are introduced below.

DBMAIL [7] is a group of programs that enable storing and retrieving mail messages from a SQL database (currently MySQL and PostgreSQL support implemented). In particular, it can be used as an IMAP server. There are other IMAP servers which are more widely used but DBMAIL was chosen because the database backend allows implementing complex features efficiently. The development community also seems to be quite active. DBMAIL is written in C and licensed under the GNU General Public License [8].

Bow [9] is a library for writing statistical text analysis, language modeling and information retrieval programs. It can among other things be used for tokenizing text files, building a vocabulary and calculating document and word vectors based on the data. The approach is statistical — Bow does not have syntactical or semantical analysis functionality. Bow is written in C and licensed under the GNU Library General Public License [10] (version 2).

Mutt [11] is a small but powerful text-based email client. It was chosen for this project mainly because the source code is easy to understand and extend. Mutt is written in C and licensed under the GNU General Public License [8].

Two tools for constructing self-organizing maps (SOM) [12] were used when analyzing the results and evaluating optimization possibilities.

SOM Toolbox [13] is a function package for Matlab [14] implementing SOM algorithm. It also contains a rich set of visualization capabilities and various helper functions. SOM Toolbox is written in the Matlab language and licensed under the GNU General Public License[3] [8].

SOM_PAK [15] is a library which implements the self-organizing map (SOM) algorithm and various helper functions. Self organizing maps can be useful for automatic classification and visualization of various types of data, including text. SOM_PAK is written in C and the license allows free use for research purposes only.

In addition to the tools chosen, several other interesting projects and software packages were found during the work. Some of them are briefly described here because they may be useful for future research related to emails and language analysis.

MDV [16] is a distributed, XML-based knowledge management platform developed at the Helsinki Institute of Physics. Initially it was planned to use MDV as the basis for the system and integrate the new feature in the email component of MDV. Later it was decided not to use it for the first prototype because MDV itself is currently in heavy development. The functionality may still be added to MDV later. MDV is written in Java and licensed under the BSD License [17] (without the advertisement clause).

CMU-Cambridge Statistical Language Modeling toolkit [18] is a suite of UNIX software tools to facilitate the construction and testing of statistical language models. It was one candidate to be used in this project but at the end Bow [9] seemed more suitable. The CMU-Cambridge toolkit is written in C and the license allows free use for research purposes only.

Gate [19] is an XML-based framework for processing human languages. It seems to have a lot of functionality and good documentation, but the emphasis is on syntactic analysis while in this project the approach is statistical. Gate is written in Java and licensed under the GNU Library General Public License [10].

[3]The toolbox requires Matlab so one must have a Matlab license to be able to use it.

Ifile [20] is a mail filtering system based on the Naive Bayes algorithm. It forms classes based on the contents of messages the user has already saved in different folders and suggests the best matching class for a new message. Ifile can also be useful when trying to manage large email archives, but the approach is different than the one described in this document. Ifile is written in C and licensed under the GNU General Public License [8].

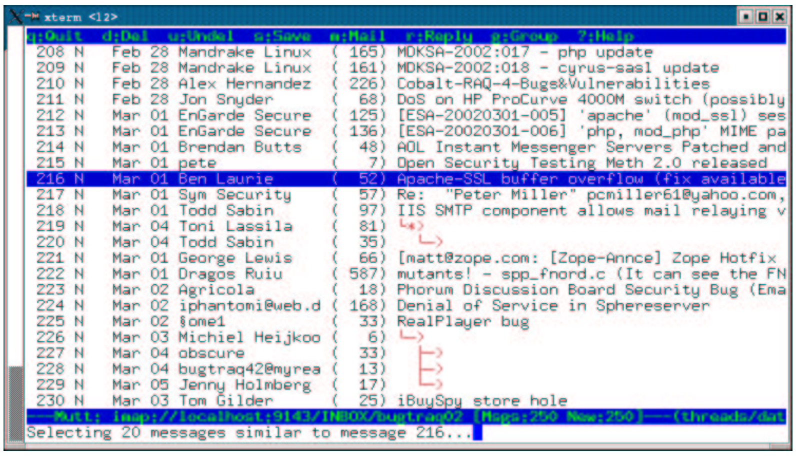
OpenNLP [21] is an organizational center for open source projects related to natural language processing. It is also a project to develop Java interfaces and XML schemas for standardizing natural language software components and data. It consists of several packages which may be useful for various language processing tasks. Most of them are written in Java.

WEBSOM [22] is a research project to organize and visualize various documents using a self-organizing map. It has been used to form topic maps of news bulletins, Usenet newsgroup articles and patent abstracts. WEBSOM has been developed in the Information Science Laboratory at Helsinki University of Technology and it is partly based on SOM_PAK. The source code is not publicly available.

3 User Interface

3.1 Current Implementation

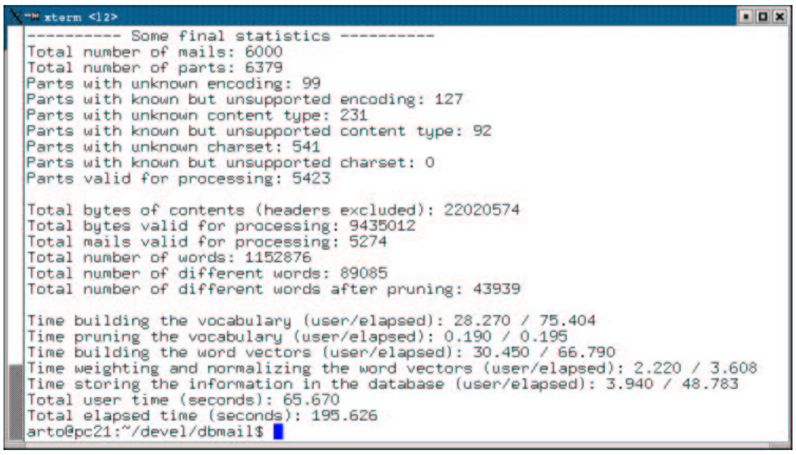
In the current implementation the interface is very simple (figure 1). The user can navigate in his mail folders as usual and when he finds an interesting message he can search for similar ones by pressing the `S` key. The system asks how many messages the user would like to see before sending the request to the server. The resulting list looks like a regular folder which is organized so that the most similar messages appear at the top of the list.



```
q:Quit  d:Del  u:Undel  s:Save  m:Mail  r:Reply  g:Group  ?:Help
 208 N   Feb 28 Mandrake Linux  ( 165) MDKSA-2002:017 - php update
 209 N   Feb 28 Mandrake Linux  ( 161) MDKSA-2002:018 - cyrus-sasl update
 210 N   Feb 28 Alex Hernandez  ( 226) Cobalt-RAQ-4-Bugs&Vulnerabilities
 211 N   Feb 28 Jon Snyder      (  68) DoS on HP ProCurve 4000M switch (possibly
 212 N   Mar 01 EnGarde Secure  ( 125) [ESA-20020301-005] 'apache' (mod_ssl) ses
 213 N   Mar 01 EnGarde Secure  ( 136) [ESA-20020301-006] 'php, mod_php' MIME pa
 214 N   Mar 01 Brendan Butts   (  48) AOL Instant Messenger Servers Patched and
 215 N   Mar 01 pete            (   7) Open Security Testing Meth 2.0 released
 216 N   Mar 01 Ben Laurie      (  52) Apache-SSL buffer overflow (fix available
 217 N   Mar 01 Sym Security    (  57) Re:  "Peter Miller" pcmiller61@yahoo.com,
 218 N   Mar 01 Todd Sabin      (  97) IIS SMTP component allows mail relaying v
 219 N   Mar 04 Toni Lassila    (  81) └*>
 220 N   Mar 04 Todd Sabin      (  35)   └─>
 221 N   Mar 01 George Lewis    (  66) [matt@zope.com: [Zope-Annce] Zope Hotfix
 222 N   Mar 01 Dragos Ruiu     ( 587) mutants! - spp_fnord.c (It can see the FN
 223 N   Mar 02 Agricola        (  18) Phorum Discussion Board Security Bug (Ema
 224 N   Mar 02 iphantomi@web.d ( 168) Denial of Service in Sphereserver
 225 N   Mar 02 §ome1           (  33) RealPlayer bug
 226 N   Mar 03 Michiel Heijkoo (   6) └─>
 227 N   Mar 04 obscure         (  33)   ├─>
 228 N   Mar 04 bugtraq42@myrea (  13)   ├─>
 229 N   Mar 05 Jenny Holmberg  (  17)   └─>
 230 N   Mar 03 Tom Gilder      (  25) iBuySpy store hole
---Mutt: imap://localhost:9143/INBOX/bugtraq02 [Msgs:250 New:250]---(threads/dat
Selecting 20 messages similar to message 216...
```

Figure 1: Choosing an interesting message and searching for similar ones.

The interface for the administrator is a command-line program called `dbmail-organize` (figure 2) which calculates the classification data. The program accepts a few parameters to control exclusion of very rare and too common words from the vocabulary. It is naturally possible to run the program automatically in desired intervals (using cron) but currently the system is static: when a new mail arrives a word vector is not calculated for it automatically. This would not be difficult to add, but a full readaptation of vocabulary and weights should still be done sometimes. It would also be easy to add more options and methods, some are already implemented in the Bow library. This could be a good topic for further research.



```
---------- Some final statistics ----------
Total number of mails: 6000
Total number of parts: 6379
Parts with unknown encoding: 99
Parts with known but unsupported encoding: 127
Parts with unknown content type: 231
Parts with known but unsupported content type: 92
Parts with unknown charset: 541
Parts with known but unsupported charset: 0
Parts valid for processing: 5423

Total bytes of contents (headers excluded): 22020574
Total bytes valid for processing: 9435012
Total mails valid for processing: 5274
Total number of words: 1152876
Total number of different words: 89085
Total number of different words after pruning: 43939

Time building the vocabulary (user/elapsed): 28.270 / 75.404
Time pruning the vocabulary (user/elapsed): 0.190 / 0.195
Time building the word vectors (user/elapsed): 30.450 / 66.790
Time weighting and normalizing the word vectors (user/elapsed): 2.220 / 3.608
Time storing the information in the database (user/elapsed): 3.940 / 48.783
Total user time (seconds): 65.670
Total elapsed time (seconds): 195.626
arto@pc21:~/devel/dbmail$
```

Figure 2: Statistics screen after running the `dbmail-organize` program

3.2 Ideas for Improvement

There are many possible improvements which could make the system better. It would be nice to highlight the matching words in the messages and provide the user a possibility to emphasize certain words or topic in a subsequent search. Placing the messages on a visual map such as the one in WEBSOM [22] could help to navigate in the archive. Time scale could also be considered: if there is a group of similar messages closely related to each other in time, the user would probably want to see that as a group. When an interesting message is found as the result of a search it should be possible to retrieve the whole thread to take a closer look at the discussion.

It would be easy and probably useful to add a function where the user would supply the keywords instead of choosing an existing message as the key of the search. The IMAP protocol already has a search command but it is limited to the currently selected mailbox and does not use any weighting to retrieve the best matching messages.

The user should have some options to filter the messages on the similarity list, for example asking only for messages from a certain author or excluding one author. He could also have more control for the criteria of similarity. However, all these should be considered carefully as too many options are only confusing to most people. The system should be able to retrieve relevant documents with minimal effort from the user.

4 Implementation Details

4.1 Extending the IMAP Protocol

IMAP protocol specification [3] provides a uniform way to add experimental commands which are not part of the standard protocol. Any command prefixed with an `X` is considered experimental and the arguments for such commands are user defined. The client can query the server about special features using the `CAPABILITY` command.

The search for similar messages is provided by a very simple command `XSIMILAR` which takes two arguments: number of the message which is used as the search key and another number which tells how many similar messages the client wants to get. The message number is relative to the current mailbox but the search is performed across all the folders of the user.

The response for the `XSIMILAR` command is similar to commands opening a mailbox (or a folder), e.g. the `SELECT` command. The idea is to return the list of similar messages to the client exactly as it were a normal

IMAP mailbox even though it is not a physical folder on the server. The server internally handles the list as a virtual mailbox so that the client may normally fetch attributes and the contents of the messages by using the relative numbers in the mailbox.

This approach requires only minor changes in the client software. However, it must be admitted that the `XSIMILAR` command does not fit seamlessly in the IMAP protocol state machine model. Normally the client is responsible for closing a mailbox before opening a new one — here the old mailbox is implicitly closed because a new one is returned. Also, as the mailbox is virtual it just silently disappears when the client closes it and opens another one.

The main problem is that that the messages may perfectly well be from several different real mailboxes. Therefore one cannot just return a list of message numbers inside the currently selected mailbox. It would be possible to return a list of message unique identifier numbers, but then it would be necessary to define new special commands or redefine some exisiting commands like `UID FETCH` to retrieve messages globally and not just from the current mailbox. This would break the standard.

Virtual mailboxes also pose semantic problems. If the user would like to edit message flags or delete a message, should it be done even though the message physically resides in another mailbox? A simple solution used in this implementation is to deny modifications by returning the similar messages list as read-only.

It might be useful to extend the `XSIMILAR` syntax to provide more detailed control and additional features. For example, currently there is no information about the criteria which messages are considered similar to each other. The command could also accept a list of messages as the source and provide a possibility to limit the result list with arguments identical to the `SEARCH` command.

4.2 Combining the Software Components

The main task was to extend DBMAIL so that it would extract the vocabulary and necessary statistics from the emails and calculate a word vector for each message. Most of the needed functionality was already present in the Bow library but putting it all together required a significant amount of glue code. Some examples:

- Storing and retrieving the data structures used by Bow to/from the DBMAIL database (MySQL [23] was used as the backend in this project)
- Maintaining a mapping between Bow document index and the email message identifiers in DBMAIL
- Retrieving the messages from database and combining all the parts of each message as one document suitable for the Bow lexer
- Filtering out binary attachments, recognizing various character sets and encodings and performing necessary conversions
- Adding statistics and debugging functions

Sometimes the “way of doing things” in Bow and DBMAIL did not match very well, but having the source code available helped in many cases. It also made possible to fix a couple of bugs which were encountered during development. Gathering various statistics about messages during parsing and conversions proved also to be a good idea — it was not only useful in the analysis at the end but also during development.

Organizing emails on the server is handled by a new program called `dbmail-organize`, which the administrator must run from the command line. It fetches the messages from the DBMAIL database, calls Bow functions to form the vocabulary and word vectors and then stores them in the database. The program offers a few options to exclude the most common and rarest words from the vocabulary. Naturally each user account has its own vocabulary and word vector data.

The IMAP server is implemented by `dbmail-imapd` which was extended to understand the new `XSIMILAR` command. When the client searches for similar messages for the first time, the server reads the vocabulary and word vector info to be cached in the memory for the duration of the connection. The actual search is currently implemented in a brute force manner so that it makes a pass through all the word vectors to find the best matching ones.

Adding the new command to Mutt was simple and took only about one working day. The modifications should be minor to almost any mail client, but because they include adding a new command to the IMAP protocol the changes may go deep in the library code. For example many popular Web based mail clients call an IMAP API provided by PHP. Therefore one would first have to modify PHP, compile the web server (e.g. Apache) with the modified PHP and then add the new function call to the actual mail client.

The SOM Toolbox and SOM_PAK were used only for analysis purposes and it didn't require any modifications.

The versions of software were primarily the newest available official releases: Bow version 1.0, Mutt version 1.4, SOM Toolbox version 2.0beta, SOM_PAK version 3.1 and MySQL version 3.23. For DBMAIL a direct copy of the cvs source tree on July 19 was chosen because it contained some important database-related optimizations which were not present in the latest version available on the DBMAIL home page (1.0rc3).

To get more detailed information about the changes see the source code and the comments there.

5 Analyzing the Textual Data

Natural language processing has been a topic of intensive research for decades. Many syntactic analysis algorithms and other tools have been produced but automatic processing of language semantics is a very hard problem and an universal solution doesn't exist. A good overview of the current state of the field and descriptions of the most common algorithms can be found in the book Speech and Language Processing by Jurafsky and Martin [24].

5.1 Feature Extraction

Choosing a good set of features is essential in any classification task. The best features for email messages would be concepts and keywords of the topics which the messages are about. Unfortunately this would require either a lot of time-consuming manual interaction with the user or complicated analysis of the semantics of the text.

This system uses a “bag of words” method in which the words in messages are directly used as features. This approach was chosen because it is simple but still very popular in natural language processing. Each message is transformed into a word vector which contains one entry for each word in the vocabulary collected from all the messages of the user. Only the contents of the messages is considered: the author, subject, date and other header fields do not affect the result.

Bigrams (two consecutive words) and trigrams (three consecutive words) are also interesting as features. As there are a very large number of different bigrams — and even more trigrams — using all them would not probably give good results. Instead trying to pick only especially interesting bigrams (e.g. first letters capitalized in both words) could be a better approach.

5.2 Pruning, Weighting and Similarity Criteria

If all the words are counted, the vocabulary and consequently the dimension of the word vectors become large. Therefore the most common and rarest words are often excluded. Finding a good balance in this area is difficult. Many rare words are just typing errors or fancy expressions which can be found in totally unrelated messages, but some are the most important keywords of the data.

The implementation accepts a few command line switches to control the pruning of the vocabulary during the calculation process. In the majority of tests rather conservative settings were used, skipping words that can be found in only one message or in more than 5 percent of all messages.

After building the word vectors the words are weighted by the "term frequency $*$ inverse document frequency" (tfidf) factor, more specifically $weigth = n * ln(msgs/ntot)$, where n is the number of occurrences of the word in current message, $ntot$ is the total number of occurrences of the word in all messages and $msgs$ is the total number of messages. This emphasizes the role of rare keywords. Finally, the word vectors are normalized to unit length to eliminate the difference between short and long messages.

The criteria of two messages being similar is simply the dot product of their word vectors: the higher the result the greater the number of shared words (keywords emphasized by the weights) and thus greater similarity. Duplicates are detected and included only once in the results. Another simple but popular measure would be the euclidean distance of vectors, in which case the most similar messages would have the smallest distance between each other.

5.3 Email Specific Issues

The large variety of client software and human habits pose many problems when trying to parse and analyze email messages. The standards which define the format [4] [5] are rather complex and it is common that some clients occasionally break the standards.

5.3.1 Encodings and Attachments

Most messages contain only plain text which can be fed directly to the lexer of the analysis library. Some mailers use quoted printable encoding to send 8-bit characters as 7-bit data. This is not an issue since converting back to 8-bit is easy.

Attachments are much more problematic. Binary files and images would not contribute to the textual data, but quite often also text is sent with special typesetting in html, portable document format (pdf) or even in a proprietary format such as a Microsoft Word document. This results in some messages or important parts of them being left completely out of the analysis.

The implementation described in this document identifies MIME attachments but does not try to analyze any other than plain text, all others are dropped. Unfortunately some mailers don't even adhere to the MIME extensions but send attachments uuencoded or binhex encoded inside the body of the message. These are interpreted as plain text and result in bogus words being added to the vocabulary. This does not do much harm to the similarity comparisons but makes the vocabulary larger and thus the program run slower.

5.3.2 Multiple Languages and Character Sets

Multiple languages are another problem. The language of the message cannot be easily deduced and therefore it is difficult to know which bytes should be interpreted as alphabetic characters. Currently this is implemented by C library functions which use the locale settings (`LC_CTYPE` environment variable) installed on the machine. This has the disadvantage that messages written in other languages may be processed incorrectly and words containing special characters to be split in several words.

A more flexible approach could be better: in the majority of languages words could be separated by looking for spaces and punctuation characters. However, several Asian languages would require completely different processing so solving the problem well would need more work.

Another still far more complicated problem is stemming, finding the base forms of the words. For English there are free implementations of the Porter stemming algorithm with produces acceptable results[4] but

[4]Actually the Bow library implements Porter stemming so it would be easy to add it to this program, but currently it is not used.

many other languages are more difficult. Finnish grammar in particular has a lot of verb forms and suffixes which are usually added at the end of nouns when one would use a preposition in English. Sometimes even the base of the word changes a bit. This results in for example “Helsinki” (the capital of Finland), “Helsinkiin” (to Helsinki), “Helsingissä” (in Helsinki), and “Helsingistä” (from Helsinki) all be counted as different words. There are working implementations of stemming for Finnish but they are only available for a licensing fee.

The current implementation accepts the following character sets:

- us-ascii
- iso-8859-*
- windows-125*

Messages using other character sets are not included in the search. In addition to languages which cannot be written in these character sets, this choice excludes all messages using the universal UTF-8 character set. UTF-8 is not yet very common in email but becoming more popular because it can be used for any language. This is a nice property but to build the vocabulary properly the system would have to use some kind of heuristics to detect the language used in the message.

5.3.3 Quotes, Signatures and Other Redundant Data

When replying to an email it is customary to quote parts of the original message. This is favorable for the search, because usually the original message and the reply are about the same topic. The quotes make them share a large number of words, therefore placing the two messages near each other also in the search results.

On the other hand, signatures have an undesirable effect. Many authors have their address and perhaps their favorite short phrase written at the end of every email. Since the signatures are identical, all words in them show up as matches. This makes the search return texts from the same author as similar even when the message is about a completely different topic than the one used as the search key.

It is advisable to separate the signature from the rest of the message using the delimiter `"-- "`, that is two dashes, one space and one newline. Signatures using the standard delimiter are easy to strip out[5]. Still, many people do not use any delimiter which makes it more difficult to detect their signatures.

Another common type of redundant data are footers which are automatically inserted by a mailing list or a mail service such as Hotmail. Like the signatures these are relatively complicated to detect automatically. While footers from a mailing list can sometimes even be useful (two messages from the same list are likely to be somewhat similar) the advertisements added by mail services only confuse the search — two different messages from Hotmail which contain the same advertisement are not very likely to have similar content.

6 System Performance

6.1 Test Sets

The system was evaluated using emails received by the author during the last couple of years. The following test sets were used:

[5]The current implementation does not try to detect signatures but it would not be difficult to add.

Number	Type	Number of messages	Size (kB)
1	Various personal received messages	500	2816
2	Various personal received messages	2500	13676
3	Messages from five mailing lists	2500 (500 from each list)	8628
4	Messages sent by the author	1000	7648
5	Combination of sets 2, 3 and 4	6000	29952
6	Personal (received + sent) and lists	20294	80312
7	Personal (received + sent) and lists	57023	246616

Table 1: Test sets for the search

The mailing lists in set 3 were Bugtraq (computer security related announcements and discussion), Finnish Linux User Group list (questions and answers to problems related to Linux, announcements of the association), NorduGrid list (discussion about Grid software development), Otakut list (discussion about Japanese animated films) and Finnish Rave Info (club announcements, discussion about parties and electronic music).

Most tests were done using the combined set 5. The idea was to simulate a subset of a personal email archive. The sets 6 and 7 were direct extracts from the author's `old mails` directory tree — most of which is not very nicely organized.

6.2 Relevance of the Retrieved Documents

The similarity of two messages is a highly subjective value. It also depends on the occasion which messages are the most relevant for the reader. Therefore the evaluation of the system was performed by trying different parameters for the vocabulary pruning and then performing searches using different source messages. From the system log it was possible to see which keywords were found in each email included to the similar messages list and the weights of the words.

Messages belonging to the same thread were usually found as similar because they often contain parts of each other as exact quotes. Also messages from the same author were emphasized, especially if the author had a long signature at the end of the message. In the latter category the system often brought up texts from completely unrelated topics, particularly short messages in which each word has more weight.

The test set 5 (6000 messages) contained a bit over million words in total (headers and binary attachments excluded), and more importantly 89057 different words. Over half of them were present in only one message thus giving no input to similarity. Excluding them already reduced the vocabulary to 44195 words. Another 14809 words were present in only two messages. This group contained a lot of words with typing errors and strange character sequences but also important keywords. Pruning rare words more aggressively reduced running time but produced inferior results, especially if excluding words present in more than 3 messages.

On the other hand it was safe to exclude words that were present in more than a few percent of all messages. This did not significantly reduce the size of vocabulary, for example pruning all words in more than 5% of the messages resulted in 266 words and those in more than 2% of the messages in 880 words to be dropped. Cutting aggressively reduced the effect of emphasized signatures (if there were enough messages from one author his signature was completely ignored), but also excluded important keywords. Most tests were done dropping the words either in only one message or in more than 5%, giving a vocabulary of 43929 words.

It was clear that the system would benefit from stemming, now matching keywords were not found in many cases because they had a different ending. In cases of only a few matching words, short messages were perhaps emphasized a bit too much, sometimes overriding a more relevant longer message. Otherwise the weighting scheme seemed to have a good balance to emphasize rare words but not ignoring others.

Choosing the keywords more intelligently could certainly improve the results, now typing errors and rare adjectives are in the same category than real keywords. However, despite the described weaknesses the system seemed to find relevant messages well enough so that it could be helpful in real use.

6.3 Running Time

The performance tests were done on a standard PC equipped with a 350 MHz Pentium II processor, 256 MB of memory and 6GB IDE disk. The operating system was Debian GNU/Linux version 3.0, Linux kernel version 2.2.19. Only a few iterations per test were done and no special precautions were taken to eliminate caching and other disturbances, so the results presented here should be taken only as rough estimates.

A summary of the results is presented in table 2. When running the `dbmail-organize` program, words were excluded from the vocabulary if they occurred either in only one or in more that 5 % of all messages. In search commands, several different source messages were used and 20 similar messages were retrieved in each case. Vocab. is the number of words in the vocabulary after pruning and all times are measured in seconds.

Set	**Msgs**	**Size**	**Vocab.**	**Organize**		**First search**		**Cached search**	
		(kB)		**user**	**elapsed**	**user**	**elapsed**	**user**	**elapsed**
1	500	2816	7044	8	16	0.2	0.6	0.02	0.03
2	2500	13676	24410	32	68	0.9	1.9	0.04	0.07
3	2500	8628	21080	21	51	0.7	1.7	0.03	0.06
4	1000	7648	11946	10	25	0.4	1.0	0.02	0.04
5	6000	29952	43929	64	190	2	4	0.1	0.2
6	20294	80312	86761	220	780	7	33	0.5	1.0

Table 2: Running times for different test sets

Running the organizing process for the largest mail set (number 7) turned out to be unreliable. The processing was stopped after running for about 20 minutes — the DBMAIL software seemed to have a problem extracting the message data from the database for an unknown reason (two separate test cases, different failures in each). Over half of the processing was already done in both cases and the memory consumption was about 100 megabytes. Organizing and searching in the test set 6 also required a considerable amount of memory, about 40 MB.

The elapsed time (wall clock time) was considerably longer than user time (processor time used by the program) in all cases. The reason is that part of the time was taken by the MySQL database and disk I/O, especially with larger message sets. Because the system was not used for other purposes during the test, the elapsed time is a much more relevant estimate here.

The time in the organizing process is roughly divided in two: building the vocabulary and building the word vectors. In both phases, all messages are retrieved sequentially from the database and processed. The rest of the operations (setting weights, normalizing etc.) take only a small portion of the time. The key factors which affect the performance are the total number of bytes and the number of messages; the latter one seems to be a bit more decisive. As expected, the time seems to grow roughly linearly compared to the number of messages.

The theoretical complexity of the search is $O(n * m)$, where n is the number of messages and m is the average length of the word vectors. During the first search, the vocabulary and word vectors are read from the database in the memory and cached for subsequent searches during the same connection. The first search is considerably slower than subsequent ones, which shows that the majority of time is used in reading the vocabulary and word vectors from the database.

It should be noted that the time measurements for the cached searches for all datasets varied considerably (the result in some cases could be half or double of that presented in the table). However, the results for the first search were more consistent which is the more interesting case when evaluating the usability of the program.

The performance is adequate for small message archives up to a few dozen megabytes, but with large collections both the CPU and memory requirements become too high. Because vocabulary and word vector data are separate for each user, it would be easier for a server to handle several users with a moderate amount of messages than one user with a large archive[6].

6.4 Optimization Possibilities Using a Self-Organizing Map

Self-organizing maps (SOMs) are a technique which can automatically adapt a set of neurons on a two dimensional plane to approximately match a data set of larger dimension. When the map is organized, similar input vectors are placed near each other on the map.

This algorithm was examined as one possibility to optimize the search. The idea was to store map coordinates of each message in the database so that the search function would have to look only at messages in nearby coordinates. The area could be selected efficiently using a single SQL statement. The map could also be used to visualize the emails as done in the WEBSOM project [22]. Two packages were evaluated: The SOM Toolbox for Matlab [13] and the SOM_PAK [15] package.

The SOM Toolbox is easy to use and has good visualization capabilities, but it was too slow when using data vectors of a large dimension. Word vectors of 1000 words from 250 messages took already more than 10 minutes to process on a 800 MHz AMD computer. The number of messages did not seem to be the problem, the large dimension of the vectors was more difficult.

The SOM_PAK performed fast enough to do some small scale tests. The word vectors of the email test set 1 (vocabulary 7044 words after pruning) were extracted from the DBMAIL database and maps of size $12*8$ and $24*12$ were generated. Then three sets of messages, original and 10 similar ones in each set[7], were plotted on the maps.

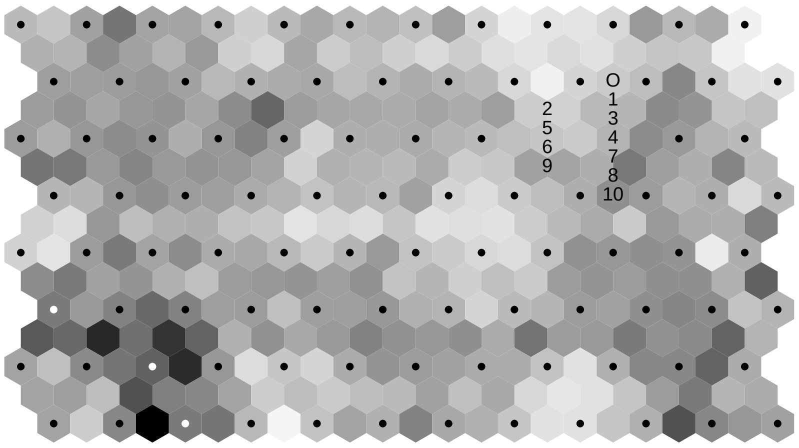


Figure 3: Original and 10 similar messages plotted on a self-organizing map.

Most of the messages in each group were placed near each other (see figure 3 for an example), and the groups on completely different parts of the map. The results on small and big maps were similar in nature:

[6]An estimate based on knowledge about the algorithm, not actual test results. The system has currently only been quickly tested with two concurrent users.

[7]The originals were chosen by hand and the similar messages found using the brute force search.

most messages were near each other but a few were placed further away — interestingly in some cases different messages on the small than on the big map.

Overall, the algoritm seemed to map similar vectors near each other decently, at least in this small set of messages and three examples. Unfortunately the performance was not adequate for bigger message sets. The small map for 500 messages took already several minutes to process and the set of 6000 messages was completely out of reach. Even the data file of the word vectors for the 6000 message set would have been about 450 megabytes (6.5 MB for the set of 500 messages) because the format accepted by SOM_PAK contains all the elements of the vectors, most of which are zero. In the Bow format only non-zero elements are stored.

The WEBSOM system has been used to organize very large text archives. Perhaps some of the advancements and optimizations found in that project or completely different algorithms would be a better choice to improve the performance of this system too.

7 Conclusions and Future Work

The system described in this paper is an unfinished prototype but it can already be used as a test platform. After smoothing some rough edges in the implementation it would be interesting to take the system in real use to see over a few months if it helps even a little bit in managing old email. The performance with large message archives is probably the biggest currently unsolved issue.

If the system is developed further, the syntax of the XSIMILAR command should be carefully redesigned. It should not require complex functionality from the client but allow more detailed control for those clients who wish to use it. The client should be allowed to add it's own search keywords and limit the list of retrieved messages using various criteria (e.g. those used in the IMAP SEARCH command). The server could also offer a way to control the method and parameters used for the organization process from the client.

There are many opportunities for future research related to language processing. The system provides a convenient platform to test more advanced algorithms in organizing emails — the message blocks can be easily retrieved and the database can also be used to neatly and efficiently store the classification data.

The software described in this document can be downloaded from `http://wikihip.cern.ch/twiki/bin/view/Grid/SimilarEmailSearch`.

References

[1] Helsinki Institute of Physics Technology Program. `http://www.hip.fi/research/technology/`. [Cited September 23, 2002.]

[2] Internet Engineering Task Force, Network Working Group, 1996. Request for Comments number 1939: Post Office Protocol - Version 3. `http://www.isi.edu/in-notes/rfc1939.txt`. [Cited September 23, 2002.]

[3] Internet Engineering Task Force, Network Working Group, 1996. Request for Comments number 2060: The Internet Message Access Protocol - version 4rev1. `http://www.isi.edu/in-notes/rfc2060.txt`. [Cited September 23, 2002.]

[4] The Internet Society, 2001. Request for Comments number 2822: Internet Message Format. `http://www.isi.edu/in-notes/rfc2822.txt`. [Cited September 23, 2002.]

[5] Internet Engineering Task Force, Network Working Group, 1996. Requests for Comments numbers 2045, 2046, 2047, 2048, 2049: Multipurpose Internet Mail Extensions (MIME). The following page lists online links and resources to these and related documents: `http://www.oac.uci.edu/indiv/ehood/MIME/MIME.html`. [Cited September 23, 2002.]

[6] Open Source Initiative, 2002. The Open Source Definition, version 1.9. `http://www.opensource.org/docs/definition.php`. [Cited September 23, 2002.]

[7] DBMAIL, a group of programs for storing and retrieving email in a SQL database. `http://www.dbmail.org`. [Cited September 23, 2002.]

[8] GNU General Public License, version 2, June 1991. `http://www.gnu.org/licenses/gpl.html`. [Cited September 23, 2002.]

[9] McCallum, Andrew Kachites, 1996. Bow: A toolkit for statistical language modeling, text retrieval, classification and clustering. `http://www.cs.cmu.edu/~mccallum/bow/`. [Cited September 23, 2002.]

[10] GNU Library General Public License, version 2, June 1991. `http://www.gnu.org/licenses/lgpl.html`. [Cited September 23, 2002.]

[11] The Mutt E-Mail Client. `http://www.mutt.org`. [Cited September 23, 2002.]

[12] Kohonen, T.; Huang, T. S. (Editor); Schroeder, M. R. (Editor). 2001. Self-Organizing Maps (Springer Series in Information Sciences, 30). 3.p. Springer Verlag. ISBN 3540679219.

[13] SOM Toolbox for Matlab. `http://www.cis.hut.fi/projects/somtoolbox/`. [Cited September 23, 2002.]

[14] Matlab software. `http://www.mathworks.com/products/matlab/` [Cited September 23, 2002.]

[15] SOM_PAK - The Self-Organizing Map Program Package. `http://www.cis.hut.fi/research/som_lvq_pak.shtml`. [Cited September 23, 2002.]

[16] Helsinki Institute of Physics, 2002. Metadata Visualisation Project. `http://mdv.sourceforge.net`. [Cited September 23, 2002.]

[17] The BSD License. `http://www.opensource.org/licenses/bsd-license.php`. [Cited September 23, 2002.]

[18] The CMU-Cambridge Statistical Language Modeling Toolkit. `http://svr-www.eng.cam.ac.uk/~prc14/toolkit.html`. [Cited September 23, 2002.]

[19] GATE - General Architecture for Text Engineering. `http://gate.ac.uk/`. [Cited September 23, 2002.]

[20] Ifile mail filtering system. `http://www.ai.mit.edu/~jrennie/ifile/`. [Cited September 23, 2002.]

[21] OpenNLP, an organizational center for open source projects related to natural language processing. `http://opennlp.sourceforge.net/`. [Cited September 23, 2002.]

[22] WEBSOM - Self-Organizing Maps for Internet Exploration. `http://websom.hut.fi/websom/`. [Cited September 23, 2002.]

[23] MySQL database. `http://www.mysql.com/`. [Cited September 27, 2002.]

[24] Jurafsky, D.; Martin, J. 2000. Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics and Speech Recognition. 1.p. Prentice Hall. 960 s. ISBN 0130950696.